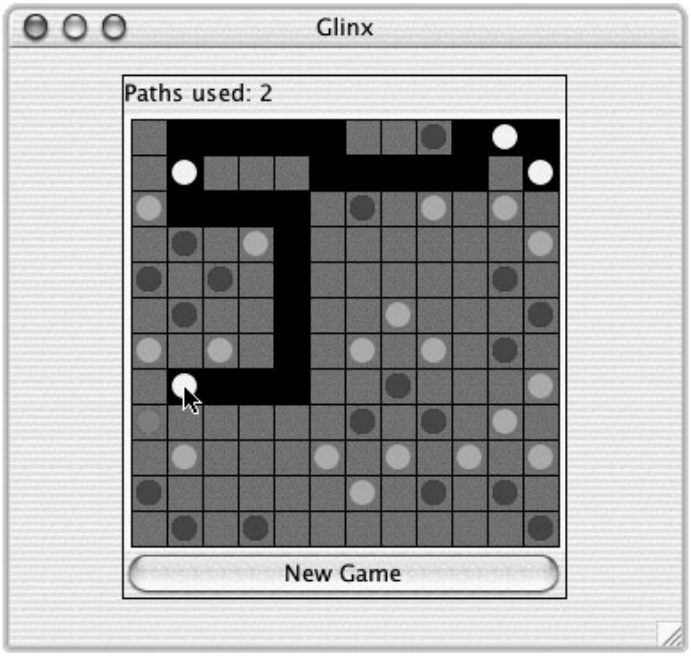
# Final Programming Project
## MiniGlinx

**Objective:** Build a simple arcade game using much of what we have learned so far.

For this assignment, we would like you to implement a game designed to somewhat resemble the game named Glinx that can be found at http://www.gamehouse.com.

Like the original game, our game is played on a grid. The grid is filled with tokens of different colors. Two or more tokens of the same color can be linked together to form a path as long as the path does not cross itself and does not pass through any cell containing a token of a different color. Once a group of tokens have been linked to form such a path, they can be removed from the game board. The goal of the game is to remove all the tokens from the grid using as few paths as possible.

A sample of what the game display should look like as you play this game is shown below.




A player forms a path by dragging the mouse from one grid cell to another along the desired path. To start a new path, the player should point at a cell containing a token that is not already part of a path and depress the mouse button. Then, each time the player drags the mouse into a new cell, the program will either add a cell to the path or remove the last cell from the path. A new cell will be added to the path if the mouse is dragged into a cell that

1. is not already part of the path,
2. is adjacent to the last cell added to the path, and
3. is either empty or contains a token of the right color.

The last cell will be removed from the path if the cell into which the mouse has been dragged was the next to last cell in the current path. This will enable the user undo a path by backing up along it.

Once the player is happy with the path drawn, all of the pieces in the path can be removed from the grid by clicking once on the last cell of the path as long as this cell contains a token.

Your program should include a button labelled "New Game" as shown in the demonstration version. When the user clicks this button, the screen should be filled with a new, randomly selected pattern of tokens so that the player can start again. At the top of the screen, you should use a `JLabel` to display the number of paths the player has used so far in the current game.

**To get you started:** To assist you with this project, we will provide you with the skeletons of four classes named `Glinx`, `Board`, `Cell`, and `Path`. Of course, by giving you these skeletons, we are suggesting how you should perform the first step of the process of designing your solution to this problem, deciding what classes to include in your program.

The `Glinx` class is intended to serve as the `WindowController` for this program. The `Board` will represent the entire game grid which will be composed of many instances of the `Cell` class. While the user is dragging the mouse around on the board, an instance of the `Path` class will be used to keep track of the collection of `Cells` that have been selected by the player.

The `begin` method of the `Glinx` class will have to create the `Board` and add the `JLabel` and `JButton` to the screen. This class should also contain methods to respond to button clicks and mouse events as described above. You should try, however, to keep these methods as simple as possible by including methods in the other classes that will make it easy for the methods in `Glinx` to do things like tell which (if any) cell contains the mouse's position, add or remove a cell from a `Path`, remove the contents of all the cells in a path from the board, etc.

The `Board` class should be responsible for creating the inital board and distributing tokens on the board randomly at the start of each game. It is also the logical place to provide a method that would translate a pair of canvas coordinates into a `Cell` or at least the row and column position of a `Cell` within the board.

The `Cell` class should provide methods to add or remove a token from the cell and to find out the color of the token (if any) currently in the cell. It may be useful to have the `Cell` remember whether or not it is in the current path.

The `Path` class is a collection of cells. It can be implemented using either arrays or recursive lists. You are encouraged to use a list but free to take either approach. You clearly need a way to add a `Cell` to a path and to remove cells. You will also need a way to remove the tokens from all cells in a path.

Of course, as you work on the details of your own design for this program you may decide you would like to add additional classes and/or eliminate some of the classes we have suggested. You are allowed to do this.

**Writing the program:** As usual, we strongly encourage you to construct your program in a step by step fashion. This should make the process of writing and testing the code simpler. It is generally easier to debug a program if you know that some parts do run correctly. In addition, it ensures that you can turn in a running program even if you have not completed all the requirements for a complete solution.

- Run the demonstration program.
- Construct a program that displays the two GUI components (the path length Label and the Button).
- Define constructors for the `Cell` and `Board` classes. Add code to your `begin` method so that when the program is run an empty board is displayed.
- Add a method to the `Board` class to randomly distribute colored tokens on the grid in the "every other square" pattern shown above.

  Initially, you should just choose a random color for each square that is supposed to contain a token. Eventually, we would like you to implement a mechanism for distributing the tokens in such a way that the number of tokens of one color is no more than one more or less than the number of tokens of any other color.
- Make the "New Game" button work so that each time it is clicked a new set of tokens replaces the old ones.

- Define a `Path` class that can hold any sequence of `Cells` and add code to the mouse event handling methods in the `Glinx` class so that you can create paths by dragging the mouse. At first, don't worry about whether all the cells in a path are the same color or even adjacent on the board.
- Make it so that when you start a new path, the old path disappears (without removing any tokens from the cells).
- Make it so that when you click on the last cell of a path, the contents of all the cells in the path are removed.
- Add the code to make sure that a cell can only be added to a path if it
  1. is not already on the path,
  2. is empty or contains a token that is the same color as all the other cells in the path,
  3. and is adjacent to the last cell added to the path.
- Refine your code for picking tokens so that you always generate as close to the same number of each color as possible. For some of you, this may be the hardest part of the assignment.

  To make this a bit easier, you may want to choose your grid size so that the number of cells is divisible by 2 times the number of token colors you use. That way, you can have exactly the same number of each color.

  It may seem easiest to pick a random color for each cell that needs a token, but working the other way around will probably be simpler. That is, put all the Cells in which tokens need to be placed in an array so that you can pick Cells randomly. Then, for each token you need to place, pick a random Cell.
- Finish up any remaining loose ends and test it throughly.

**Extra credit suggestions:** Here are a few suggestions for extra credit features that you can add that will make the game more fun to play. Those who are inspired to add extra features to raise their grades should recognize that most of your grade will be determined by the quality of the code for the basic problem including both its correctness and clarity.

1. Add an "undo" button to undo your last move if you mess up. (It is very frustrating to have to start all over if you make one mistake.)
2. Embellish your `Board` class so that it can lay out tokens in several patterns like the original Glinx and use these extra layouts as advanced levels of the game.
3. Implement a scoring mechanism more like that used in the original game. Give points for the number of tokens in each path removed in a way that favors long paths. Give extra points for each token removed if the path removed includes the last remaining tokens of a given color.