Victoria University of Wellington
School of Engineering and Computer Science

SWEN222: Software Design, Assignment 1

Due: Sunday 13th August @ Mid-night

1 Introduction

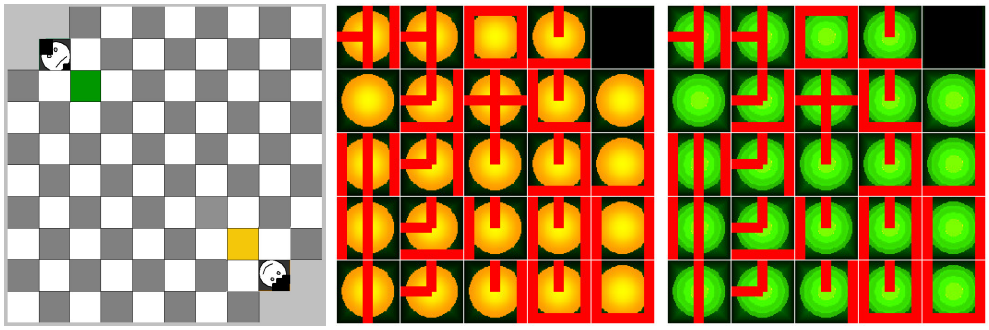
Sword and shield is a two player game of pure skill, with no element of luck. It has no hidden information, like chess. You are to implement a text interface for Sword and shield in Java, allowing people to play the game while sitting at the same computer. While implementing your program,

you should focus on using proper object oriented design and, when possible, libraries. Both standard libraries like "Collections" and third party libraries may be used.

2 Rules of the game

2.1 Pieces

Pieces: a 10*10 board as in the image, A set of 24 square pieces and a set of 24 green pieces as in the image. Every side of each piece have one of those 3 symbols: Sword, Shield or nothing. Notice how each piece is different from all the other ones, and all the possible permutations of the three symbols are allowed.



Shields are vertical lines around the outside of the piece, and swords are lines that go through the piece. For example the piece in the top left has three swords and a shield whereas the piece to the right does not have the shield. The third piece in that row has four shields and no sword.

The direction a piece is facing is important, as it determines what reactions may occur.

2.2 Reactions

Independently of their colour, whenever two pieces meet on the board, if they point a sword to one another, a *reaction* happens. It is not relevant if the pieces are of the same colour or of different colours. The two Faces are also considered pieces for reactions. Faces are immobile and drawn on the board in position 2*2 and 9*9.

- Sword against nothing: the piece with nothing is eliminated.
- Sword against sword: both pieces are eliminated.

- Sword against shield: the piece with sword is pushed back one cell. If a piece go out of the board is eliminated. In this game, the corner positions 1*1,1*2,2*1,2*2, 9*10, 9*10, 10*10 are considered out of the board.
- Sword against Face: Victory condition; the Face attacked by the sword loses the game.

Eliminated pieces are removed from the board, can not be created again and need to be placed in the *cemetery*.

If multiple reactions are possible (that is, there is more then one sword pointing to something) the player holding the turn decides which reaction takes place, that is, they schedule what happens first. You can see an example of multiple reactions on the supporting slides on the course web page.

2.3 Turns

As in chess, the game proceeds in turns starting with the yellow player.

Every round:

- At the start of your turn you may create a piece if your creation grid is empty. Your creation grid is at position (3,3) for yellow and (8,8) for green. Select a piece that is not on the board or in the cemetery, and place it onto the creation square in the orientation you prefer.
- Next, you may choose to move or rotate any number of your pieces, choosing the order. More in details:

  (a) If you want, you can pass the turn

  (b) Chose a piece that you have not already moved or rotated,

  (c) Rotate it (to any new orientation) or move it (a single cell up, down, left, or right). You can freely move your pieces in one of the adjacent positions: when a piece moves (or is pushed by another piece) all the pieces in its way are pushed away n the same direction. No two pieces may be in the same square. Remember that pieces pushed off the board are sent to the cemetery.

  (d) Apply all the reactions. There could be a very long chain of reactions. Moves that requires to perform and infinite amount of reactions are invalid, just choose another move.

  (e) go back to (a), remember you will need to chose a different piece every time.

3 Implementation

You need to implement a textual interfaced for the game. To represent the various pieces use a letter (a,b,c,d..) surrounded by symbols for its swords and shields, for example, here are two pieces next to each other:

```
  |  |
 -e# D
  #  #
```

'e' has two swords (to it's top and left) and two shields (to it's bottom and right). This corresponds to piece (2,2) in the pieces image. 'd' has one sword (to it's top) and one shield (to it's bottom). This corresponds to piece (4,1) in the pieces image.

The face for yellow is represented by a 0, and the face for green is represented by a 1.

You need to display the state of the game by showing the pieces on the board, the ones in the cemetery and the ones ready to be created and the player having the turn (green or yellow).

In any round, the user will start typing "`create <letter> <0/90/180/270>`" to create a certain piece in one of the 4 possible orientations; otherwise the user may write "`pass`" to chose to not create.

Then the user will write "`rotate <letter> <1-4>`" to rotate a piece
or "`move <letter> <up/right/down/left>`" to move a piece in one direction.

When there is a reaction, the user will be asked to write the letter of the two pieces that are going to interact. Note: The game must provide the set of valid choices, the user can select its preferred one and finally the game must check that the select one is a valid reaction.
The use may then give another move/rotate command, or write "`pass`" to end its turn. The user may then give another move/rotate command, or write "`pass`" to end their turn. Note that "`pass`" can not be used when a reaction needs to happen.

In any moment a player can type "`undo`" to revert the last command.

All input/output for this should occur via `System.in` and `System.out`. **DO NOT USE A GRAPHICAL USER INTERFACE MARKS WILL BE DEDUCTED FOR DOING THIS!**

4 Submission

Your program code should be submitted electronically via the *online submission system*, linked from the course homepage. Your submitted code should be packaged into a jar file, including the source code (see the export-to-jar tutorial linked from the course homepage). Your program should include appropriate Javadoc comments, and provide a suite of JUnit test cases. Be sure your submission includes also all the relevant third party library code. Include also a *readme.txt* with the commands to compile and run your code from the command line; that is outside of Eclipse.

5 Partial completion

This assignment requires to implement non trivial logic in the behaviour of the game. We suggest you to proceed in the following steps:

1 Ignore rotations and reactions. At this stage of development, players will be able to create pieces, and to move them on the board. Pieces can be eliminated if they chose to move outside of the board or if they are pushed outside of the board by another piece. Even without reaction, a piece moving will push other pieces that are on the way, to enforce "only one piece for cell". In this way, a piece can be pushed out of the board.

   Since winning the game requires a reaction against the face, the game can not have a winner at this stage. Be sure the undo functionality works well in this simple setting.

2 Add rotations. Check undo still works properly.

3 Add reactions. If a move would cause an infinite amount of reactions, the user can simply undo the move. The game need not to check that the chain of reaction is terminating. Now the game can end, thus you also need to handle the concept of victory.

If for any reason you end up unable to complete all the functionalities, declare in your Design report what stage(1,2,3) you reached. For stage 1 you mark is going to have a cap at 70%, for stage 2 at 80% and for stage 3 100%, of course.

6 Required elements in your submission

In addition to the source code, various pieces of design documentation and testing are required. Those should be submitted as Pdf files; other formats will not be accepted. The required documents are:

1. **Class Diagram** illustrating the main aspects of the class hierarchy. This should not be cluttered with unnecessary information such as, for example, `toString` methods.
2. **Design report** A 200 word report written in your own words giving an overview of the design for the game. While you can mention the (textual) user interface, keep the focus on the code implementing the logic of the game.
3. **Good code highlight** A 200 word description commenting a 2-50 lines fragment of source-code you wrote (as part of your final submission for this assignment) that you believe to be brilliant, well designed or otherwise worthy of praise.
4. **Testing discussion** A 200 word report describing how you approached testing your code: how you designed the various tests and how those provide confidence in the correctness of your work.

7 Assessment Marks [100 marks]

- **Class Diagram (10 marks)** Marks will be awarded for correct use of the following: *inheritance*, *multiplicity*, *associations*, *classes* and *attributes*. Marks will also be awarded for *clarity*.
- **Design report and code quality (45 marks)** This report will serve as a guide to understand the structure of your code. Marks will be awarded for how well it conveys aspects of the program's design. Marks will also be awarded for how well-written and easy to follow it is. While examining your design, your code will be explored and executed. Marks will be awarded for
  - Overall code style which includes: good use of naming for methods, fields, classes and variables, good division of work into methods, so as to avoid long and complex chunks of code.
  - Good use of JavaDoc comments on all public classes, interfaces and methods. Marks will also be awarded for good use of internal (i.e. non-javadoc) comments. Typically, these are found within a method body describing some aspect of how it works. Comments should make sense and correctly describe what is happening.
  - Code correctness: the game can be correctly played, the rules of the game are enforced and the program does not crash.
- **Smart code highlight (15 marks)** Marks will be awarded for how good and well designed the proposed code is. The code will be judged in the context of the program, and if that is considered to be the right code to solve the job at hand.
- **Testing discussion (30 marks)** This report will serve as a guide to understand the structure of your tests, Marks will be awarded depending on how well your tests cover your code; how well you provide confidence in the correctness of your work both in your design intent (as emerge from your report) and your design implementation (as emerge from tests in your code). This is your opportunity to convince us of your code correctness.

  As a rule of thumb, we are looking at something between 20 to 50 tests, depending on how you design them.