Victoria University of Wellington, School of Engineering and Computer Science

SWEN221: Software Development

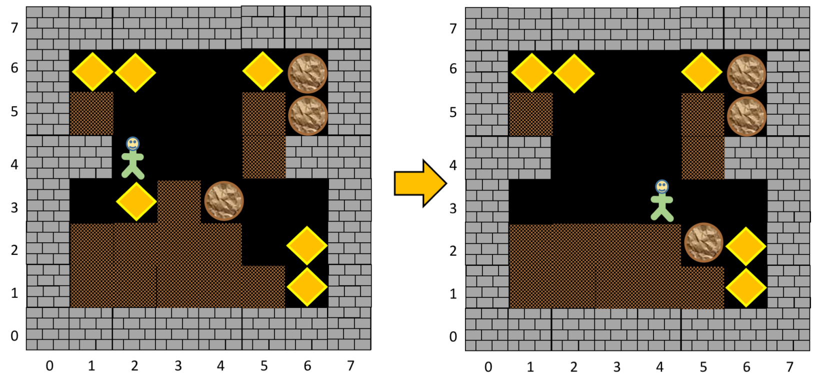
Mid-term Test (worth 10% of overall mark)

Boulder Dash

This test is about a computer game called *Boulder Dash* that was originally released for the Atari in 1988. You may read about this game here `https://en.wikipedia.org/wiki/Boulder_Dash` and play it online here `http://bbcmicro.co.uk`. In the game, the player digs through an underground cavern in search of *diamonds* whilst avoiding *falling rocks*. A rock will fall downwards if it is not supported by something (e.g. earth, the player or another rock). A rock will fall to the side if it is on top of another rock and there is nothing to prevent this (i.e. nothing to its *left* and *lower left* or *right* and *lower right*). The player may push rocks to the left or right provided there is nothing where the rock will go. The player cannot move through *rocks* or *walls* and cannot push more than one *rock* at a time. The aim of this test is to complete a program that checks a sequence of moves in the game of Boulder Dash is valid.

The program reads in a file which represents a *game of Boulder Dash*. An example game file is given below on the left, along with a graphical visualisation of the starting and final boards on the right:

```
4 8,8
M(S;1)
D(E;1)
P(E;1)
```



The first line of the file gives the number of moves, followed by the *width* and *height* of the board. The remaining lines give the moves of the game. Only four kinds of move are supported:

- `M(N/S/E/W;1-9)` — The player moves a number of steps from his/her current position in the given *direction*, which is: `N` (North), `S` (South), `E` (East) or `W` (West). Finally, the *number of steps* is a number between 1 and 9 (inclusive). Diamonds are automatically taken during this.

- `D(N/S/E/W;1-9)` — The player digs a number of steps from his/her current position in the given *direction*. The *direction* and *number of steps* are the same as above.

- `P(E/W;1-9)` — The player pushes a rock for a number of steps from his/her current position in the given *direction*, either `E` (East) or `W` (West). The *number of steps* are the same as above.

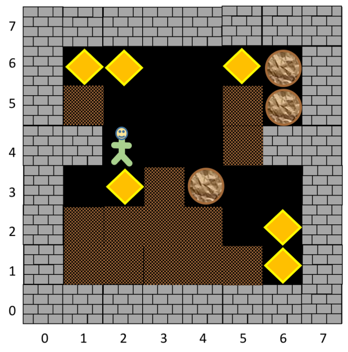
- `O` — Signals that the game is *over*. This occurs only when a rock falls on the player.

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

`http://ecs.victoria.ac.nz/~djp/files/tt18_boulder.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:



```
7|#|#|#|#|#|#|#|#|
6|#|+|+|_|_|+|O|#|
5|#|:|_|_|_|:|O|#|
4|#|#|X|_|_|:|#|#|
3|#|_|+|:|O|_|_|#|
2|#|:|:|:|:|_|+|#|
1|#|:|:|:|:|:|+|#|
0|#|#|#|#|#|#|#|#|
  0 1 2 3 4 5 6 7
```

Here, "`X`" indicates the player, "`O`" indicates a rock, "`+`" indicates a diamond, "`#`" indicates a wall and "`:`" indicates earth.

1 Getting Started (worth 5%)

Begin by importing the code provided into Eclipse and running the JUnit tests. Several tests should be failing, and helpful information regarding these errors is printed into the Eclipse console. There is one bug in `Parser.java`, and your aim is to find and correct this mistake. Upon completing this, you should find that tests `test_01` … `test_02` now pass. **NOTE:** you should also find that many or all of the other tests now fail!

2 Invalid Walking Moves (worth 10%)

You should find that some or all of the tests `test_03`, ..., `test_09` currently fail. This is because the method `WalkingMove.apply()` does not check for the following invalid moves:

1. *Trying to move into a position that is already occupied by a wall, rock or some earth.*
2. *Trying to move through a position that is already occupied by a wall, rock or some earth*

Having implemented these rules, you should find that tests `test_09`, ..., `test_09` now pass.

3 Valid Pushing Moves (worth 10%)

You should find that some or all of the tests `test_10`, ..., `test_13` currently fail. This is because the method `PushingMove.apply()` does not update the position of the rock, only the position of the player. Upon fixing this, you should find that tests `test_10`, …, `test_13` now pass.

4 Invalid Digging Moves (worth 15%)

You should find that some or all of the tests `test_14` … `test_18` currently fail. This is because `DiggingMove.apply()` incorrectly uses `AbstractMove.isMoveValid()` which is not suitable for `DiggingMove`. Instead, you should create a method `DiggingMove.isDiggingValid()` which checks for the following invalid move:

- *Trying to dig into a position that is* **not** *already occupied by earth.*

Having fixed these two problems, you should find that `test_14` … `test_18` now pass.

5 Valid Digging and Diamond Acquisition (worth 20%)

You should find that some or all of the tests `test_19` ... `test_21` currently fail. This is because both `DiggingMove` and `WalkingMove` use `AbstractMove.movePiece()`. This method moves a piece from one position to another, *but currently does not clear the path in between.* You should implement a method `clearPath(Position,Game)` for doing this and call it from `movePiece()`.

Having fixed these two problems, you should find that `test_19` ... `test_21` now pass.

6 Invalid Pushing Moves (worth 10%)

You should find that some or all of the tests `test_22`, ..., `test_33` currently fail. This is because the method `PushingMove.apply()` does not check for the following invalid moves:

1. *Player trying to push rock up or down*
2. *Player trying to push something other than a rock*
3. *Player trying to push rock into wall or another rock*
4. *Player trying to push more than one rock at a time*

Having implemented these rules, you should find that tests `test_22`, ..., `test_33` now pass.

7 Falling Moves (worth 20%)

You should find that some or all of the tests `test_34`, ..., `test_45` currently fail. This is because the implementation for falling rocks is incomplete. In particular, it should support the following:

1. *A rock with nothing underneath continues to fall until it lands on something*
2. *A rock on a rock with nothing on its left and lower left will fall to the left*
3. *A rock on a rock with nothing on its right and lower right will fall to the right*

Having done this, you should find that tests `test_34`, ..., `test_45` now pass.

HINT: One way to implement these rules is to add a method `AbstractMove.moveRocks()` which is called as appropriate from `WalkingMove`, `DiggingMove` and `PushingMove`. The method `moveRocks()` itself calls another *recursive* method `AbstractMove.moveRock()`

8 Game Over (worth 10%)

The implementation does not currently support the concept of "game over". In particular, it should support the following:

1. *If a rock falls downwards and lands on the player, the game is over and the only valid move after this* `GameOverMove`*.*

To implement these rules, you need to adjust your mechanism for falling rocks to know when a rock is falling versus when it is at rest. Having completed this, you should find that tests `test_46`, ..., `test_50` now pass.

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

```
http://ecs.victoria.ac.nz/cgi-bin/auth/submit?course=SWEN221
```

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:

```
boulderdash/Game.java
boulderdash/io/GameError.java
boulderdash/io/Parser.java
boulderdash/moves/AbstractMove.java
boulderdash/moves/DiggingMove.java
boulderdash/moves/GameOverMove.java
boulderdash/moves/Move.java
boulderdash/moves/PushingMove.java
boulderdash/moves/WalkingMove.java
boulderdash/pieces/Diamond.java
boulderdash/pieces/Earth.java
boulderdash/pieces/Piece.java
boulderdash/pieces/Player.java
boulderdash/pieces/Rock.java
boulderdash/pieces/Wall.java
boulderdash/util/Position.java
```