COMP1010 Fundamentals of Computer Science

Assignment 3

Last updated: April 26, 2021

Assignment Details

- The assignment will be marked out of 100 and it is worth 10% of your total marks in COMP1010.
- Due date: Sunday, 16 May 2021 at 21:00 AEST.
- Late penalty: 20% per day or part of (if you submit an hour late, you will be given a 20% penalty).
- Topics assessed: ArrayList, Classes and Objects

Background

In this assignment, you are going to write an implementation of a classic tile-matching game called SameGame, which is arguably the precursor to more well-known tile-matching games such as Bejeweled and Candy Crush. It does not matter if you have not played any of these games, because the rules of SameGame are actually quite simple. In a game of SameGame, you are given a rectangular board filled with tiles of different colours, such as the one shown in Figure 1.

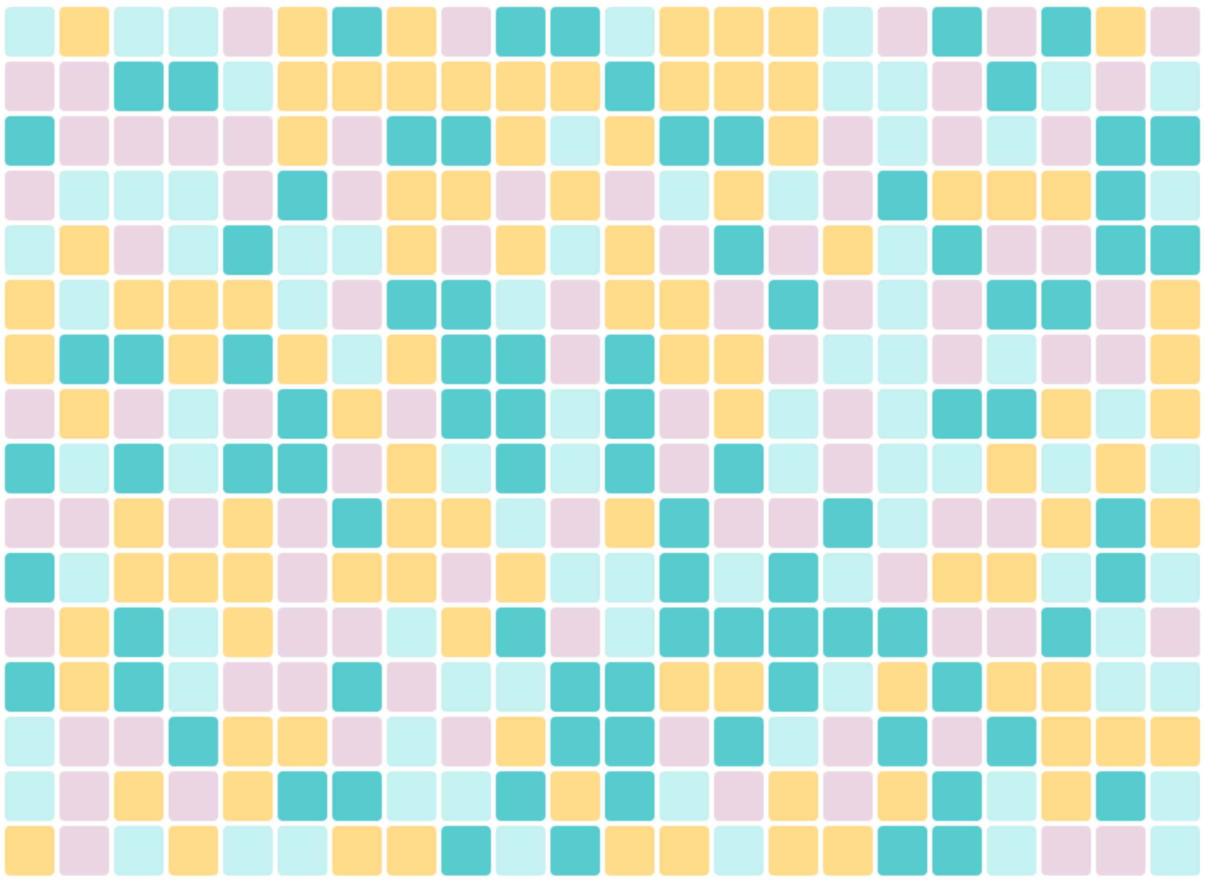

Figure 1: Example of a SameGame game.

The goal of the game is to remove as many tiles as you can from the board. A tile can only be removed if it is adjacent to another tile of the same colour, where adjacent here means either above, below, to the left, or to the right.

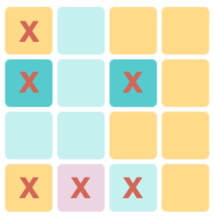

Figure 2: Example of a board with non-removable tiles (marked with a red cross).

For example, in Figure 2, the yellow tiles on the top-left and bottom-left corners are not removable since they do not have any adjacent tiles of the same colour. The same applies for the dark-green and purple tiles. The light-blue tile in the bottom row has a tile of the same colour nearby, but it is not adjacent, so it is not removable either. All the other tiles are removable.

When a tile is removed, all tiles with the same colour which are adjacent to it are also removed. For example, if we were to remove any of the yellow tiles on the third or fourth column, then every other yellow tile in those columns will be removed, because each of them are adjacent to a tile that is removed. You can see how this works in Figure 3.

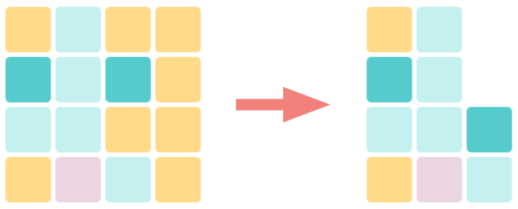

Figure 3: Example of removing a tile (a yellow tile in the third or fourth column).

Once a tile is removed from the board, any tile above it are no longer supported and will move down to take the empty space(s), as you can see with the dark-green tile in the third column. If we proceed by removing the light-blue tile in the first or second column, then the yellow and dark-green tiles on the top left corner will move down, as shown in Figure 4.

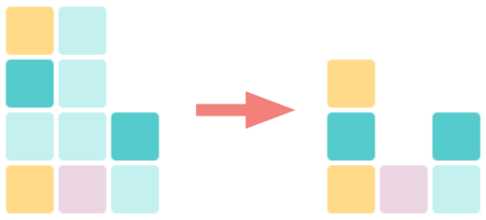

Figure 4: Removing the light-blue tiles after removing the yellow tiles.

At this point, the game ends because all the remaining tiles cannot be removed. A completely empty column (or columns) will be removed from the board and all columns to the right of the removed columns will shift to the left. You can see an example of this in Figure 5 after we remove the purple tiles, and then again after removing the light-blue tiles in the second column. Note that it is possible to remove multiple columns at the same time.

The player's **score**, given at the end of the game, is simply the number of tiles that were removed from the board, so if a player manages to remove 10 tiles, then the player's score is 10. If follows that the maximum score is the number of tiles at the beginning of the game, therefore a player can achieve the maximum score only if the player manages to remove all the tiles from the board.

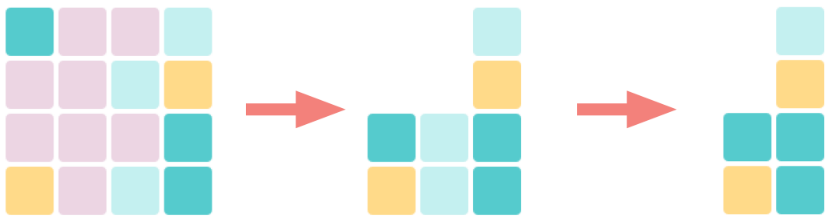

Figure 5: Removing purple tiles, followed by removing light-blue tiles.

In summary, here are the three main rules of the game:

1. Tiles can be removed when they are adjacent to other tiles with the same colours, and all adjacent tiles with the same colour are removed together as a group.
2. Tiles on the board will move down to take up any empty spaces below it (i.e. when we remove the tiles below them).
3. Columns with no tiles will be removed from the board, and the remaining columns to its right will take their place.

It is important to note that tiles will not move leftwards unless a column is completely empty, so you should always move a tile downwards first before moving it leftwards.

SimpleSameGame

To get you started with the assignment, I added a simpler version of the game with only one row, which I am going to refer to as SimpleSameGame. The implementation for SimpleSameGame is easier, which is why I had it written as a different class (more on this later).

Figure 6: Example of a SimpleSameGame game.

In this version, you do not have to worry about tiles falling down when it is no longer supported. Whenever a tile is removed, you simply have to move all adjacent tiles with the same colour (this is equivalent to removing a column in SameGame).

Your Task

Your task is to complete all the methods in two files which control the logic of the games explained above:

- `SimpleSameGame.java` (worth 29 marks)
- `SameGame.java` (worth 71 marks)

You can find the exact marks allocated for each method that you have to implement in Grade Distribution section further down in this document. The details of what each of these methods do can be found in the code template give to you.

Before you get started though, there are a few more things worth discussing here. First of all, you are given the `Tile.java` class which represents the coloured tiles. You **should not modify** `Tile.java` since it is not part of your submission.

The board for a game of SimpleSameGame is represented using an `ArrayList<Tile>`, and in the case of SameGame, using an `ArrayList<ArrayList<Tile>>`. The implementation for SimpleSameGame is quite straightforward, so I will leave it to you to work out. However, in the case of SameGame, we must establish the following rules:

- rows are numbered from 0, starting from the bottom-most row
- columns are numbered from 0, starting from the left-most column

So, supposing that the game board is represented by the variable `board`, then to access the tile on the bottom-left corner, you need to call

```
board.get(0).get(0)
```

You can see Figure 7 to see how the board should represented.

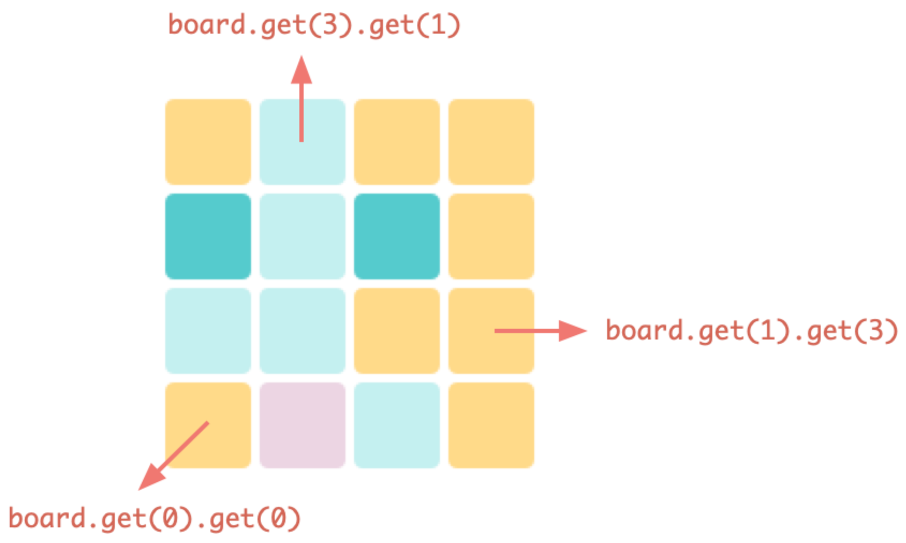

Figure 7: Accessing the tiles on the board

You should create the board when the constructor for SimpleSameGame or SameGame is called. I have implemented the constructor for the SimpleSameGame class, and one of the task you need to complete is to use that as a basis to create the constructor for the SameGame class. Both constructors use a random number generator which is given to you (`Random rgen`). Here is the constructor for SimpleSameGame:

```
public SimpleSameGame(Random rgen, int size) {

  // create an ArrayList of four different Color objects
  ArrayList<Color> colors = new ArrayList<Color>();
  colors.add(Color.decode(ColorDef.LIGHTBLUE));
  colors.add(Color.decode(ColorDef.LIGHTPURPLE));
  colors.add(Color.decode(ColorDef.PEACH));
  colors.add(Color.decode(ColorDef.TURQOUISE));

  // initialise the ArrayList representing the board
  board = new ArrayList<Tile>();
  for(int i = 0; i < size; i++) {
    // get a random number from 0 to colors.size()-1
    int current = rgen.nextInt(colors.size());
    // create a tile with a random colour from the colors array
    board.add(new Tile(colors.get(current),1));
  }
}
```

The constructor starts by adding four `Color` object into the ArrayList `colors` (this part is already written for you in the SameGame constructor). It then creates an instance of `Tile` class for each tile, and add it to `board`. Each tile is given a random colour by calling `rgen.nextInt(colors.size())` to generate a random number between 0 and 3, and then picking that colour from `colors` (the ArrayList containing `Color` objects). The constructor also assigns 1 to the score of the tile, but this is optional (it may help in implementing the other methods). In the constructor for SameGame, you must add the tiles **one row at a time starting from the bottom row**. In other words, if you were asked to make a board of size 4×10, then randomly create 10 tiles for the bottom row, then randomly create another 10 tiles for the row above that, and so on. Remember the rule above for row and column numbering.

Once a tile in a row is removed, it should be replaced by the tile directly above it (i.e. in the next row), or replaced by the null if none exist. You should not remove an empty row from `board`, and so it is expected to have a row (or rows) of nulls at some stage of the game, as shown in Figure 8. You also should not simply delete a tile from a row because this will shift the remaining tiles in the row to the left. You should only move tiles to the left when all the tiles in a column have been removed.

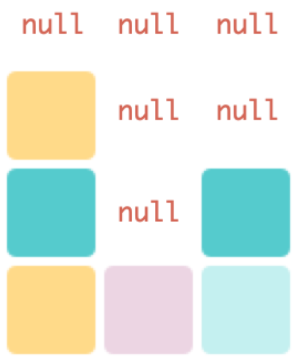

Figure 8: Replace removed tiles with nulls.

You need to implement and then call the method `rearrangeBoard()` to move the tiles downwards. The method `trickleDown(ArrayList<Tile> bottom, ArrayList<Tile> top)` is meant to help you in doing so, but you are free to write `rearrangeBoard()`jk without using `trickleDown()` (you would still have to implement `trickleDown()` to get the marks though). Once a column is empty, you also need to shift some columns to the left, and technically this should be part of `rearrangeBoard()` as well, HOWEVER, since removing columns takes a bit more effort, I have decided that you should write another method to handle this.

The method that you should implement for removing empty columns is `deleteEmptyColumns()`. Once you have completed this method, you should call it from `rearrangeBoard()` (because, as I said earlier, deleting empty columns is part of rearranging the board). The JUnit test for `deleteEmptyColumns()` actually calls `rearrangeBoard()` instead of directly calling `deleteEmptyColumns`, so please make sure that you add a call to `deleteEmptyColumns()` from `rearrangeBoard()`. The JUnit test for `rearrangeBoard()` only checks if you can move the tiles downward, so you can still get full marks for `rearrangeBoard()` even if you did not manage to complete `deleteEmptyColumns()`.

Visualisation

To help you with this assignment, I have also added a simple visualisation of the game which you can use to check if you have implemented the game logic correctly. All the files can be found in the package `graphics`. To run the game, you can run either `SimpleSameGameBoard.java` or `SameGameBoard.java`, depending on which version of the game you would like to run.

The visualiser is very simple; it creates either a SameGame or a SimpleSameGame instance, and draw the board (i.e. the ArrayList or ArrayList of ArrayLists) on the board. Every time a player click on a tile, it will call the relevant methods from the SameGame or SimpleSameGame class, and then requests a new board to display (this is done in the `paintComponent()` method). The only part of the code that you may want to understand is this one, inside `SameGameBoardTile.java`:

```
// add behaviour to each tile when clicked (we use mouseReleased since
// this is actually better at detecting clicks
public void addMouseListeners() {

  this.addMouseListener(new MouseInputAdapter() {
    @Override
    public void mouseReleased(MouseEvent e) {
      // check if the clicked tile is a valid tile to remove,
      // and remove it if it is
      if (board.game.isValidIndex(row, col) && board.game.isValidMove(row,col)){
        board.game.removeTile(row, col);
        board.repaint();
      }
      // if there are no more valid moves, then calculate the score
      // and inform the board so that it can be displayed
      if (board.game.noMoreValidMoves()) {
        board.score = board.game.calculateScore();
        board.gameFinished = true;
      }
    }
  });
}
```

All the code does is adding a behaviour on each tile so that when the tile is clicked it will call some methods to control the game logic. The methods `isValidIndex(...)`, `isValidMove(...)`, `removeTile(...)`, `noMoreValidMoves()`, and `calculateScore()` are some of the methods that you need to implement so that the game visualiser will work properly. In particular, the method `removeTile` is likely to be the hardest part of the assignment, so if you manage to implement that, you will be rewarded with a fully working, albeit simple, game.

You can change the parameters of the game by modifying the `main` method in either SameGame or SimpleSameGame class, in particular, the line

```
SameGameBoard b = new SameGameBoard(378354,4,10);
```

The first input parameter is the seed for the random number generator, followed by the number of rows and the number of columns for the game. If you initalise a random number generator (the `Random rgen` object mentioned earlier) with a fixed seed, then it will always produce the same sequence of random numbers. Therefore, whenever you start the game with the above call, you should always get the following board:

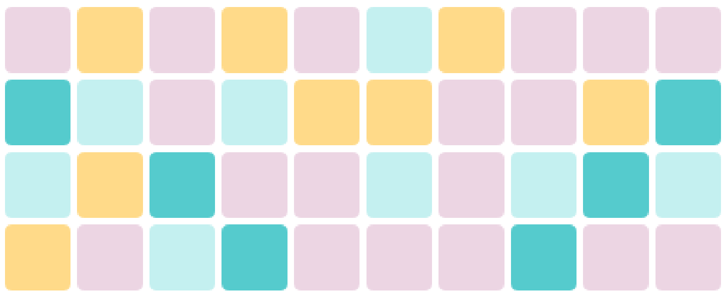

Figure 9: SameGame board of size 4×10 using seed 378354

If you want to have a random board everytime you start the game, then you can replace the above line with

```
SameGameBoard b = new SameGameBoard(System.currentTimeMillis(),4,10);
```

which will use a different value for the random number generator seed every time it starts.

Grade Distribution

Here is the list of methods that you need to implement and their associated marks:

- Pass Level (64 marks):
  - in `SimpleSameGame.java`:
    - (4 marks) — `getBoard`
    - (5 marks) — `isValidIndex`
    - (5 marks) — `isValidMove`
    - (5 marks) — `noValidMoves`
    - (10 marks) — `removeTile`
  - in `SameGame.java`:
    - (5 marks) — Class constructor
    - (5 marks) — `calculateScore`
    - (5 marks) — `getBoard`
    - (5 marks) — `isValidIndex`
    - (10 marks) — `isValidMove`
    - (5 marks) — `noMoreValidMoves`
- Credit Level (10 marks):
  - in `SameGame.java`:
    - (5 marks) — `trickleDown`
    - (5 marks) — `rearrangeBoard`
- Distinction Level (10 marks):
  - in `SameGame.java`:
    - (10 marks) — `deleteEmptyColumns`
- High Distinction Level (16 marks):
  - in `SameGame.java`:
    - (16 marks) — `removeTile`

JUnit Tests

As in previous assignments, you are given a JUnit test (`UnitTest.java`) which you should use to ensure that your code meets all the requirements. The assignment will be automarked using a similar test suite. Obviously the values used in the tests will be modified when your assignment is being marked, but the tests themselves should be very similar.

For the `removeTile` method in `SameGame.java`, there is a test with a large board (1000×1000) that you must finish within 10 seconds in order to get full marks.

Submission Format

Please submit both `SimpleSameGame.java` and `SameGame.java`. You need to submit both files unzipped (i.e. do not zip them together), or you will be given a zero.

Penalties

Similar to the other assignments, there are some other strict conditions that you must follow:

- Any compilation errors and/or infinite loops in any of the submitted files will result in an automatic zero.
- Submissions must be self-contained and does not require any other file besides `Tiles.java` (and the JUnit test files). Any dependency on an external file (e.g. using functions from outside the submitted files) will result in an automatic zero.
- Changing any package declaration in any file will result in an automatic zero.
- Adding any import statement in any file will result in an automatic zero.
- Modifying the visibility of any methods or attributes in the code template will result in an automatic zero (please leave everything as public so that the JUnit tests can access them directly).

Change Log

If there are any changes to this documentation after its release, it will be listed here. Please check to ensure you have read the latest documentation. Any changes will be announced on iLearn.