Victoria University of Wellington
School of Engineering and Computer Science

SWEN221: Software Development

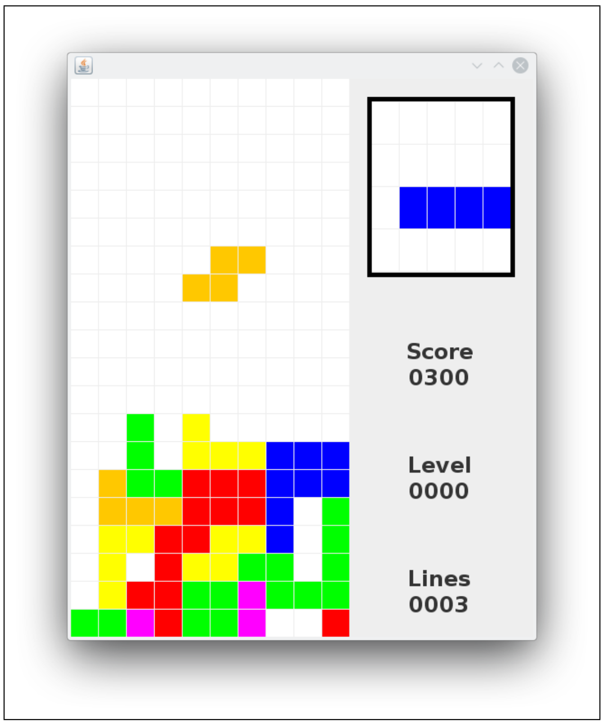
Assignment 2 (worth 8%)

Due: Monday 11th May @ 23:59

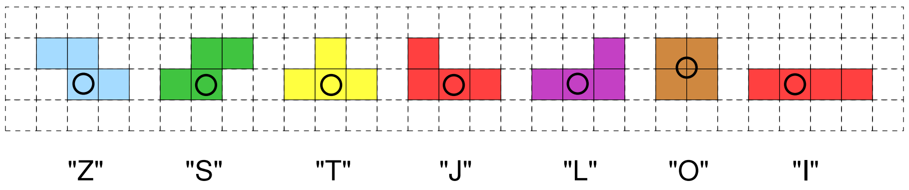
This assignment will expose you to a more complex class hierarchy. A key challenge will be in understanding the flow of control through the program, particularly as there is extensive use of polymorphism. This assignment will also test your debugging skills, and you should strive to be methodical in how you approach debugging.

Tetris

This assignment concerns the classic game of “Tetris”, originally developed by the Russian programmer Alexey Pajitnov. The tetris pieces (called *Tetromino*’s) are controlled using the keyboard (in this case, the arrow and space keys). As is usual in the game, the pieces are subject to “gravity” meaning that they slowly descend downwards. Pieces can be made to descend more quickly using either the *down arrow key* or much more quickly using the *space key*. Likewise, pieces move left or right, and rotate clockwise using the *up arrow key*. The code includes a graphical user interface so you can play it:



In Tetris, there are seven tetrominoes:



Each tetromino has an *axis of rotation* illustrated above using a circle and an *orientation* (i.e. *North, East, South, West*). The tetromino above are illustrated in their *North* orientation. Finally, for more information on the game of Tetris, refer to `https://en.wikipedia.org/wiki/Tetris`.

Getting started

To get started, download the `tetris.jar` file from the lecture schedule on the course website. As usual, you can run the program from the command-line as follows:

```
java -jar tetris.jar
```

A simple GUI should appear on your screen, and you should be able to play the game. *Remember, however, that at this stage the game contains a number of bugs and missing features.* For example, pieces move in the opposite direction from what you are expecting!

Understanding the Code

Each piece is implemented as a separate class which extends the abstract class `Tetromino`. When you import the tetris.jar file, you should find the following Java packages:

- The `swen221/tetris/gui/` package contains the graphical user interface and the main method. **You do not need to understand the inner workings of this in order to complete the assignment. NOTE:** you do not need to modify any code in this package.
- The `swen221/tetris/logic/` package contains the class `Game` encoding the logic of a game of Tetris, and the class `Board` representing the current state of the board.
- The `swen221/tetris/tetromino/` package contains the interface `Tetromino` and the concrete implementations for all tetromino.
- The `swen221/tetris/moves/` package contains a class for each of the different kinds of move that can be made in the game. These contain code related to structuring a move, and ensuring it is valid.
- The `swen221/tetris/tests/` packages contains many jUnit tests to check your implementation of the game. **NOTE:** To make the automatic marking possible, you can not modify the files already present in this folder, but you may add your tests in a separate file (e.g. `MyTests.java`).

Part 1 — Simple Moves (20%)

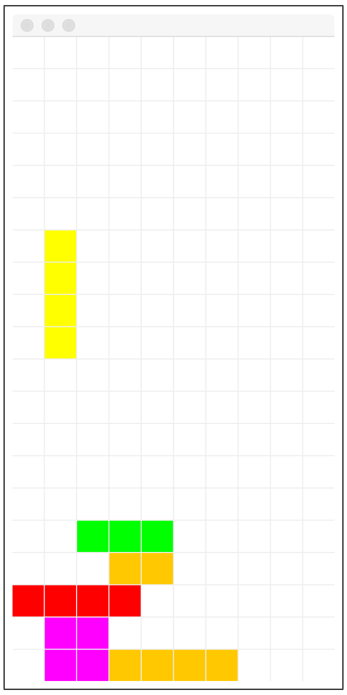
The first objective is to make sure the game correctly recognises all of the simple moves that a piece can make. Specifically, this part concerns mostly minor problems in the tetromino implementations (e.g. J_Tetromino). At this stage more complex moves, such as for remove lines and ending the game, can be ignored. A suite of tests for this part is provided in `swen221/tetris/tests/Part1Tests.java`. You can run these tests individually, or all together. **You are advised to write additional tests to ensure the system is working correctly.**

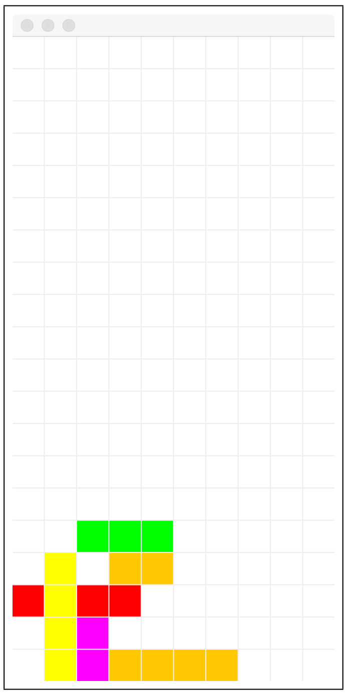
Part 2 — Landings (20%)

The second objective is to implement the mechanism for *landing* pieces. In particular, this includes the "drop move" where a piece descends immediately from its current place. When a piece is landed, the next turn continues. A suite of tests is provided in `swen221/tetris/tests/Part2Tests.java`. **You are advised to write additional tests to ensure the system is working correctly.**

Part 3 — Invalid Moves (20%)

The third objective is to make sure the game correctly recognises when a move is invalid. For example, a piece cannot be moved off the side of the board. Likewise, a piece cannot be dropped below the bottom of the board, or into the middle of another piece. The following illustrates such a situation, where one piece is landed in the middle of two others:

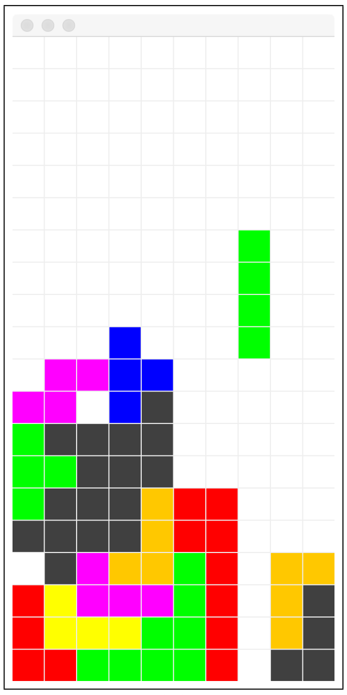

(1)

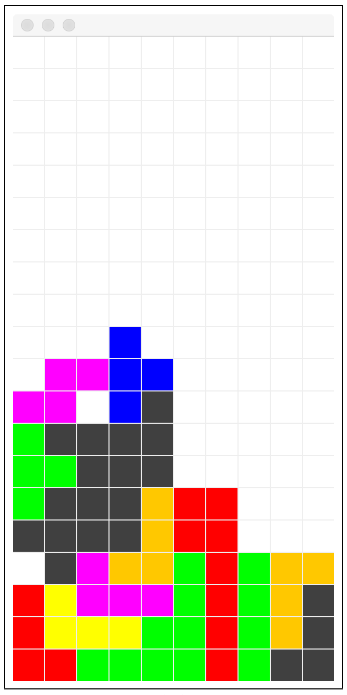

(2)

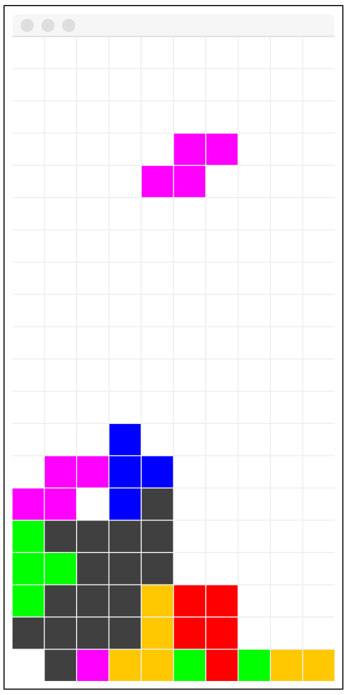
A suite of tests for this part is provided in `swen221/tetris/tests/Part3Tests.java`. You can run these tests individually, or all together. **You are advised to write additional tests to ensure the system is working correctly.**

Part 4 — Line Removal (30%)

The final objective is to correctly handle line removal rows when a tetromino has been "landed". The following illustrates the a tetromino being landed and a number of full rows being removed:


(1)


(2)


(3)

Here, we see that *three full rows* are removed when the `I` tetromino is landed. A suite of tests for this part is provided in `swen221/tetris/tests/Part4Tests.java`. **You are advised to write additional tests to ensure the system is working correctly.**

HINT: We recommend that you enable checking of JavaDoc comments for your project. To do this, JavaDoc errors must be enabled by configuring the "Java Compiler"→"JavaDoc" menu, and additionally setting all visibility options to "private" and checking "Validate tag arguments". This ensures JavaDoc comments are provided for all public, protected and private declarations, including methods and fields, and that tags are not malformed, etc.

Submission

Your lab solution should be submitted electronically via the *online submission system*, linked from the course homepage. The minimum set of required files is:

```
wen221/tetris/gui/Tetris.java
swen221/tetris/logic/Rectangle.java
swen221/tetris/logic/Board.java
swen221/tetris/logic/Game.java
swen221/tetris/moves/ClockwiseRotation.java
swen221/tetris/moves/AbstractTranslation.java
swen221/tetris/moves/AbstractMove.java
swen221/tetris/moves/Move.java
swen221/tetris/moves/MoveLeft.java
swen221/tetris/moves/MoveRight.java
swen221/tetris/moves/MoveDown.java
swen221/tetris/moves/DropMove.java
swen221/tetris/tetromino/Tetromino.java
swen221/tetris/tetromino/AbstractTetromino.java
swen221/tetris/tetromino/ActiveTetromino.java
swen221/tetris/tetromino/I_Tetromino.java
swen221/tetris/tetromino/J_Tetromino.java
swen221/tetris/tetromino/L_Tetromino.java
swen221/tetris/tetromino/O_Tetromino.java
swen221/tetris/tetromino/S_Tetromino.java
swen221/tetris/tetromino/T_Tetromino.java
swen221/tetris/tetromino/Z_Tetromino.java
```

You must ensure your submission meets the following requirements (which are needed for the automatic marking script):

1. **Your submission is packaged into a jar file, including the source code**. *Note, the jar file does not need to be executable.* See the following Eclipse tutorials for more on this:

   `http://ecs.victoria.ac.nz/Support/TechNoteEclipseTutorials`

2. **The names of all classes, methods and packages remain unchanged**. That is, you may add new classes and/or new methods and you may modify the body of existing methods. However, you may not change the name of any existing class, method or package. *This is to ensure the automatic marking script can test your code.*

3. **All testing mechanism supplied with the assignment remain unchanged.** Specifically, you cannot alter the way in which your code is tested as the marking script relies on this. However, this does not prohibit you from adding new tests. *This is to ensure the automatic marking script can test your code.*

4. **You have removed any debugging code that produces output, or otherwise affects the computation.** *This ensures the output seen by the automatic marking script does not include spurious information.*

Note: Failure to meet these requirements could result in your submission being reject by the submission system and/or zero marks being awarded.

Assessment

This assignment will be marked as a letter grade (A+ ... E), based primarily on the following criteria:

- **Correctness of Part 1 (20%)** — does submission adhere to specification given for Part 1.
- **Correctness of Part 2 (20%)** — does submission adhere to specification given for Part 2.
- **Correctness of Part 3 (20%)** — does submission adhere to specification given for Part 3.
- **Correctness of Part 4 (30%)** — does submission adhere to specification given for Part 4.
- **Style (10%)** — does the submitted code follow the style guide and have appropriate comments (inc. Javadoc)

As indicated above, part of the assessment for the coding assignments in SWEN221 involves a qualitative mark for style, given by a tutor. Whilst this is worth only a small percentage of your final grade, it is worth considering that good programmers have good style.

The qualitative marks for style are given for the following points:

- **Division of Concepts into Classes**. This refers to how *coherent* your classes are. That is, whether a given class is responsible for single specific task (coherent), or for many unrelated tasks (incoherent). In particular, big classes with lots of functionality should be avoided.
- **Division of Work into Methods**. This refers to how well a given task is split across methods. That is, whether a given task is broken down into many small methods (good) or implemented as one large method (bad). The approach of dividing a task into multiple small methods is commonly referred to as *divide-and-conquer*.
- **Use of Naming**. This refers to the choice of names for the classes, fields, methods and variables in your program. Firstly, naming should be consistent and follow the recommended Java Coding Standards (see `http://g.oswego.edu/dl/html/javaCodingStd.html`). Secondly, names of items should be descriptive and reflect their purpose in the program.
- **JavaDoc Comments**. This refers to the use of JavaDoc comments on classes, fields and methods. We certainly expect all `public` and `protected` items to be properly documented. For example, when documenting a method, an appropriate description should be given, as well as for its parameters and return value. Good style also dictates that `private` items are documented as well. **HINT:** it is highly recommended that you turn on JavaDoc checking in Eclipse as this will help ensure better quality comments.
- **Other Comments**. This refers to the use of commenting within a given method. Generally speaking, comments should be used to explain what is happening, rather than simply repeating what is evident from the source code.
- **Overall Consistency**. This refers to the consistent use of indentation and other conventions. Generally speaking, code must be properly indented and make consistent use of conventions for e.g. curly braces.

Finally, in addition to a mark, you should expect some written feedback highlighting the good and bad points of your solution.