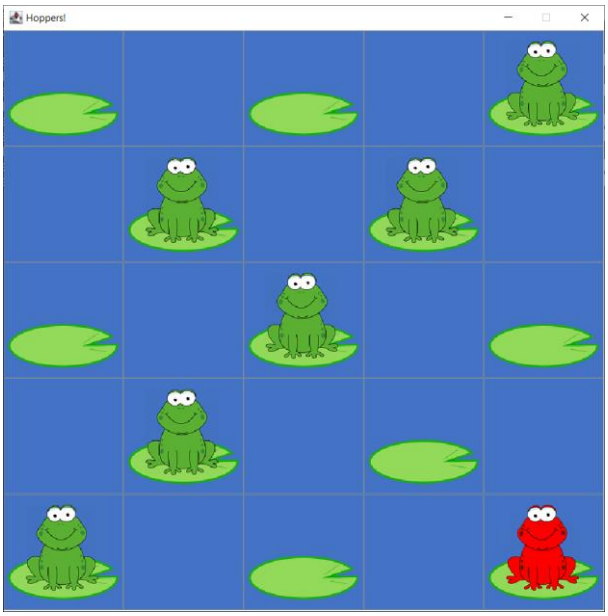
Lancaster University

Assessed Exercise: **Hoppers!**
Moodle Submission Deadline: **16:00 Friday Week 20**
Assessment Mode: **IN LAB ASSESSMENT IN YOUR OWN WEEK 20 LAB SESSION**

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 an interactive model of a simple board game called **Hoppers**, which is a variant of the classic game Solitaire (https://www.youtube.com/watch?v=E-1eK3vDV_Y). An example of the assignment is shown below for reference:



Game Rules

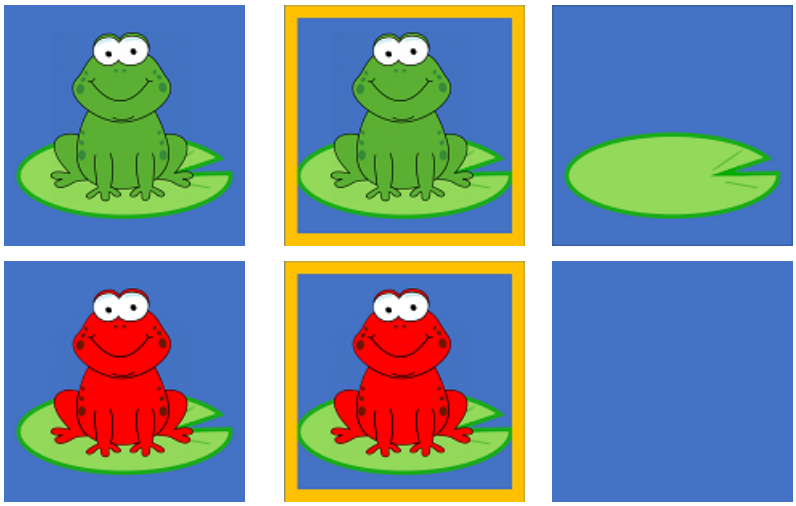
The rules of the game are very simple.

- The game board consists of one red frog, some number of green frogs and 13 pads.
- The frogs are placed on the pads at the start of the game in positions defined by a level card. There are 40 level cards in total, with varying levels of difficulty.
- The player can move one frog at a time. Frogs can jump horizontally, vertically or diagonally, but may only be moved onto a pad. Two frogs may **NOT** share the same pad.
- A frog can only jump to a different pad if they jump **OVER** another frog onto an empty pad. When this happens, the frog that was jumped over is removed from the game board.
- The player wins if (and only if) the red frog is the only frog on the board. (i.e. all green frogs have been removed).

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.

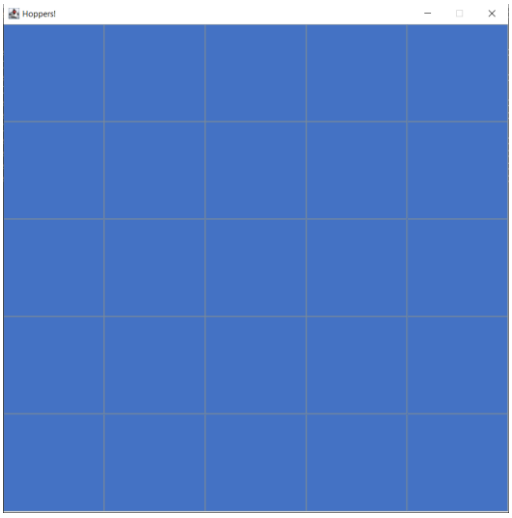
This zip file gives you access to the following images. Feel free to make use of these in your program:



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 called **Board** to represent your Graphical User Interface (GUI).
- Create a class called **Square** to represent a clickable square on your GUI. Think carefully about which Swing class might help you here.
- Add further instance variables to your **Square** class so it can hold information about its location on a board. Write accessor (get) methods for your instance variables.
- Write a constructor for your **Square** class that creates a square, and records its location based on parameters to the constructor.
- Write a constructor for your **Board** class that uses Swing to show a window, creates 25 instances of your Square and uses this to create an empty board consisting purely of water.



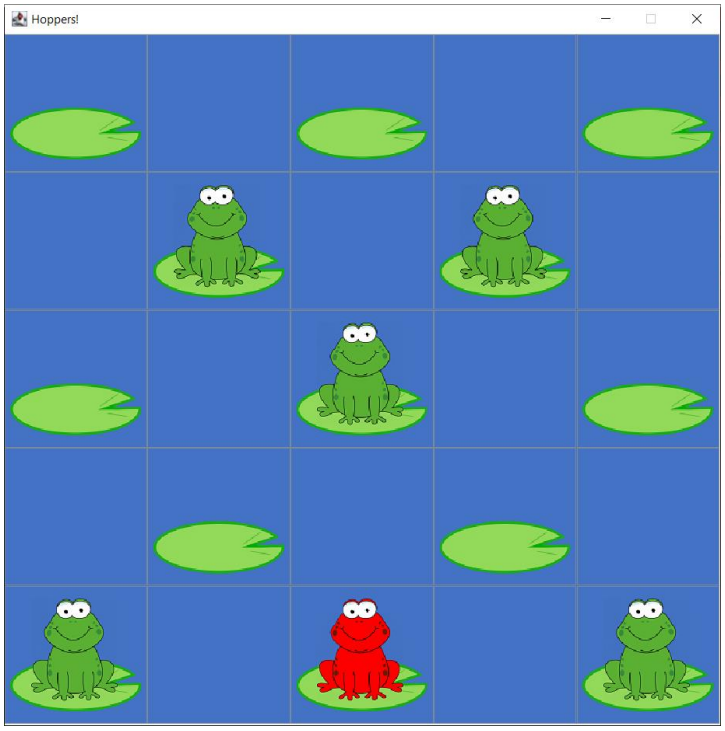
HINT:
Images in Swing applications are represented by a class called **ImageIcon**. The ImageIcon class contains a constructor that lets you create an instance of an ImageIcon from a given filename. The **JButton** class is able to create buttons showing images as well as text… In particular, the JButton class has a **constructor** that allows a JButton to be created from an ImageIcon. It also has **accessor** and **mutator** methods (getIcon and setIcon) for the image currently associated with a JButton.

```
ImageIcon i = new ImageIcon("Water.png");
JButton b = new JButton(i);

b.setIcon(i);
```

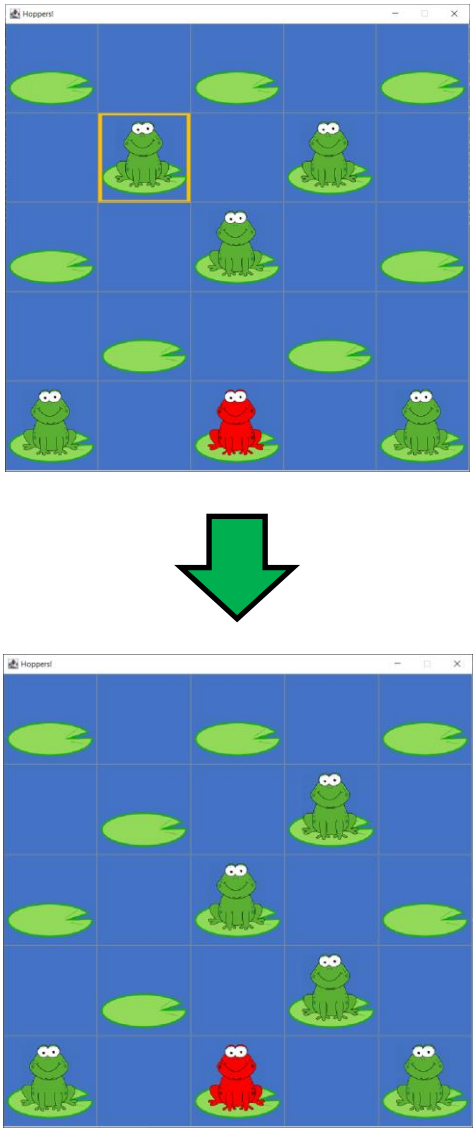
Task 2: Adding pieces…

- Add an instance variable to your Square class so that each Square knows what piece, if any, is contained on that square.
- Update the constructor in Square so that it takes an additional parameter – the type of piece (if any) that is to be placed on the square. Store this information in the instance variable you just created and write code so that the appropriate image is shown on the GUI.
- Update your Board class to create a Board that is consistent with the following level:



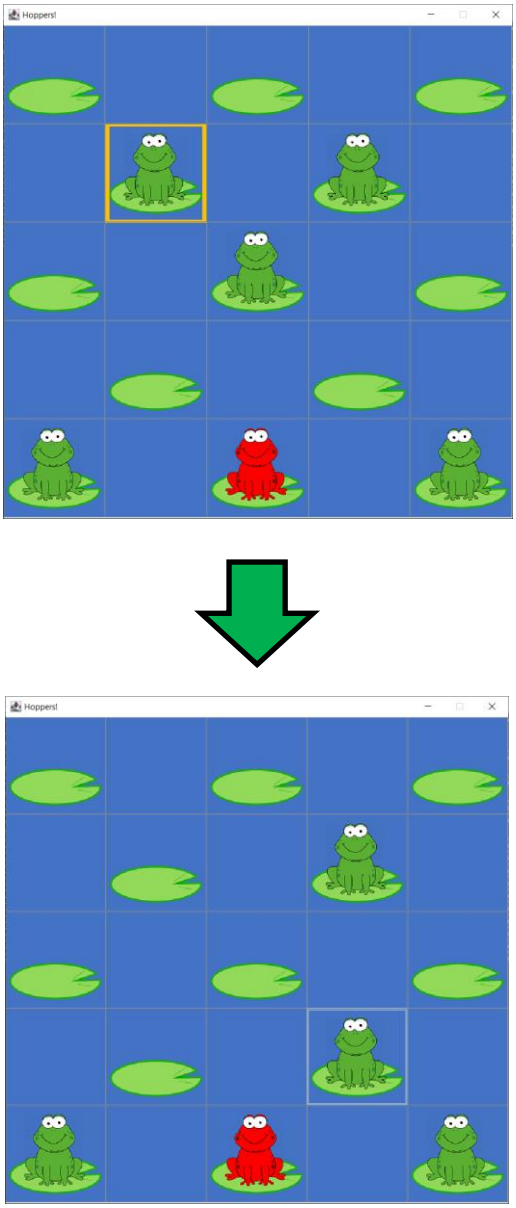
Task 3: Moving Pieces…

- Develop your Board class so that it can detect when any of the Squares are clicked with a mouse.
- Write a **moveTo** method for your Square class. This should take another Square as a parameter. When invoked, this method should move the piece on that square to the square provided as a parameter.
- Update your Board class so that when a user clicks on a square containing a piece, a subsequent click to an empty square will move that piece to the square indicated. You should use your moveTo method to achieve this.
- You should also make use of the additional images provided to highlight any piece that is currently selected.



Task 4: Ensure Legal Moves…

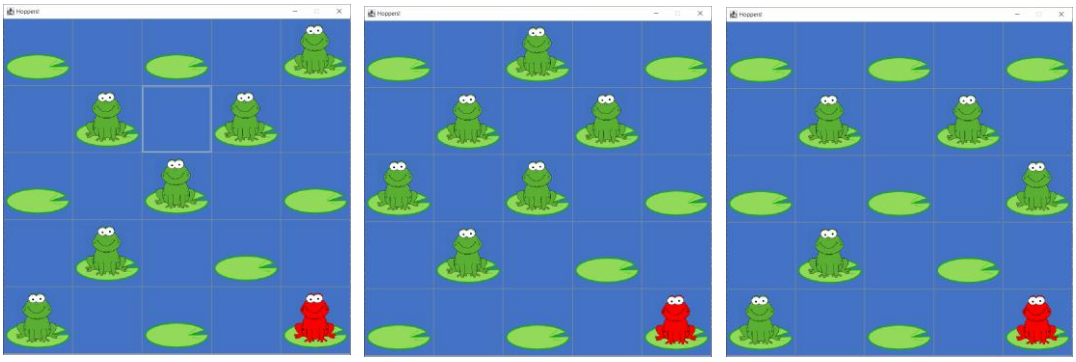
- Develop your software such that it will only permit legal moves. Any attempt to perform an illegal move should be ignored.
- Update your software to accurately remove pieces that have been jumped over.



Task 5: Additional Features…

You have the fundamentals of a complete application. Develop your program to add more features to improve the users experience:

- Determine when the game is won and display an appropriate message.
- Add support for multiple levels. Be sure to include a simple way for users to select which level they want to play. Three levels are listed below. Images of all 40 levels in the game will be available on request in your scheduled lab session.



Hacker Edition

If you have completed all the tasks, 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. Or perhaps a level designer, or even a level solver…

Lancaster University

Assessment

This work will be assessed through a practical demonstration and code inspection of your work.

Be prepared to show your program in your Week 20 practical session, and to answer questions about it posed by your markers.

To gain marks from this assignment:

- **You MUST submit your code to Moodle by the advertised deadline.**
- **You MUST demonstrate your work IN YOUR OWN LAB SESSION.**

FAILURE TO ADHERE TO THE ABOVE MARKS WILL RESULT IN A MARK OF ZERO FOR THIS EXERCISE. THE UNIVERISTY'S TYPICAL LATE SUBMISSION PENALTY DOES NOT APPLY TO THIS EXERCISE.

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	50%
Code Structure and Elegance	20%
Code Style and commenting	10%
Use of Version Control	10%
Ability to Answer Questions on Code	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.

N.B. IF YOU USE TECHNIQUES BEYOND THOSE TAUGHT IN THE COURSE, YOU MAY BE SIGNIFICANTLY PENALISED IF YOU CANNOT FULLY EXPLAIN THEM.

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 and 2.
D: Working program meeting all requirements of Task 1.
F: No working program demonstrated, or 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: One code block showing poor naming, scope or indentation or occasionally vague and/or inaccurate comments.
C: Two or Three code blocks showing poor naming, scope or indentation, code is partially commented **or** systematically vague and/or inaccurate comments.
D: Four or Five code blocks showing poor naming, scope or indentation or comments.
F: Five or more code blocks showing poor naming, scope or indentation or comments.

Version Control:
A: Project is versioned using a private git repository with a strong and timely commit history
B: Project is versioned using a private git repository with a strong or timely commit history
C: Limited use of git, commits are infrequent or weak commit messages
D: Limited use of git, commits are infrequent and weak commit messages
F: No use of version control

Ability to Answer Questions on Code:
A: All questions answered in detail.
B: Student failed to explain one technical question about their code.
C: Student failed to explain two technical questions about their code.
D: Student failed to answer questions, but could provide a basic overview of their code.
F: No evidence that the student understands the code being demonstrated.

Star Categories:
By showing a demonstrable example of additional work and/or good programming practice, your marker may choose to award a star (*) grade to your work. If you feel your work shows additional merit beyond the specification, it is your responsibility to make your marker aware of this during your in lab marking session. Examples of a star award would be for a substantial number of additional features, such as those described in the Hacker Edition section.