Lancaster University

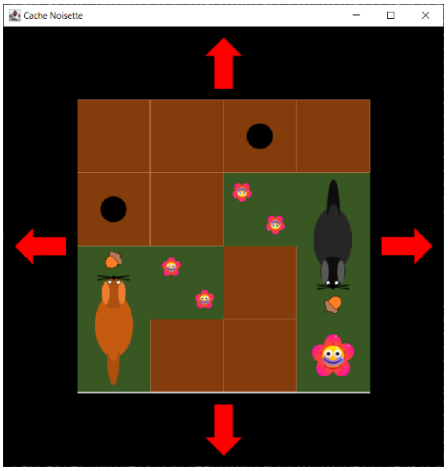
Assessed Exercise: **Cache Noisette!**
Moodle Submission Deadline: **16:00 Friday Week 20**
Assessment Mode: **Submission of completed coursework to moodle**

Aims

This assessed exercise is designed to test your understanding of all the software development concepts we've seen in the lectures and labs, and their application using Java and the Swing APIs. The assignment is separated into incremental tasks, with marks being awarded for the successful completion of each task. Your assignment is to create a Java Swing based interactive puzzle game called **Cache Noisettes** using Java and Swing.

I bought this game on holiday in France a few years ago – it roughly translates as "Nut Store". It's a single player puzzle game that involves placing a number of squirrel pieces on a 4x4 grid, and moving them (following specific rules) such that they each drop a nut that they are carrying into a hole in the gameboard. Sound easy? Not so much… the squirrel pieces have a nasty habit of covering the holes…

A photo of the Cache Noisettes game is shown below, alongside an example of what your Java version might ultimately look like (although you are free to design your solution as you wish):



Game Rules

The rules of the game are very simple.

- The game board consists of a 4x4 grid of empty spaces.
- There are holes in four of those spaces. These are always in the same place, as illustrated:

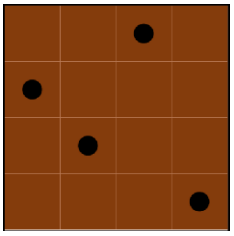


Figure 1: An Empty Cache Noisettes Board

- There are game pieces that look like squirrels. Up to four squirrel pieces may be placed on the board, each has its own colour and shape. At the start of the game, each squirrel carries a nut. The nut resides on the same square as the squirrel's head.
- There are many levels to the game (there are sixty levels in the full game!). Each level defines which of the squirrel pieces are in use for that level, their starting position on the board, and the rotation of the pieces (a squirrel may face up, down, left or right on the board). Once placed at the start of the game, the rotation of the pieces cannot be changed.
- There is also a game piece that looks like a flower, that may be placed into one of the holes, which makes that hole unusable, and also means that no other game piece may occupy that space on the board. The position of the flower (if any) is also defined for each level.
- Once the game starts, the player may move any of the squirrel pieces left, right, up or down on the board, one space at a time. Squirrels may not move diagonally, and no part of a squirrel may leave the 4x4 grid. Similarly, no part of a squirrel may be moved into a space that is occupied by another piece (such as another squirrel or a flower). Any part of a squirrel may however move onto a hole.
- If the part of a squirrel holding a nut moves over a hole, the nut drops into that hole. A hole may only hold at most one nut.
- The game is won when every nut is placed into a hole.

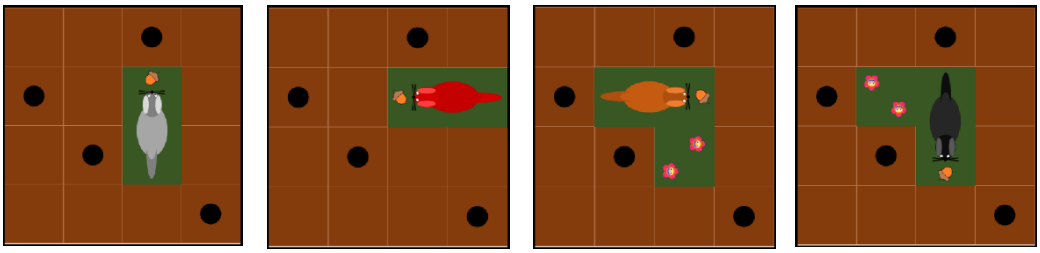


Figure 2: The squirrels in the game, facing up, left, right and down respectively.

Note the Grey and Red squirrels are straight and occupy two board spaces, but the Brown and Black squirrel pieces are 'L' shaped and occupy three spaces.

Resources

Since SCC110 is a Software Development course and not a course in computer graphics, we've provided some images for you. **Download the images.zip file** from the SCC110 web page and decompress it into the directory where you are developing your software.

This zip file gives you access to the following images. Feel free to make use of these in your program. However, if you prefer to create your own graphics for fun, this is of course just fine.

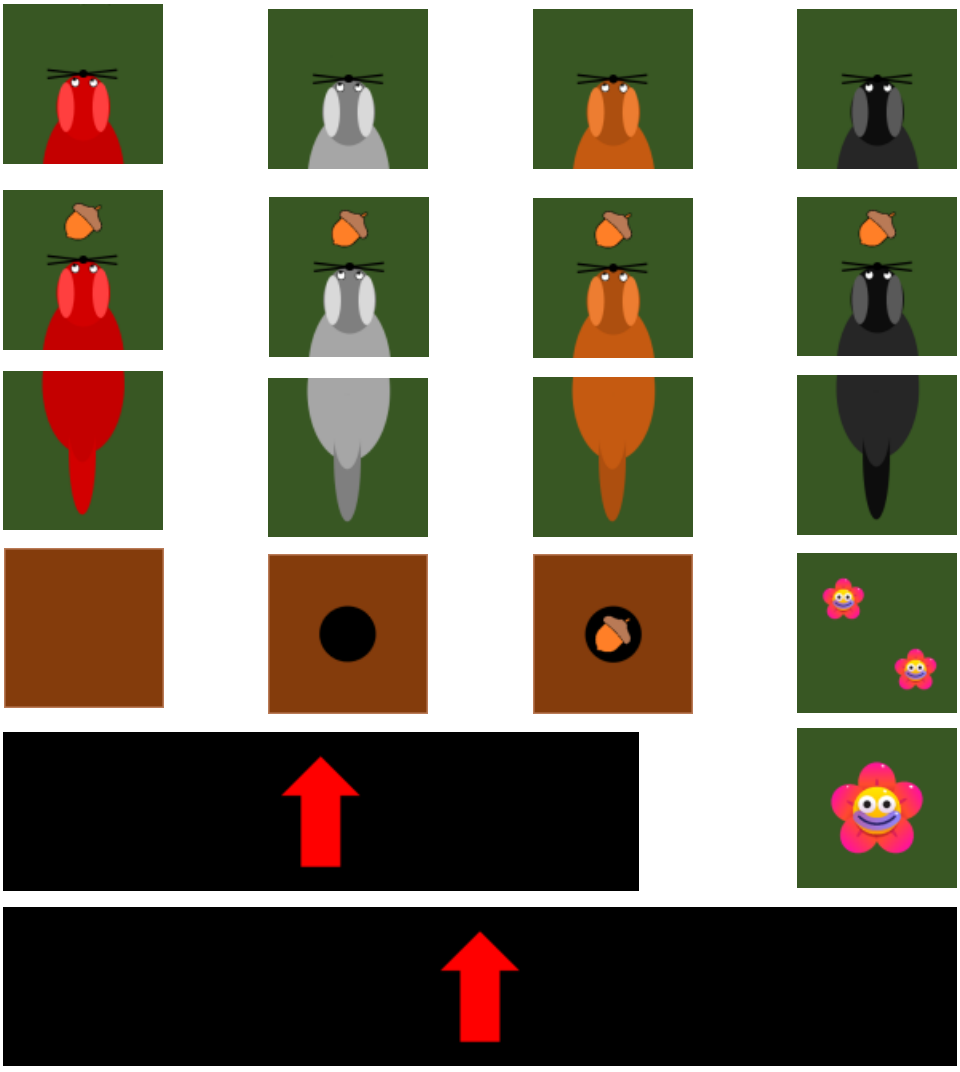
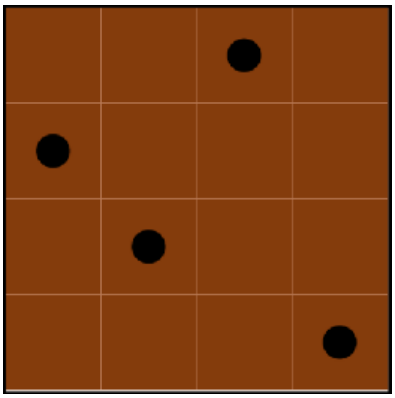


Figure 3: Images available for use

Lancaster University

Task 1: Creating the Board…

- As professional software developers, you know you should be using a version control system for your project… So start by creating a **private** github repository for your work. 😊
- Download the image resources from the SCC110 web page and extract them into the directory where you will be developing your code.
- Create a class to represent your game board, including a Graphical User Interface (GUI) using Swing. **Think carefully about which Swing classes will help you here**. Refer back to the lectures if you need a reminder on how to do this.
- Remember to write a constructor for your class that uses Swing to show a window with an empty game board.
- You are free to develop your work using your own computers, but the code you write **must work on the SCC Ubuntu virtual machine on mylab.lancs.ac.uk.** This is where you're work will be marked.
- Similarly, we recommend you use VS Code and the command line to write and compile you code. If you choose to use something else, **ensure your submitted work can be compiled and run from the command line using a simple javac command**. Failure to abide by these rules may result in a fail mark for the functionality part of your work.



HINT:

Images in Swing applications are represented by a class (of course!). We have provided a class called **Picture** for you to use in your solution. The **Picture** class contains a constructor that lets you create an instance of a **Picture** from a given filename and chosen rotation (in degrees).

The **JButton** class is able to create buttons showing images as well as text. In particular, the JButton class contains a **constructor** that allows a JButton to be created from a **Picture** as follows:

```
Picture p = new Picture("Empty.png", 0);
JButton b = new JButton(p);
```

Task 2: Adding pieces…

- Update your program so that it can create the Cache Noisettes Level 1, as shown below. Note that you need only support the red and grey squirrels at this stage.
- Pay particular attention to how you choose to represent squirrels in the game. For example, you may wish to consider creating a class to represent a **Squirrel** to help keep your code simpler and more elegant.
- When using the images provided, note that the filenames follow a pattern. Consider how to use this in your code to make it easy to reuse code you have written in representing different squirrels.

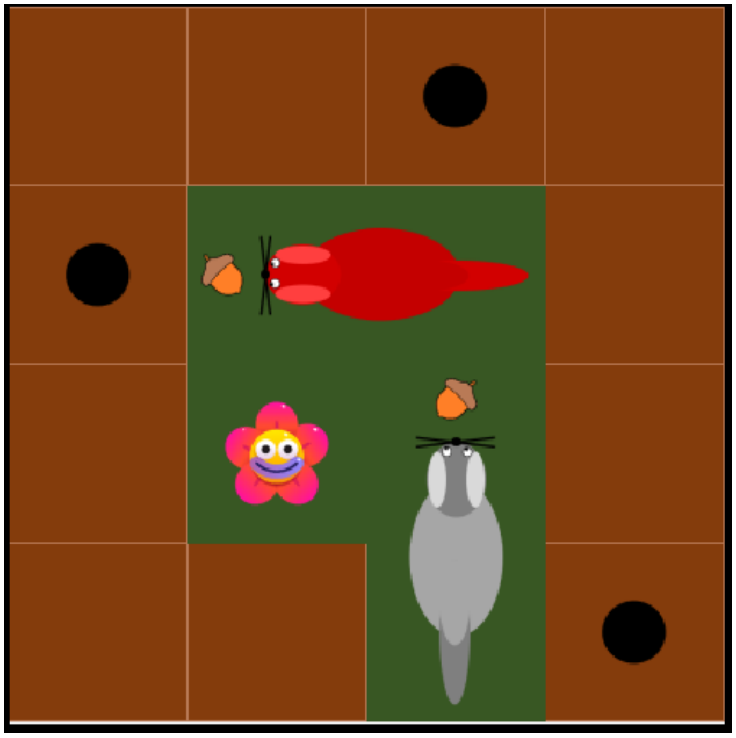


Figure 4: Cache Noisettes Level 1

HINT:

The **JButton** class is also able to change the image shown on the button, using the **setIcon**() method. For example, a **JButton** called b can change what it looks like as follows:

```
Picture p = new Picture("RedSquirrel1.png", 0);
b.setIcon(p);
```

Task 3: Moving Pieces…

- Add buttons to your user interface to provide a way to move a squirrel around the board. An example is shown below, but you may choose any user interface you wish.
- Write a method to allow the squirrels to be moved around the board.
- You may, for example, choose to select a squirrel by clicking on its head, then use the directional buttons to move that squirrel around the board.

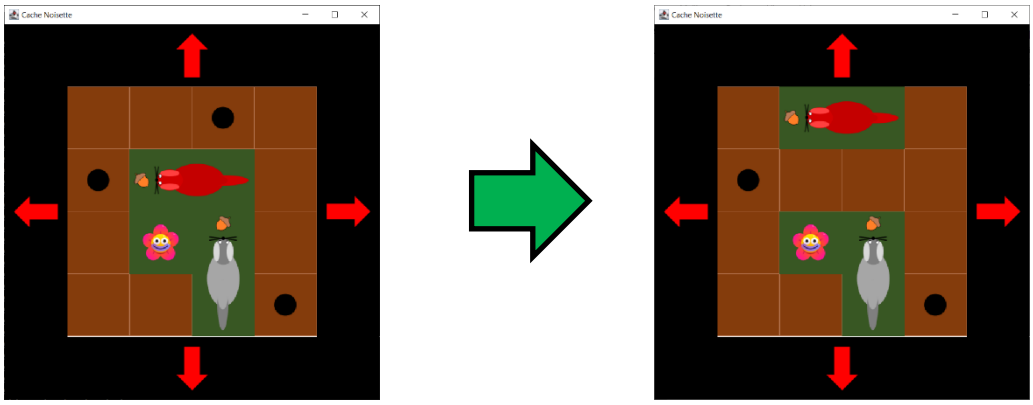


Figure 5: An example of the Red Squirrel being move up

HINT:
Moving a **JButton** is tricky… but changing what it displays is much simpler. Consider how you can make use the **setIcon**() method described earlier to make it look like a squirrel is moving…

Task 4: Ensure Legal Moves…

- Develop your software such that it will only permit legal moves, as described at the start of this document. Any attempt to perform an illegal move should be ignored.
- Remember to verify that no part of a squirrel may leave the game board, or overlap another squirrel or flower.
- Update your software so that nuts can be placed into holes appropriately.

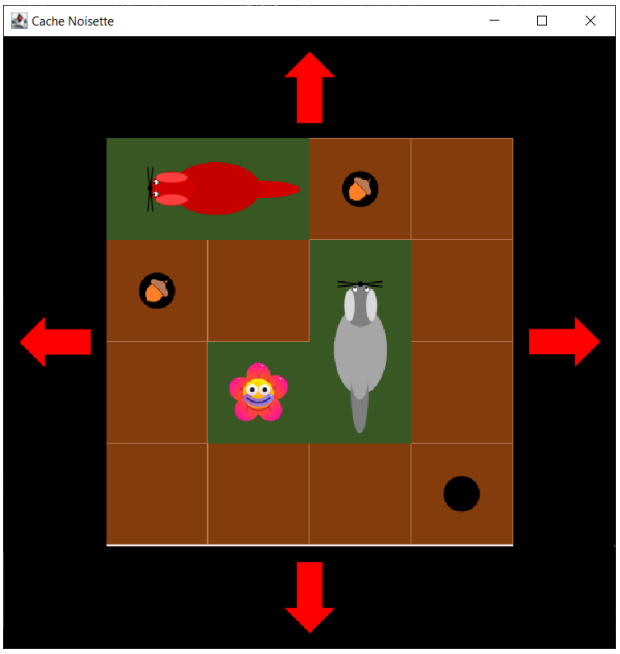


Figure 6: A completed game. Note that the nuts are displayed inside holes, and that the squirrels no longer carry them.

Task 5: Additional Squirrels and Levels (Advanced Task)

Develop your program so that it can correctly support other levels.

- Note that this will require you to support the use of any of the **four** squirrels (which may be of a different shape), and in any orientation.
- Determine when the game is won and display an appropriate message.
- Include a simple way for users to select which level they want to play. Three levels are listed below. Images of other all 60 levels in the game are available on request in your scheduled lab session should you wish to support more in your game (although this is not expected).
-

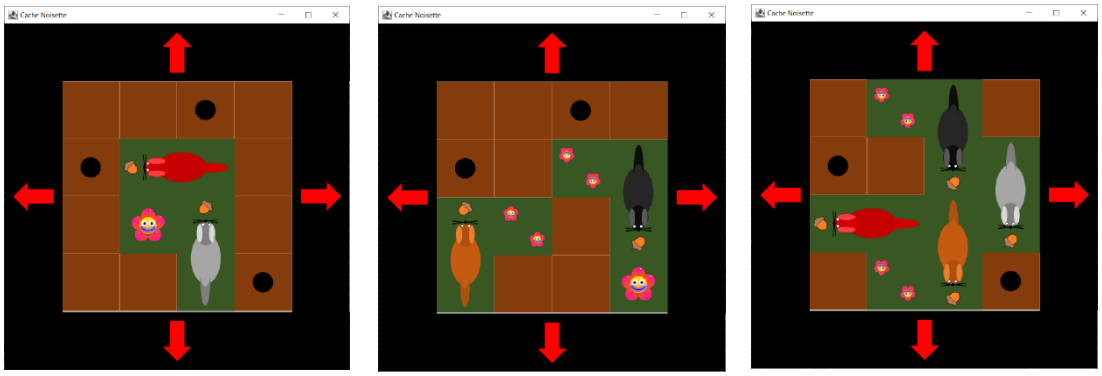


Figure 7: Cache Noisettes Levels 1, 10 and 40 (respectively)

Hacker Edition

If you have completed all the tasks and want to do more, use your imagination to add further features that stretch your abilities. Ideas may include a timer that measures how long it takes the player to complete the puzzle, improved accessibility for users who cannot use a mouse or trackpad.

Lancaster University

Assessment

This work will be assessed through:

- Offline marking of your code to moodle
- the completion of a short reflective self-assessment form, which will be provided on moodle in Week 20.

You MUST submit your code to Moodle by the advertised deadline of 16:00 Friday Week 20 (UK time). Late submissions will follow standard University policy.

Marking Scheme

Your work will be marked based on the following five categories. Your final grade will be determined based on a weighted mean of these grades according to the weighting shown in the table below.

Functionality.	60%
Use of Object Oriented principles (Classes, constructors, encapsulation, composition)	20%
Code Style and commenting	10%
Reflective Self Assessment	10%

In all cases a grade descriptor (A, B, C, D, F) will be used to mark your work in each category. The following sections describe what is expected to attain each of these grades in each category.

Markers can also recommend the award of a distinction (+) category overall if they feel a piece of work exhibits clearly demonstrable good programming practice.

NB: You may use techniques beyond those taught in the course if you understand them, but you must be able to fully explain them if we ask. Not being able to fully explain your program will likely incur a mark penalty.

Functionality:
A: Fully working program meeting all requirements of Tasks 1, 2, 3, 4 and 5.
B: Working program meeting all requirements of Tasks 1, 2, 3 and 4.
C: Working program meeting all requirements of Tasks 1, 2 and 3.
D: Working program meeting all requirements of Task 1 and 2.
F1: Working program meeting all requirements of Task 1.
F2: Program does not meet requirements of Task 1.

Code Structure and Elegance:
A: Well written, clearly structured code showing student's own examples of good OO practice.
B: Well written, clearly structured code.
C: Clearly identifiable but **occasional** weakness, such as repetitive code that could be removed through use of a loop, poor use of public/private, unnecessary / unused code, inappropriate naming and scoping of variables.
D: Clearly identifiable **systematic** weakness, such as multiple examples of repetitive code that could be removed through use of a loop, systematically poor use of public/private, large sections of unnecessary / unused code, consistently inappropriately named and scoped variables.
F: All the above.

Code Style and Commenting
A: Consistently well indented, well named and well scoped variables with Javadoc commenting.
B: occasional poor naming, scope or indentation or occasionally vague and/or inaccurate comments.
C: Systematic poor naming, scope or indentation, code is partially commented **or** systematically vague/inaccurate comments.
D: Systematic poor naming, scope or indentation, code is partially commented **and** systematically vague/inaccurate comments.
F: No attempt made

Reflection:
A: Accurate self-assessment of coursework
B: Self-assessment is inaccurate by one grade
C: Self-assessment is inaccurate by three or more grades
F: No self-assessment submitted

Star Categories:
Your marker may choose to award a star (*) grade to your work at their discretion for demonstration of consistent high-quality work throughout your coursework, or for the demonstration of significant additional work.