ALGONQUIN COLLEGE

ASSIGNMENT 1.1 – GUI for Game (Piccross[1])

General View

Due Date: prior or on **Oct 9th 2021 (midnight)**

- **2nd Due date** (until 16th Oct) - 50% off.

Earnings: 10% of your course grade.

Purpose: Create the interface and basic organization of the Game Piccross.

- This is the first task in JAP. This application will be developed during this semester and the code will be updated to incorporate new functionalities.
- **PART I:** What you need to create the interface considering the model:
  - A basic splash screen to start the application.
  - A basic controller (to be increased in the next assignments).
  - The main window for the game (Piccross).
- **PART II:** Additional elements are required to define:
  - Code documentation (using Java code conventions)
  - Javadoc files.
  - Executable Jar.

Note 1: About Teams
As mentioned in CSI, teams are acceptable for submission obeying the rule that both students should belong to the same Lab Session. And remember that demos are required to get points in specific weeks when developing the code. In this case, it is required to include a kind of git information (it can be given by screenshot, git status, etc.) that can identify that both students are developing the code.

[1] The name "Piccross" is intentionally changed in comparison with the original "Picross" game since some functionalities will be different. This idea was originally proposed by Prof. Daniel Cormier, instructor in some labs this semester.

Part I – Creating the Piccross

1.1. BASIC GAME

The **Piccross** (based on Picross) is a simple visual game in which you need to be able to identify the picture defined in a cross board using one specific logic.

Some references for this game can be found easily on the internet. For instance, you can play an interesting version using the following site: http://picross.net/

Example:

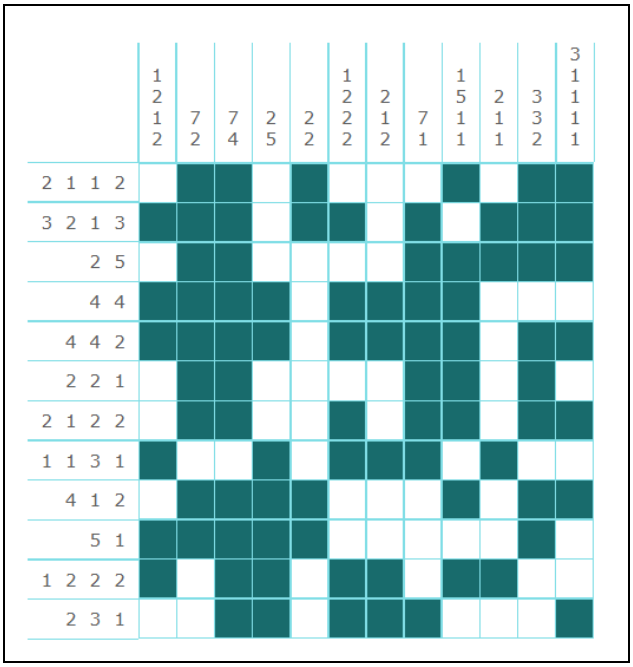

Fig. 1 – Picross example

- In short, each row and column indicate a sequence of numbers that are indicating how many blocks (squares) should appear.
  - The initial board is completely "blank" and the user must consider the logic to be able to select appropriately the correct distribution of squares.
  - For instance, the sequence (5,1) is indicating that we should find a contiguous sequence of 5 squares followed by some spaces (blank squares) and another one.
  - The problem is that we do not have any idea about this distribution.
  - Our game (Piccross) is a basic adaptation that is using the same idea.

1.2. SCREEN SPLASH

The beginning of the game must include a splash screen (see Fig. 2).

Fig. 2 – Splash screen

Class GameSplash

The class **GameSplash** will manage the splash screen. This screen should stay during some seconds (for example, between five to ten seconds) and then, close to let the main application appears.

- Change the original image to include your identification (or your team).

1.3. GAME CONTROLLER

GameController will assemble the UI, primarily in its default constructor.

This class must also have an **inner class** called **Controller**. You will use this to manage the visual components.

The **Controller** class will have very basic code. It reads the action command (or JavaFX equivalent) of the button that was pressed, and then prints it to the console. All buttons (and the checkbox) must have a unique action command.

- **NOTE**:
  - If you are using Swing, **GameController** must extend JFrame. You may have to define your JFrame parameters in GameController instead of the **Piccross** class.
  - If you are using JavaFX, **GameController** may extend Scene, Pane or any descended Pane subclass as you see fit.

1.4. BASIC GAME INTERFACE

1.4.1. General Interface

The **Piccross** has a very simple interface, as indicated in the following figures.

In the beginning, the interface is supposed to show a neutral panel:

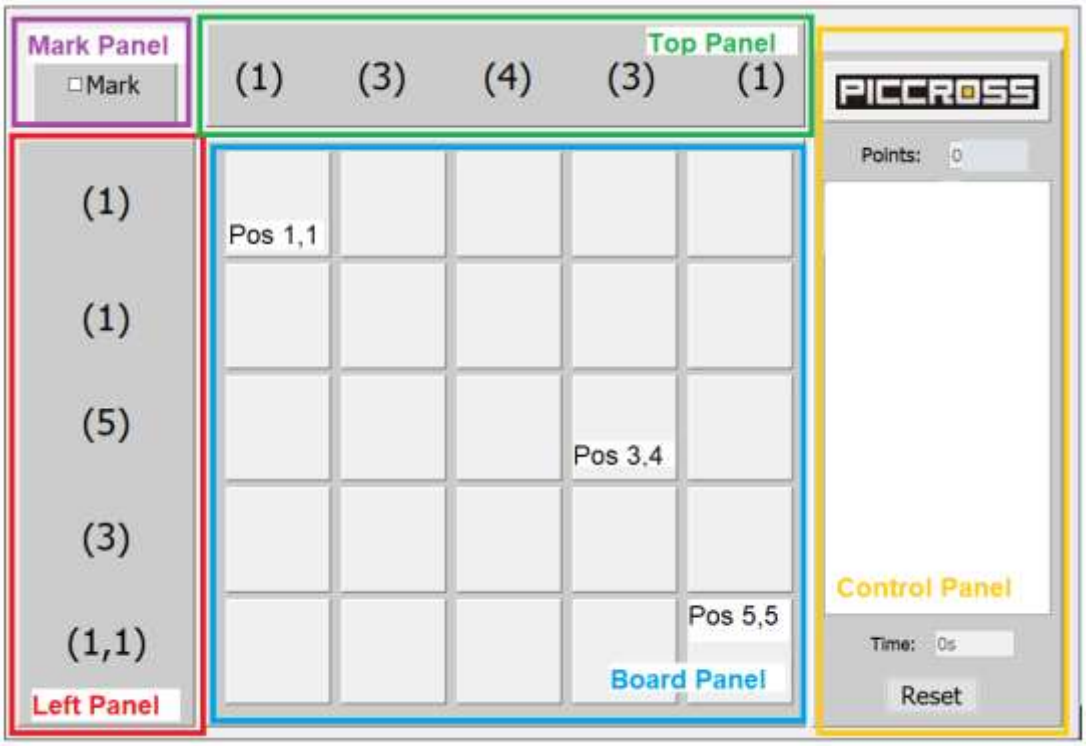

Fig. 3 – Initial game *(colors are only descriptive)*

Note that this game is composed by several panels:

- **Board panel**: a set of panels that will compose the board of panels that can be changed using colors (using the checking box “mark”).
- **Mark panel**: When we set the control for mark or not the squared.
- **Left panel**: Indicating the number of squares to be shown by each line (use the set of **Labels** for this).
- **Top panel**: Indicating the set of squares to be shown in each column (the **Labels** are using the number of squares).
- **Control panel**: Where you can see the following components:
    - **Image logo**: Using the Piccross.

- **Points**: Showing the points at this moment.
- **History Area**: To describe the sequence of movements performed by the user.
- **Time**: To show the number of seconds since the beginning of game.
- **Reset button**: To restart the game.

1.4.2. Initial Behavior

The purpose for this initial assignment is that any action can be shown in the **history area** at Control Panel. The messages to be printed are supposed to be like this:

- Message type 1: When checkbox “Mark” is selected or unselected.
- Message type 2: When one specific panel that represents a square is clicked.
- Clean message: When “Reset” button is pressed.

The list of messages is sequential. It means that you need to “index” the sequence. Ex:

```
…
Mark set;
Pos 1,3 clicked;
Mark reset;
…
```

- For while, it is not required to use routines to count the time (in seconds since the beginning of the game), neither to test if the current layout is matching with one specific picture.

1.4.3. GUI Restrictions

The definition for all colors and sizes are not formalized here, since the general layout is respected. However, we have general restrictions:

- All classes must belong to one specific package (“**piccross**”).
- This layout uses Java’s default look and feel.
- The frame (or stage) must not be resizable.
- Your name / team name must appear in the title bar.
- You may ONLY use BorderLayout, FlowLayout and GridLayout (or the JavaFX equivalent).
- This UI must be assembled manually.

- The use of UI builders is not permitted this time and will imply in deductions.

1.4.4. EXTRA: Preview about Future Version

Note 2: Future Game Behavior
The activity now is only related to the game interface and basic behavior. The following description is anticipating some aspects of the game that will be updated / changed in future assignments.

- *In the future implementation, the game is supposed to match with one specific pattern (the original "picture" behind the board).*
  - *Each square correctly clicked is giving **1 pt** in the game (not used for while).*
  - *Wrong selection (in comparison with the internal representation) will stop the game showing the marks in the end.*
  - *This is the reason to use the "**Reset**" button.*
  - *Once clicked, the colors of square panels will be changed (green / red) according to the pattern to be used.*

Example:

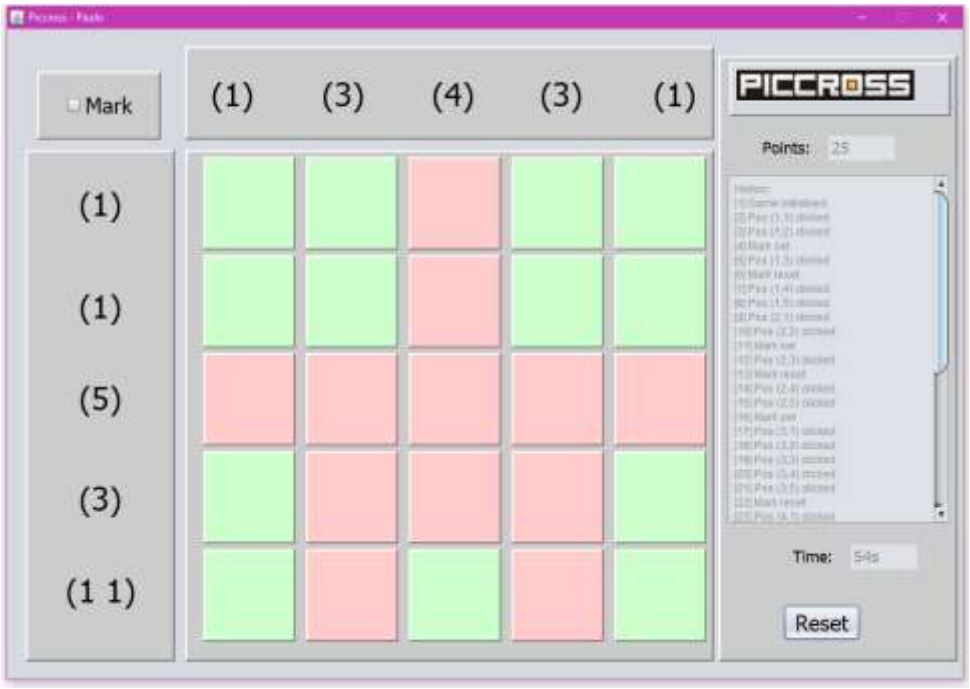

***Fig. 4** – Main window (after execution)*

Visually, the user actions are using mouse click over the panels will observe the "mark" (if is set or not) according to this basic rule:

- *Correct square clicked (and "mark" is not set): color green is shown.*
- *Correct square clicked (using marks): red color is shown.*
- *Incorrect selection: game finishes.*
- *For this assignment, the original behavior (the internal pattern) can use a local array representation. But in the future, it will not be "hard-coded"*
    - *The pattern must be loaded dynamically. For example, the above representation for the image is a kind of file using the following input (in binary):*

```
00100
00100
11111
01110
01010
```

- *Or, considering decimal representation:* `4 4 31 14 10 .`

Part II – Documentation and Packing

2.1. BASIC DOCUMENTATION

CODE ELEMENTS (*Basic criteria*)

- Code well organizes, methods and functions grouped logically, classes in order, etc.
- Label naming (variables, constants, methods, functions, etc.) consistent and meaningful.
- Code consistently indented and spaced (define a consistency /organization).
- Good commenting which includes block comments.
- In the case of teams, Git files are required to be include.

JAVADOC (*following Java Standard*)

- Javadoc outputs are required, and the pages / folders must be included.

JAR (*Executable file*)

- Finally, it is required to create and include JAR files (that can open the binary using Java)

Note 3: Read Assignment Standard
The code and patterns to be used in submission must follow the Java Code Conventions, but also the elements described in the Assignment Standard.

Evaluation

- Please read the Assignment **Submission Standard and Marking Guide** (at "Assignments > Standards" section).
- **About Plagiarism**: Your code must observe the configuration required (remember, for instance, the "splash screen" using your name. Similarly, we need to observe the policy against ethic conduct, avoiding problems with the 3-strike policy...

Marking Rubric

Maximum Deduction (%)	Deduction Event
	Severe Errors:
100	Late submission (after 1 week due date)
up to 100	Does not compile with javac
up to 100	Compile but crashes after launch
up to 100	**Does not comply with the problem specifications**
up to 50	The GUI appearance is not identical with the specified GUI and provided images.
up to 30	Using different containers and layout managers than those specified
up to 10	Inadequate comments (headers, code explanation), insufficient javadoc
up to 30	Missing or incorrect classes and package
up to10	Bad Controller specification (not inner class or displaying action command)
up to 30	Missing component (various text fields, buttons, labels, graphics)
10	Missing splash screen
up to 10	Other minor errors
up to 10	Bonus: clean code, elegant code, enhancements, discretionary points.
Final Mark	**Formula:** 10*((100- ∑ penalties + bonus)/100), max score 10%**.**

Submission Details

- **Digital Submission: Compress** into a **zip** file with all files (including source code, images, Javadoc, Jar).
- Upload the zip file on Brightspace. The file must be submitted prior or on the due date as indicated in the assignment.
- ***IMPORTANT NOTE:*** The name of the file must be **Your Last Name** followed by the last three digits of your student number followed by your lab **section number**. For example: **A11_Sousa123.zip**.
  - If you are working in teams, please, include also your partner. For instance, something like: **A11_Sousa123_Cormier456.zip**.
  - **Remember:** Only students from the **same section** can constitute a specific team.
- ***IMPORTANT NOTE:*** **Assignments will not be marked if there are not source files in the digital submission.** Assignments could be late, but the lateness will affect negatively your mark: see the Course Outline and the Marking Guide. All assignments must be successfully completed to receive credit for the course, even if the assignments are late.

File update: Aug 02nd 2021.

Good luck with A11!