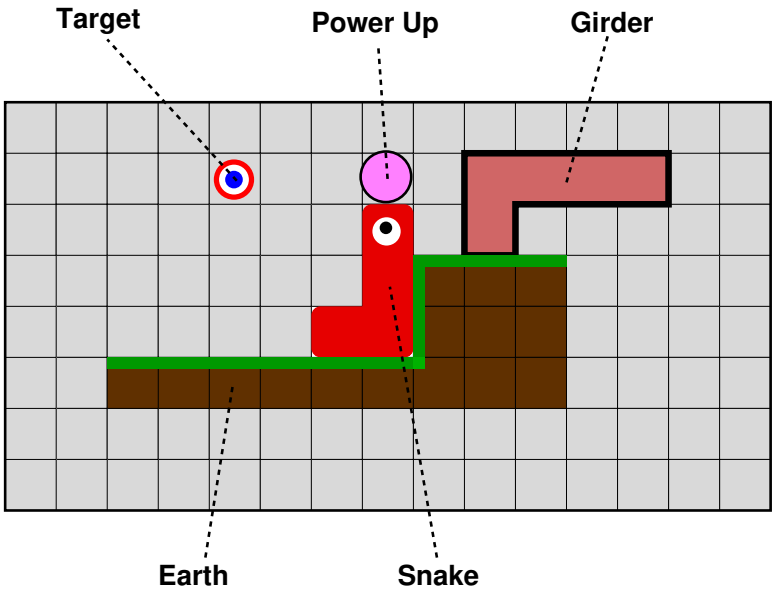
Victoria University of Wellington, School of Engineering and Computer Science

SWEN221: Software Development

Test 1 (worth 20% of overall mark)

SnakeFall

This test is based loosely on a simple strategy game called *SnakeFall*. In the game, the player moves the *snake* around the board in an effort to reach the *target*:



You can play the original game for yourself using a web-browser from the following location (suggest *"Crashing Down"* as an illustrative example):

`https://github.com/thejoshwolfe/snakefall/wiki`

In this test, we are using a *simplified* version of the game which has the following characteristics:

(Snake) There is only **one** snake in the game which can have at most five sections.

(Tiles) The game board is split into a 2D array of *tiles*. Each tile is either: *empty space*, *part of the snake*, *solid ground*, *a "power up" pill*, *part of a "girder"* or *the target*.

(Lengthening) When the snake eats a *"power up" pill*, it increases in length by one section.

(Gravity) The snake is subject to *gravity*. This means that, if no part of the snake is supported then it falls downward until it either becomes supported or falls of the end of the board.

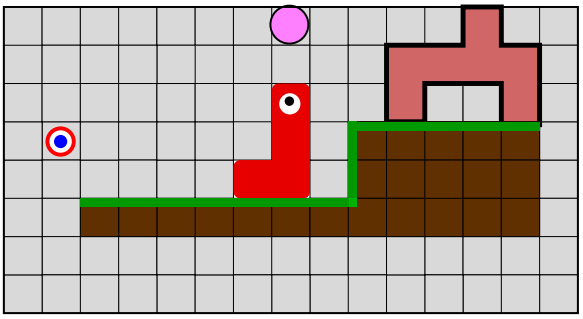
(Girders) These are solid objects in the game which can be moved around by the snake. They are subject to gravity and, if unsupported, fall downwards like the snake (more on this later).

Download. You can download the code provided for the SnakeFall program here:

`http://ecs.victoria.ac.nz/~djp/files/tt_snakefall_2020.jar`

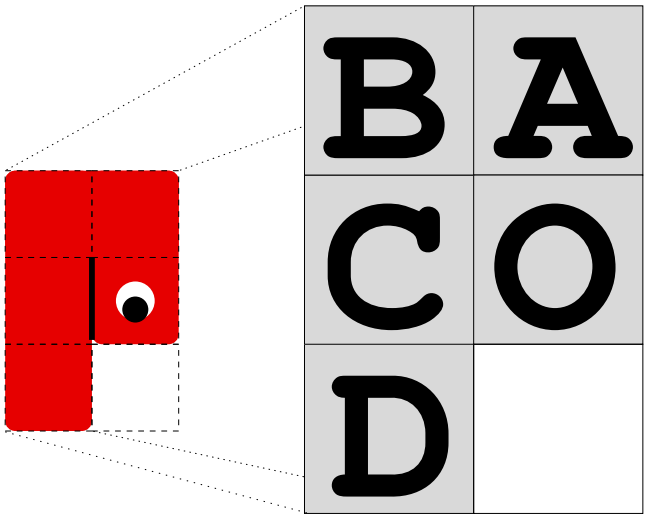
You will find several Java source files, including a JUnit test file.

ASCII Representation. In the JUnit tests, the board is represented using an array of ASCII characters. The following illustrates such a board and its ASCII representation:

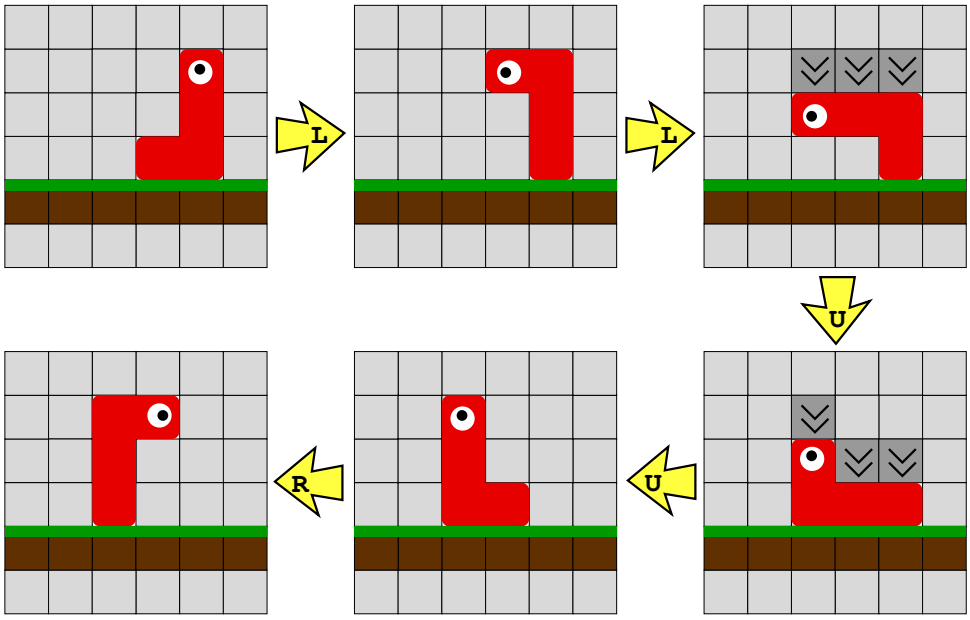


```
7|       *    N  |
6|           NNNN |
5|       O  N  N |
4| @     A #####  |
3|      CB #####  |
2|  ############  |
1|                |
0|                |
  012345678901234
```

Here, the *target* is represented using "`@`", the *power up* using "`*`", the *solid ground* using "`#`" and the girder using "`N`". The snake is represented with "`O`" for *Section 0* (i.e. the head), the letter "`A`" for *Section 1*, the letter "`B`" for *Section 2* and so on. The reason for using letters to identify sections of the snake is to ensure the layout is clear. For example:



An *input sequence* consists of a sequence of letters "`U`" (up), "`D`" (down), "`L`" (left), "`R`" (right) optionally followed by "`!`" (game won) or "`?`" (game lost). An example input sequence is "`LLUUR`". The following illustrates this sequence being executed on a given board:



Observe how, in some cases, the snake falls downward due to gravity as it becomes unable to support itself. The shaded areas indicate where the snake was positioned briefly before gravity applied.

1 Basic Moves (worth 5%)

Begin by importing the code provided into Eclipse and running the JUnit tests. A number of tests are failing because of two problems in the method `SnakeMove.apply()`:

- The snake appears to always move `Up`, regardless of what is specified in the input sequence.
- Snakes with more than two sections leave parts of their body behind when moving.

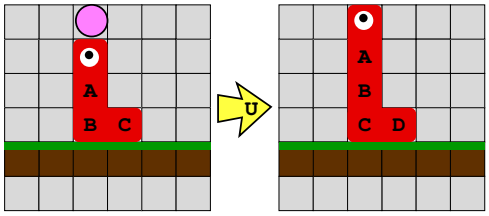
Having fixed these bugs, you should find tests `test_01`...`test_10` now pass.

2 Obstructions (worth 10%)

A large number of tests are failing due to problems related to the detection of *obstructions* during movement of the snake. Specifically, an obstruction is something the snake cannot move through (e.g. solid ground or the snake itself). Currently, the method `SnakeMove.apply()` does not consider obstructions at all. The method `Tile.isObstruction()` is provided to aid in identifying which tiles are obstructions. Having fixed these bugs, you should find tests `test_11`...`test_20` now pass.

3 Lengthening (worth 15%)

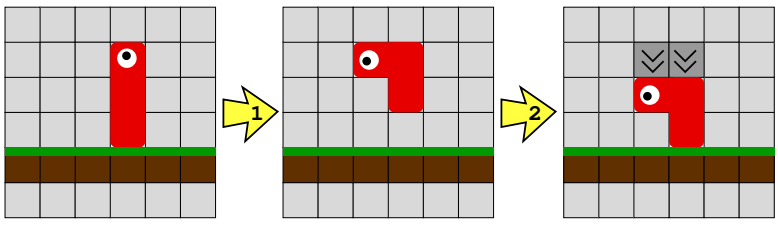
Some or all of the tests `test_21`,...,`test_26` currently fail due to problems related to how the snake lengthens when eating a "power up" pill. The snake lengthens by moving into the tile containing the "power up", and then a new section is appended to the end of the snake. The following illustrates:



Here, we see that the head along with sections `A-C` have all moved in the usual manner and, at the end, a new section `D` has been appended in the location previously held by section `C`. Having fixed these bugs, you should find tests `test_21`...`test_26` now pass.

4 Gravity (worth 30%)

Some or all of the tests `test_27`,...,`test_40` currently fail because the game does not implement gravity. If, after the snake has moved, it is *unsupported* by solid ground or a "power up" pill then gravity applies (ignoring girders for now). In such case, all sections of the snake move downwards until at least one section is supported. The following illustrates a *single move left* in two steps:



Here, we see the snake initially moves as normal but, at this point, finds itself floating in mid-air. Then, in the second step, gravity is applied and the snakes falls down until it is supported again. At this point, the sequence for a single move left is complete. Having fixed the issues related to gravity, you should find tests `test_27`...`test_40` now pass.

HINT: The method `Game.applyGravity()` provides some hints as to the process of applying gravity. For now, you can also ignore the possibility of the snake falling off the board.

5 Game Over (worth 15%)

You should find that some or all of the tests `test_41`,...,`test_51` currently fail. This is because the concept of *game over* has not been implemented. Roughly speaking, the game ends in one of two possibilities:

1. **(Game Won)**. The snake eats the target. At this point the game stops immediately and the player wins.
2. **(Game Lost)**. The snake falls off the end of the board. As soon as any part of the snake falls off the board, the game stops immediately and the player has lost.

An input sequence ending in either "`!`" or "`?`" indicates that, after the last move, the game is either won ("`!`") or lost ("`?`"). Having implemented this feature, you should find tests `test_41`...`test_51` now pass.

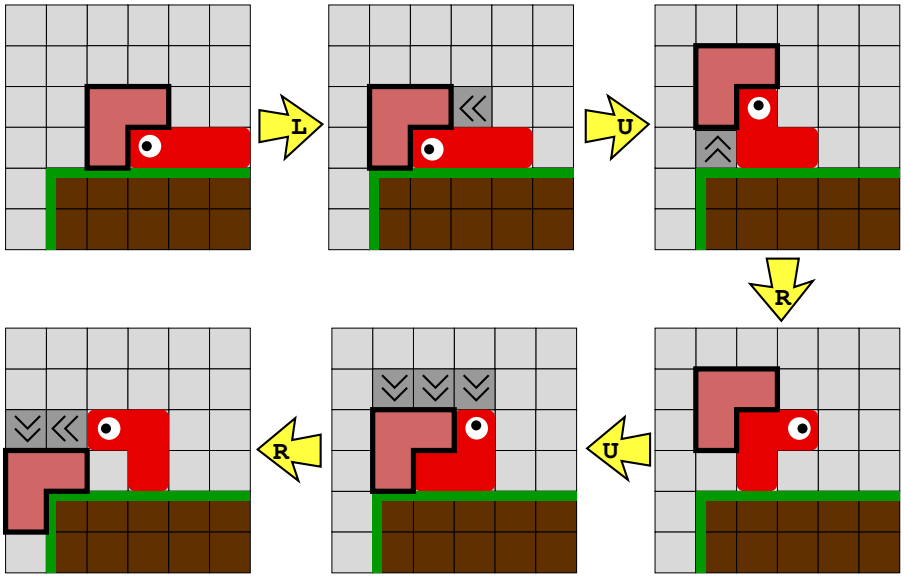
HINT: If the input sequence uses either "`!`" or "`?`" incorrectly, then a `GameError` exception should be thrown.

6 Girders (worth 25%)

You should find that some or all of the tests `test_52`,...,`test_64` currently fail. This is because code for implementing girders has not been implemented at all. Some notes about girders:

- **(Movement)** Girders are fixed objects such that, when moved in a given direction, all sections of the girder move together. The snake can move a girder by pushing it in a given direction.
- **(Obstructions)** If any section of a girder is obstructed from moving in a particular direction, then the girder as a whole cannot move in that direction.
- **(Gravity)** Girders are subject to gravity. Thus, if no section of the girder is supported, then it moves downwards as for the snake. Girders can be supported by *solid ground*, *the snake*, *a "power up" pill* or *other girders* (which are themselves supported).

The following illustrates a simple sequence involving girders.



For **simplicity**, *you can also assume there is only ever one girder on the board at any given time.* Likewise, you can assume *the target supports a girder.* Having implemented this feature, you should find tests `test_52`...`test_64` now pass.

Submission.

Your test solution should be submitted electronically via the *online submission system*:

`https://apps.ecs.vuw.ac.nz/submit/SWEN221/Terms_Test_1`

Late submissions will get zero marks (unless you have arranged this with us, which will only be in exceptional circumstances). The minimum set of required files is:

```
snakefall/io/Parser.java
snakefall/io/GameError.java
snakefall/Game.java
snakefall/events/Event.java
snakefall/events/GameOver.java
snakefall/events/SnakeMove.java
snakefall/util/Position.java
snakefall/tiles/GroundTile.java
snakefall/tiles/PowerUpTile.java
snakefall/tiles/TargetTile.java
snakefall/tiles/SnakeTile.java
snakefall/tiles/Tile.java
```