

COMP1406 - Assignment #3

(Due: Tuesday, March 6th @ 9:00pm)



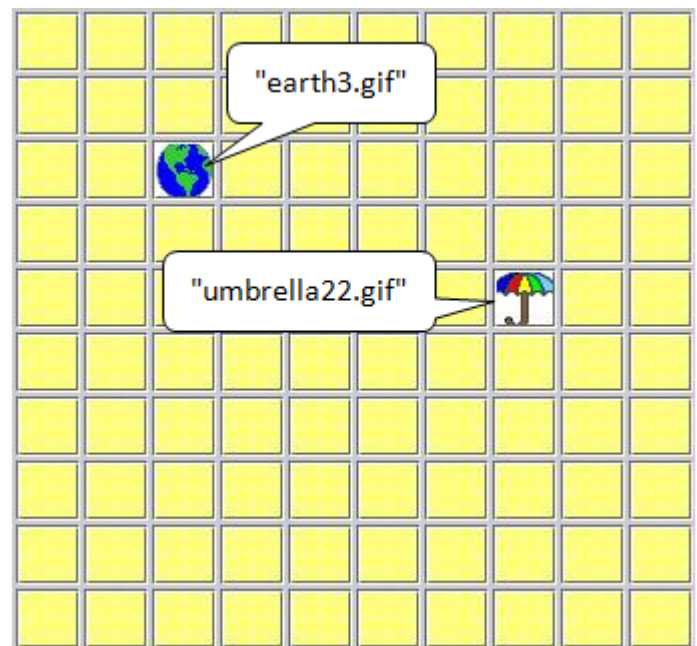
In this assignment you will create an application which represents the "memory" game (a.k.a. "Concentration"). You will learn how to connect various GUI components together as well as to an underlying model. You will also get to use the **Timer** class.

The game of memory consists of a bunch of cards (or tiles) with a picture on one side of them. There are exactly two cards for each picture. So a 10 x 10 grid of cards would have only 50 unique pictures, each duplicated once. The game begins by shuffling the cards and then arranging them in a grid fashion by laying them upside down so that the pictures are not showing. The player then flips over two cards. If the pictures on the two cards match, they are removed from the playing area (see picture below). Otherwise, they are both turned back upside down. This process repeats until no more cards remain. The point of the game is to see how long it takes to match all the cards. Or perhaps you set a time limit and see how many matches you can find within the time limit. That's it! There's not much more to the game :).



(1) The Model Class

Create a class called **MemoryGame** which represents the model for our game. It should keep track of a 2D array of tiles (or cards). You can decide if you want to make a **Tile/Card** class, but in essence, each "tile" needs only to be a String which will contain the name of a picture file. Your game should also keep track separately of the number of **guesses made** (i.e., each pair of flips is considered one guess), the number of **correct guesses** (i.e., correct matches), and the number of **remaining matches** (e.g., a 10x10 grid would start off with 50 remaining matches). Write whatever code you feel is necessary but you should have the following functionality:



- All MemoryGames should have tiles arranged in a 10 x 10 grid (i.e., array)
- Your should have the ability to detect when the game is completed (e.g., write an **isOver()** method that returns a boolean).

Each time a **new** MemoryGame is made, it should create randomly arranged tiles with names that match files contained in an **icons** folder submitted with your assignment. This folder should contain various **.gif** or **.jpg** files representing the pictures to be used in the game. You can download a zip

In the next part of the assignment, you will be able to hide/show pictures by using this same **setSelected()** method as the user flips cards over. Ultimately, you will use **setEnabled(false)** once matches have been found to grey-out the pictures and disable the buttons.

Note that each time this window is run, the icons are arranged randomly (with 2 copies of each picture). Make sure that you have a **main** method that brings up a similar-looking window. The TA will use this application to determine if your tiles are being placed randomly.

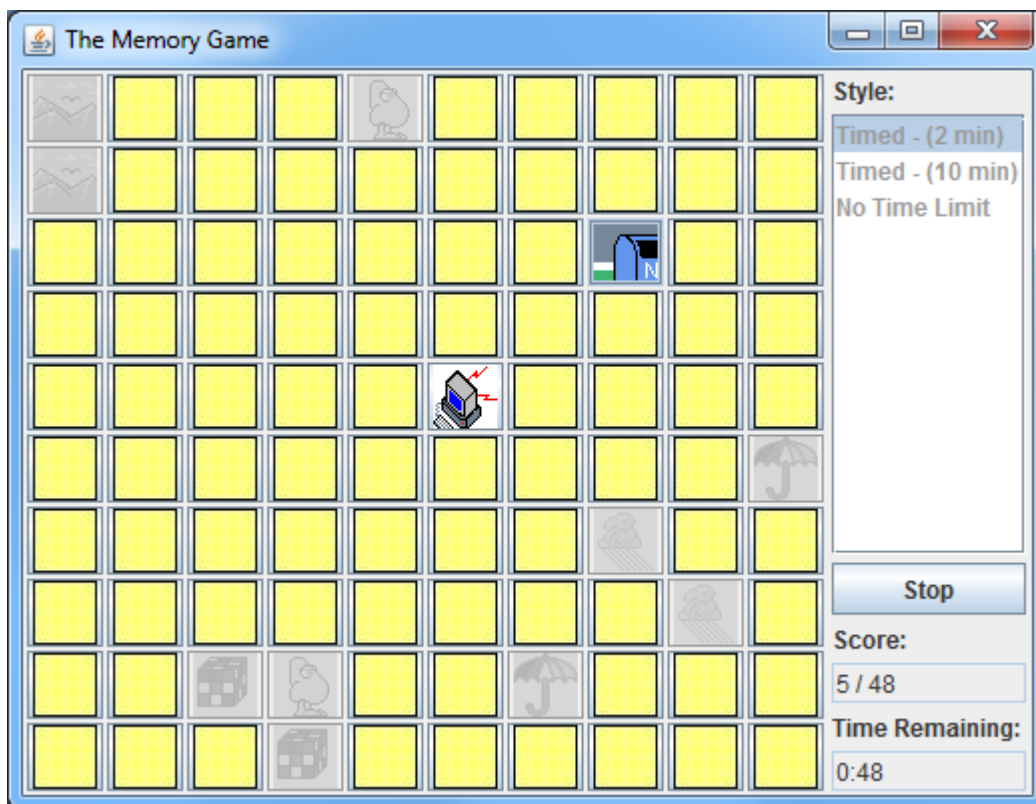
Lastly, to eliminate the border around the inside of a button (otherwise the picture will be cut off), you can use:

```
aButton.setMargin(new Insets(-30,-30,0,0));
```

It is ok if part of the picture is cut off ... just try to get it looking roughly as shown in the picture above.

(3) The Game

OK, finally we will make the completed GUI. The game should look roughly as shown on the next page. You will notice that the panel from part (2) is being used in the GUI (in this picture, the game is in the middle of being played). You must use **LayoutManagers** (of your choosing) to produce the window appearance ... do **not** use a **null** layout. Your window must be able to be resizable and still look reasonable afterwards.



Here are the rules for the GUI which you **MUST** get working. When the window is first opened:

- all buttons in the grid are disabled and show no pictures. Perhaps grab a common **blank.jpg** picture to use as your standard **JButton** icon.
- the **Timed - (2min)** game is selected from the list
- the buttons is enabled and says "**Start**"
- the score/time remaining fields have 0/0 and 0:00, respectively. These fields should always be non-editable.
- The user can select a game style from the list and then press **Start** to begin playing
- once the game is playing (keep a boolean instance variable), the list is disabled and the **Start** button's text is changed to "**Stop**".
- the grid buttons should all be enabled upon startup with no pictures showing (i.e., **setSelected(false), setEnabled(true)**)
- if a timer game is being played, a timer (see instructions at end of assignment) counts down each second and the remaining time is displayed in the text field each timer tick.
- the **Timed - (2min)** game style counts down from 2 minutes, the **Timed - (10min)** game style counts down from 10 minutes, the **no time limit** game style has no time limit
- if a timed style game is used, the game stops when the timer reaches 0
- when the game stops, all grid buttons are disabled, the timer is stopped, the scores remain visible, the **Style** list is enabled again and the **Stop** button is changed back to **Start**.
- When the user selects one button in the grid, its picture is shown. When a second button is clicked (as long as it is not the same one), that second button's picture is shown. The program should delay for 0.5 seconds when the second button is clicked, so that the user can see the picture. You will want to have the 2nd button draw immediately and then delay for 0.5 seconds. You can do this as follows:

```
button2.paint(button2.getGraphics());
try {Thread.sleep(500);} catch(InterruptedException ex) {}
```

- When the second button is pressed, this represents a chosen pair. Call a method in your model class to determine if these tiles match.
 - If the tiles match, their corresponding buttons are disabled, but still selected. You should see the pictures grayed out as above.
 - If the tiles do not match, they are both set to be unselected, but still enabled.
 - You'll have to think of how to determine whether a button click is a first selection or whether it is a second selection
 - If ever all matches are made, the game stops such that all buttons are disabled, the timer is stopped, the scores remain visible, the **Style** list is enabled again and button text becomes "**Start**".

You have freedom to write code in any way you please. However, you should make use of proper updating procedures that you learn in class. You may lose marks for bad coding style and/or for an ugly or poorly working user interface.

A timer is easy to use in JAVA. You create a **Timer** by specifying the amount of time that you want to elapse between timer ticks (in milliseconds) and you provide an **ActionListener** event handler (i.e., just as you do with **JButtons**). Here is an example that creates a timer that ticks every **500 ms** (i.e., twice a second):

```
Timer myTimer = new Timer(500, new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        handleTimerTick();
    }
});
```

Here is the event handler:

```
private void handleTimerTick() {
    ...
    update();
}
```

The timer will not start unless you tell it to. You can start and stop a timer, perhaps when the user clicks a button, by calling **myTimer.start()** or **myTimer.stop()** ... assuming that your timer was stored in a variable called **myTimer**.

NOTE: Submit (on WebCT) all **.java** and **.class** files needed to run. You **MUST NOT use packages** in your code, **nor projects**. Submit ALL of your files in one folder such that they can be opened and compiled individually in JCreator. Some IDEs may create packages and/or projects automatically. You **MUST** export the .java files and remove the package code at the top if it is there. Do NOT submit JCreator projects either. **JUST SUBMIT the JAVA and CLASS FILES**. Note that if your internet connection at home is down or does not work, we will not accept this as a reason for handing in an assignment late ... so make sure to submit the assignment WELL BEFORE it is due !

Please NOTE that you WILL lose marks on this assignment if any of your files are missing. You will also lose marks if your code is not written neatly with proper indentation. See examples in the notes for proper style.