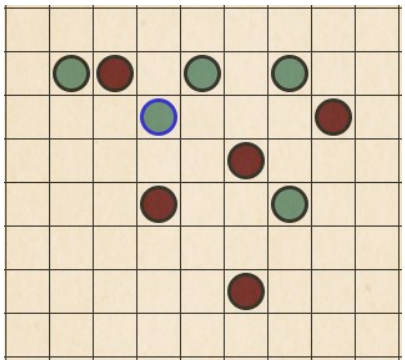
COMP10062: Assignment 8

The Assignment

In this assignment you will implement some basic functionality for a grid-based game app. This is a summative activity that will require the use of encapsulation, inheritance, polymorphism, association, exception handling, and **ArrayLists**.

Basic Performance



Many tabletop games are played on a grid (Chess, Checkers, Connect 4, Tic Tac Toe, and miniature-based combat games for example).

The basic tabletop grid for this assignment should have the following functionality:

ADD

The user should be able to add pieces to the grid by clicking on the cell (square). Pieces should appear in alternating colors. For example, first click player 1's piece will appear (let's say green circle in this example). The second click player 2's piece will appear (brown circle in this example).

REMOVE

The user should be able to remove pieces by entering a row number and column number in text fields (not pictured in above example) and selecting the "remove" button.

SELECT AND MOVE

The user should be able to select pieces by clicking on a piece. Selecting a piece should cause it to draw itself differently – perhaps with an outline or a change of background color. An example of a selected piece is shown above with the green piece outlined in blue. Once the user has selected a piece, they can click an empty cell to move the piece there.

THE PIECES

Each type of piece (in this example the green and brown circles) should be represented by its own class with a **draw** method, and you should use inheritance and abstract classes effectively when designing them.

Each time a piece is created, it should be stored in a polymorphic **ArrayList**. When a piece is added, removed, or changed, you should clear the screen and redraw the grid along with every game piece in the **ArrayList**.

Your app should not look exactly like the example shown above.

How to Snap to a Grid

Let's say you use a cell size of 40 pixels high by 40 pixels wide. The top left corner of the cell (square) at row **r** and column **c** is at **x = 40*c** and **y = 40*r**. This simple calculation will help you draw the grid lines and place the pieces.

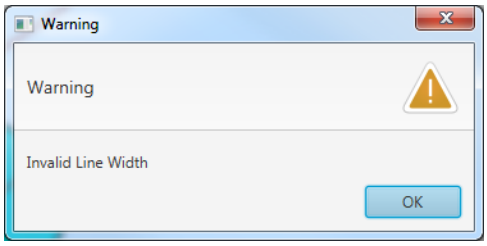
To find out where a user clicked on the grid, cast their **x** and **y** coordinates to integers, then **x / 40** gets you the number of the column they clicked on and **y / 40** gets you the row.

Error Handling

It should be impossible for the user to trigger an unhandled exception. If a **TextField** or other element cannot be converted to a numeric value, you should catch the exception and display an error message.

Similarly, if a user tries to move a game piece on top of another piece or remove a piece from an empty cell, you should disallow it and show an error message.

You can display a message on screen in a Label control or use a dialog pop-up to do this as shown below:



The warning above was created with the code shown below:

```
new Alert(Alert.AlertType.WARNING, "Invalid Line Width").showAndWait();
```

Optional Extras

There is so much more that you could do with this! All the ideas below are optional.

- Allow a larger variety of pieces
- Implement two “teams” of pieces in separate ArrayLists
- Make the board bigger than the canvas. Allow the user to zoom in and out and move the visible portion of the board around
- Add more types of pieces with different allowable moves
- Build on the basics to implement a playable game
- An ArrayList is not the most time-efficient structure for storing grid pieces. Research multi-dimensional arrays and then store the grid pieces in a two-dimensional array instead.

Handing In

See MyCanvas for submission instructions.

Make sure you follow the **Documentation Standards** for the course.

Evaluation

Your assignment will be evaluated for performance (40%), structure (40%), and documentation (20%) using the rubric on Canvas.