Preamble

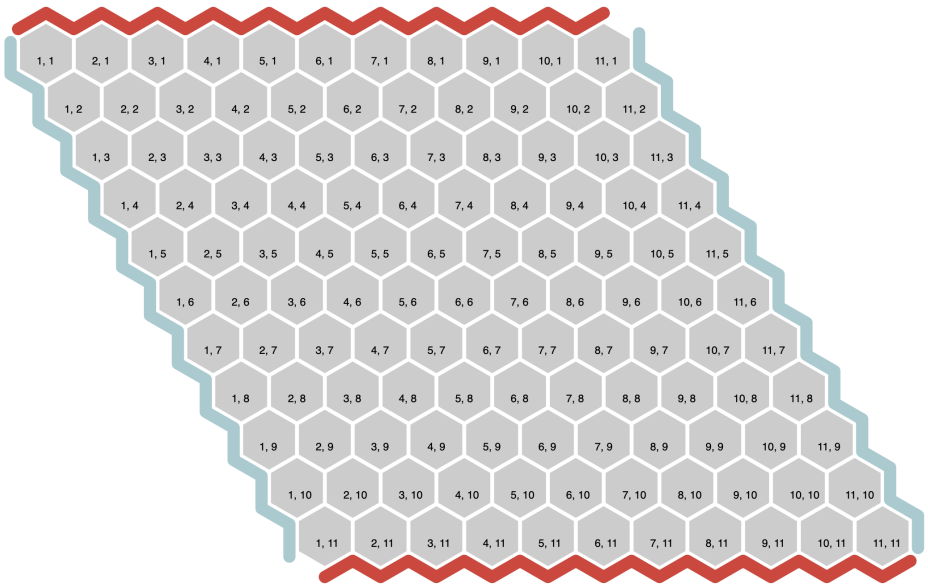
As before. Submit the `Hex.java`.

The Hex game

This topic covers programming the game Hex.

Hex is a two-player game called *Red* and *Blue*. It is played on a diamond-shaped board tiled with hexagonal squares, any size, but typically 11×11. Each player has two opposite edges of the board. Red goes first. Players take turns capturing empty squares on the board by marking them with their colors. The first player to connect their two edges with a sequence of adjacent squares of their color wins the game.

note. If we play on an $n \times n$ board, we can identify the squares by coordinates $(\texttt{i}, \texttt{j})$ with `i` and `j` begin integers between 1 and n inclusive. We agree that Red has the top and bottom edges, while Blue has the left and right edges.



1 The code

`HexGUI` implements a graphical interface, which relies on the class `Hex` to represent a game state and implement the rules. You can already launch `Hex`, and click on the boxes, but nothing happens because it is up to you to code the class `Hex`. (Note, however, that the key `C` allows you to display the coordinates.) As you go through the questions, you will need to add fields and methods, if necessary, and complete the constructor `Hex(int n)`

note. Some interface features are accessible using keyboard keys:

- `R` resets the board
- `Space` plays a random move
- `L` displays on each of the box the result of `label(i, j)`
- `C` displays coordinates
- `P` displays the colors of the boxes (useful for checking that the edges are well initialized)
- `A` plays a full random game

1.1 State of the boxes

Implement the method `Player get(int i, int j)` that returns the player who owns square (i, j) (`Player.RED` or `Player.BLUE`), or `Player.NOONE` if that square has not yet been played. To represent the edges, we will allow i and j to take the values 0 or $n + 1$.

One can introduce a field `Player grid[][]` that is correctly initialized.

Run `Hex` and press the key `P` to verity the initialization.

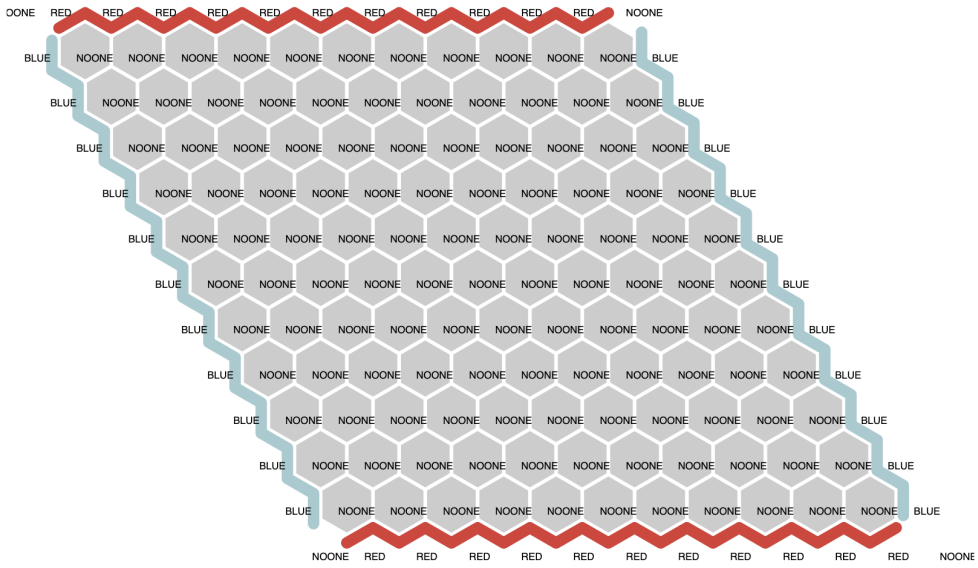


Figure 1: A properly initialized board

1.2 Game rounds

Implement the method `boolean click(int i, int j)` whose call signals that the player with the move plays square (i, j). If it is a legal move, the method updates the board state and returns `true`. Otherwise it does nothing and returns `false`.

Implement the method `Player currentPlayer()` that returns the player with the trait.

You should be able to play `Hex` by running `Hex`. The key `R` resets the board state. The space bar (which you can hold down) plays random moves. The key `A` shuffles a full game.

1.3 Victory

We would now like the interface to signal when one of the players wins the game.

Implement the method `Player winner()` that returns the player whose position is winning, or `Player.NOONE` if neither player has won yet. We can use a union-find data structure and associate the integer $i + (n + 2)j$ with the cell (i, j).

Change the method `currentPlayer()` to return `Player.NOONE` when one of the players has won the game.

To make debugging easier, you can modify the method `int label(int i, int j)` to, for example, return the related component ID of the box (i, j) . These IDs can be displayed in the GUI by typing the key `L`.

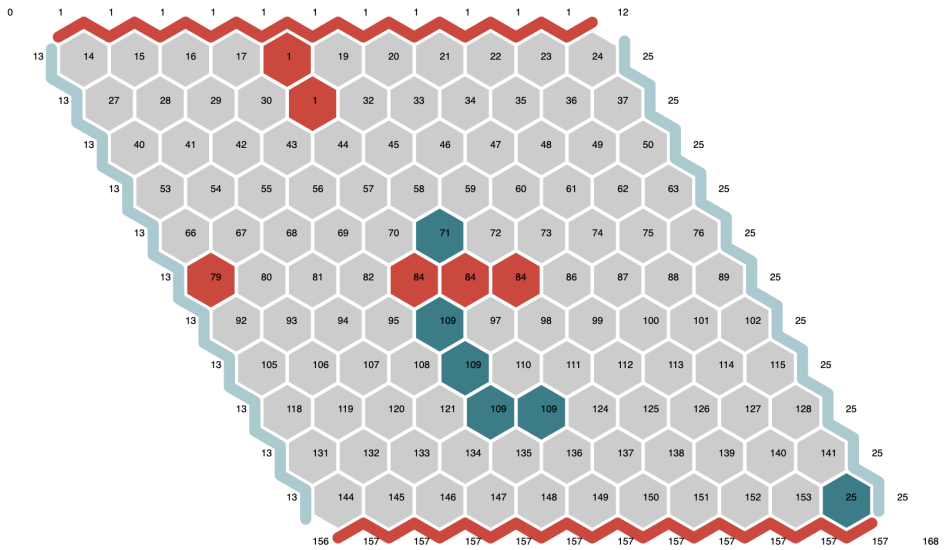


Figure 2: Example (strategically not very inspired) of a board with the display of the identifiers of the related components.

1.4 If you have time

Challenge your classmates.