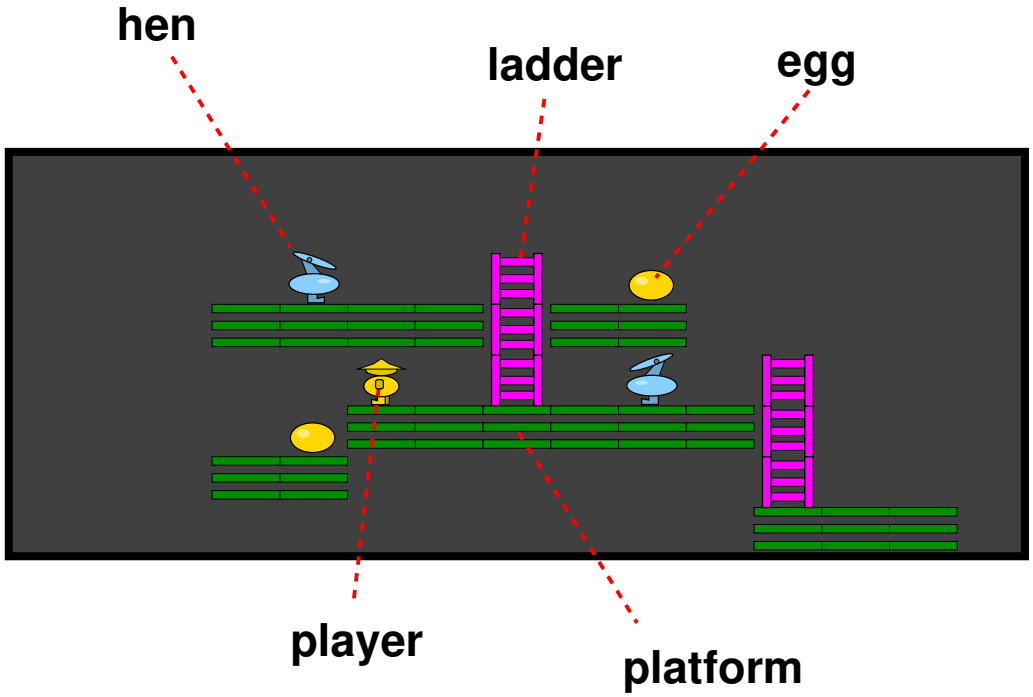
Victoria University of Wellington, School of Engineering and Computer Science

SWEN221: Software Development

Test 1 (worth 20% of overall mark)

Chuckie Egg

This test is about a computer game called *Chuckie Egg* that was originally released for the ZX Spectrum and BBC Micro in 1983. You may read about this game here `https://en.wikipedia.org/wiki/Chuckie_Egg` and play it online here `http://bbcmicro.co.uk`. In the game, the player explores a *hen house* collecting *eggs* by navigating *platforms* and *ladders*, whilst avoiding the *hens* on patrol. The game is won if the player manages to collect all the eggs; likewise, the game is lost if the player is captured by a hen or falls through the bottom of the hen house. The aim of this test is to complete a program that checks a sequence of moves in the game of Chuckie Egg is valid:



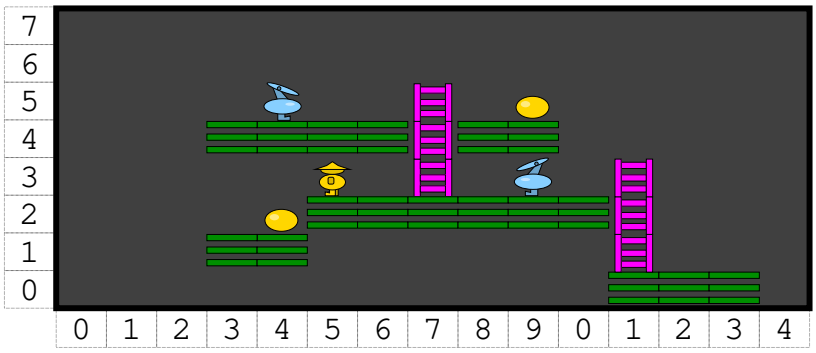
In this test, we are using a *simplified* version of the game. In particular, there are no *seeds* or *elevators* and the player cannot *jump*.

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

`http://ecs.victoria.ac.nz/~djp/files/tt_egg_2021.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|                |
6|                |
5|    p  # *      |
4|   ====#==      |
3|     O # q #    |
2|    *=======#   |
1|   ==       #   |
0|            === |
  012345678901234
```

Here, the *player* is represented using "`O`", an *egg* using "`*`", the *platforms* using "`=`", the *ladders* using "`#`" and the *hens* using "`p`" (facing right) and "`q`" (facing left).

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`".

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 a problem in the method `PlayerMove.apply()`:

- The player appears to always move right, regardless of what is specified in the input sequence.

Having fixed this bug, 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 player. Specifically, an obstruction is something the player cannot move through (e.g. a platform). If the player attempts to move into an obstruction, they should be prevented (i.e. the player remains in the same place). Currently, the method `PlayerMove.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 Collecting Eggs (worth 15%)

You should find that some or all of the tests `test_21`,...,`test_30` currently fail. This is because the method `GameOver.apply()` has not been properly implemented. This method is called when either "`!`" or "`?`" occurs at the end of the input sequence and indicates that, after the last move, the game is either won ("`!`") or lost ("`?`").

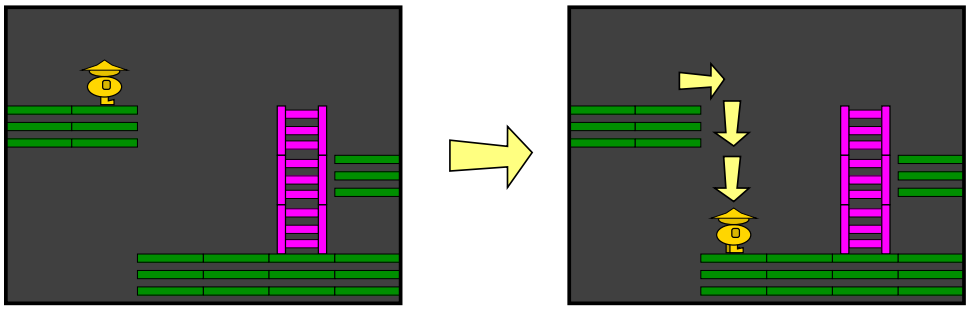
There are several problems to fix:

1. If the game is indicated as won, but the player has not yet collected all the eggs then a `GameError` exception should be thrown.
2. Likewise, if the game is indicated as lost but the player remains on the board, then the game is not actually finished and a `GameError` exception should be thrown.
3. Finally, the player cannot carry on moving after the game is finished.

Having fixed these issues, you should find tests `test_21`...`test_30` now pass.

4 Gravity (worth 30%)

Some or all of the tests `test_31`,...,`test_40` currently fail because the game does not implement *gravity*. If, after the player has moved, they are *unsupported* by a platform then gravity applies (ignoring ladders for now). In such case, the player is moved downwards until they are supported (i.e. lands on a platform) or the game is over (i.e. because they fell off the board or collected the last egg). The following illustrates a *single move right* where gravity is applied:

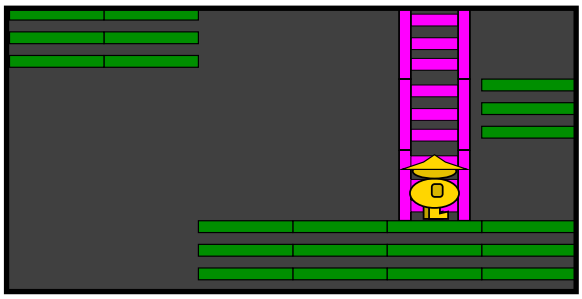


Here, we see the player initially moves right and then gravity automatically drops the player down to the platform below. An empty method `Game.applyGravity()` is provided for implemented this feature. Having fixed the issues related to gravity, you should find tests `test_31`...`test_40` now pass.

5 Ladders (worth 20%)

Some or all of the tests `test_41`,...,`test_52` currently fail because the game does not yet support *ladders*. To fix this, there are several challenges:

1. Creating a `Ladder` class, and ensuring it is instantiated in `Game.createPieceFromChar()`.
2. Updating `Player` to record whether the player is on a ladder (or not).
3. Updating `PlayerMove.apply()` to determine when the player is on a ladder. Note the ASCII representation of the player on a ladder is "`@`". The following illustrates:



```
3|==  # |
2|    #=|
1|    @ |
0|  ====|
  012345
```

4. Updating `PlayerMove.apply()` so that, *when the player is on a ladder*, they can move *up* and *down* it.

Having correctly implemented the above, you should now be passing at least tests `test_41`,...,`test_43`. However, **more work is required** to pass the remainder for this section (`test_44`,...,`test_52`).

6 Hens (worth 20%)

Some or all of the tests `test_53`,...,`test_70` fail because there is currently no support for *hens*. Some notes about hens:

- **(Movement)**. Hens move either left or right. A hen moving in a given direction continues until it reaches the end of the platform (or is obstructed). Notice from page 2 that the ASCII representation of a hen depends on the direction it is moving.
- **(Turning)**. Once a hen has reached the end of a platform (or is obstructed), the hen will turn around and then begin moving in the opposite direction. This process requires one time step. For simplicity, we assume hens are obstructed by *platforms*, *ladders* and *eggs*.
- **(Game Over)**. The game is over when a hen catches the player. Note, the player cannot "move through" a hen under any circumstance.

For **simplicity**, you can assume *hens never meet each other*. Having implemented this feature, you should find tests `test_53`...`test_70` 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:

```
chuckie/Game.java
chuckie/events/Event.java
chuckie/events/GameOver.java
chuckie/events/PlayerMove.java
chuckie/io/GameError.java
chuckie/io/Parser.java
chuckie/tiles/Egg.java
chuckie/tiles/Platform.java
chuckie/tiles/Player.java
chuckie/tiles/Tile.java
chuckie/util/Position.java
```