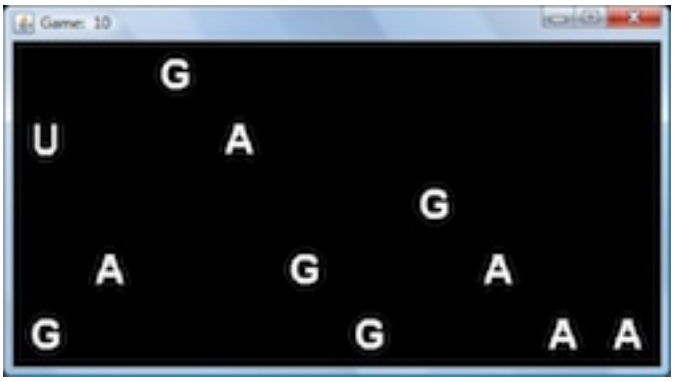
COSC102 Spring 2015

OOP 2 : Side-scroller Game Homework 4

Due date: **Monday, March 2nd, at 11:55 p.m. on Moodle**

In this assignment, you implement a computer game, which basic version display looks like the screenshot below. Once you have implemented this version you expand your code to create a unique game of your design.



Overview

The user controls an image (U) as shown in the screenshot, that moves up and down the left edge of the screen. Other images, Gs and As, appear at the right edge of the screen, and move steadily toward the left. There are two types of image:

- some that the user tries to get (G) and
- other that the user tries to avoid (A).

The user earns a score, shown in the tiled bar next to the name of the game. The game ends when a particular condition is reached. **And the rest is up to you!** In particular you choose

- the game's theme/story including its title, the images, their distribution, and anything else that determines the game's theme and playability,
- the scoring function, and
- the game-over condition.

Of course, you can do as much more as you like, as long as your game fits the description given in this document. Be sure to choose a theme for your game that is **appropriate for a general audience**. The improvement part of the grade takes into account your creativity: Mario games have been common so might not be considered too creative.

Release your game by posting your `YourFirstNameLastNameGame.class` file on Moodle forum so that your classmates can try it out. **Pay attention to** the following two things

1. the class name for your game starts with your name and
2. the file to post on Moodle forum is a `zip` file containing the `YourFirstNameLastNameGame.class` (which is *not the same than the* `.java`) and your images (the specific U, A and G conveying your game's theme).

You will lose 10 points if you don't post anything or if you post something else which is not the bytecode of your game. For example if you post by mistake a `.java` file on the forum the maximum grade you might obtain for your work is 90/100.

Start thinking now about ideas for your game. Your game MUST have a theme: games that perfectly reproduce only the features in the description below (i.e. similar to our demo) will receive about 70%. It is an open-ended assignment so you can practice creativity and aim for the level of difficulty you desire.

You are likely to find this assignment challenging. Make sure you leave yourself enough time to make your game interesting, as well as functional.

```
Essential Interface of Provided Classes

class Location
---------------
Location(int row, int col)
int getRow()
int getCol()
boolean equals(Location other)

class Color
------------
Color(int red, int green, int blue)
int getRed()
int getGreen()
int getBlue()
boolean equals(Color other)

static Color makeColor(int r, int g, int b)   // factory class method

class Grid
-----------
Grid(int numRows, int numCols)
Grid(int numRows, int numCols, int bColor)   // constructors to set up colors: background
Grid(int numRows, int numCols, int bColor, int lColor) // line color can help debugging

int getNumRows()
int getNumCols()
Color getColor(Location loc)
void setColor(Location loc, Color color)
String getImage(Location loc)                          //returns null if empty
void setImage(Location loc, String imageFileName)   //pass null image
                                                       //in order to remove an image
void setTitle(String title)
void pause(int milliseconds)
int checkLastKeyPressed()          //returns -1 if no key pressed since last call

class Cell
-----------
The class Cell is used by the class Grid. You don't have to use it directly,
feel free to look at it if you are interested!
```

Setup

1. Download ScrollingGame.zip, which contains the files provided for this assignment.
2. Unzip them in a new folder `hw4` in your `102` directory.
3. Open `Game.java`: **all the code you write for this assignment will go in this file.**
4. Rename the file `Game.java` to be `YourFirstNameLastNameGame.java` replacing with your actual name `YourFirstNameLastName` part.
5. Change the file `YourFirstNameLastNameGame.java` so that the program compiles.
6. Run the demo, which uses `MattGame.class`, by setting the class instance `boolean DEMO` to `true`: that's the basic game you have to replicate first.

While we refer to `Game` we mean `YourFirstNameLastNameGame.java`.

Basic game Description

Five instance variables have been declared.

- `grid` holds the grid used to store and display images.
- `userRow` keeps track of the row in which the user-controlled image (U) appears, on the left edge of the grid.
- `msElapsed` is initially used to keep track of the number of milliseconds that have elapsed since the start of the game. Its real purpose it to redraw at regular interval and not to draw too often. It will need be reset when the speed of the game is altered (step 9 below).
- `timesGet` keeps track of the total number of times the user collided with (G) images, which are the things the user collects in the game.
- `timesAvoid` keeps track of the total number of times the user collided with (A) images, which are hits the user is supposed to avoid.

Develop the `Game` class progressing with the following steps.

1. There is a parameterless constructor and a second constructor permitting to vary the dimensions of the grid. The second one needs to be written. If you construct a new `Game` and run your program a single image appears on the most left column of the grid: how is it different depending of the constructor used? Your game should always work under the two conditions, i.e. a client might create a game specifying the grid size or using the default: your code has to be robust under both circumstances.

   The lame images, "user.gif", "get.gif", and "avoid.gif", have been provided for testing. Eventually, you must replace them with images that are appropriate for your game's theme.

2. Look at the `play` method, which has mostly been implemented for you (you don't have to change it until step 9.): this method is the game's engine. Every `pauseTime` (100 milliseconds by default), it handles a key press (if any), if appropriate (i.e. sufficient time as elapsed) images (As and Gs) should scroll to the left and new images (again a combination of As and Gs) should populate the grid column closest to the right edge of the window, the title of the window should be updated to show the current score, and `msElapsed` is incremented, repeating until the game is over.

   In this assignment you implement the methods called by play. Start by completing `handleKeyPress()`. The following line of code checks for a key press:

   ```
   int key = grid.checkLastKeyPressed();
   ```

Right now the provided code assures that when the user presses `q` the game window closes.

Write code so that if the user pressed the up arrow `(key == KeyEvent.VK_UP)`, the user image (U) moves up one row (unless it is already in the top row). Likewise, when the user pressed the down arrow `(key == KeyEvent.VK_DOWN)`, the user image needs to move down one row (unless the user is already in the bottom row). Either way, make sure to correctly update the value of `userRow`.

Test your code by compiling and running **your game**:

```
Game.test();
```

does it by constructing a new `Game` object (make sure it isn't MattGame) and calls its `play()` method. You should be able to move the user image up and down without it moving off the screen (or causing an exception).

3. Complete the `populateRightEdge` method, which should randomly place in the rightmost column of the grid images that you want the user to get, such as "get.gif", and images that you want the user to avoid, such as "avoid.gif". Of course, you get to choose what "randomly" means here, using `Math.random()` or a `Random` object. Perhaps `populateRightEdge` should sometimes place nothing, sometimes place several images. But most often a single image should appear. (If you're unsure, just choose some arbitrary rule for now, and tune it later.)

   Note that your placement of items on the right edge of the screen should never put players in a scenario, which can't be won: the player should always have some path through the game without colliding with an "avoid" object. Thus, never make a column filled with "avoid" objects. Furthermore, patterns of successive columns should always allow the player to pass through without hitting an "avoid".

   Test your code with lines similar to the following:

   ```
   Game g = new Game();
   g.populateRightEdge();
   ```

4. Complete the `scrollLeft` method, which moves G and A images one column to the left. For example, if before the call to `scrollLeft` the grid is as on the left screenshot then after the call it is as on the right:

   Note that the G that was in the leftmost column has disappeared, and that an A has moved into the leftmost column. Note that the rightmost column is always empty after `scrollLeft` is called. Finally, note that the user-controlled U does not move. For now, assume that there is no image immediately to the right of the user, so you need not to worry about the possibility of a collision in this section.

   Hint: You might find it easier to write a helper method that scrolls one row.

   Be sure to test your code by running the game.

5. Complete the `handleCollision` method, which is called whenever the user might collide with an image at the given location. Check if the image is `null`. If so, no collision has occurred, and there's nothing for you to do. If the image is something you want to get (such as "get.gif"), then increment `timesGet`. If the image is something you want to avoid (such as "avoid.gif"), then increment `timesAvoid`. Either way, remove the image at that location by setting it to `null`.

6. There are two ways that the user can collide with an image:

   - the image may scroll left into the user, or
   - the user may move up or down into an image.

   Thus, `handleCollision` is called in both scenarios

   - in the `scrollLeft` method, before doing any scrolling (Hint: If the user is about to get hit by something, in what location would that something be, immediately before scrolling?)
   - in the `handleKeyPress` method, passing the location that the user is about to move into to `handleCollision` (Hint: Don't place the user's image in the new location until after you've called `handleCollision`.)

   Test your code by playing your game, making sure that it behaves reasonably.

7. The fields `timesGet` and `timesAvoid` influence either the score or the game-over condition, or both. For example, getting something good might increase the score, hitting something bad decrease the score, and the game ends after a set amount of time. Or alternatively, maybe getting something good increases the score, and hitting enough bad things ends the game. Alternatively, the score might depend on how long the user stays alive, and the user dies when they have hit more bad things than good things. There are many possibilities here.

   Complete the `getScore` method, which should return the current game score. In determining the score, you might use `timesGet`, `timesAvoid`, and `msElapsed`.

   Then modify `updateTitle` to show the title of your game. You might show other statistics in the title, in addition to the score.

   Test that the score now updates correctly, as you play your game and collide with images.

8. Complete the `isGameOver` method, which should return true at the end of the game. This may be a function of `timesGet`, `timesAvoid`, and `msElapsed`. Be sure to test that your game ends at the correct moment.

9. To help you tune the game add more branches to the `handleKeyPress` method code. In particular, add the following two keys to control the game speed by modifying the `pauseTime` value.

   - When the user presses comma (',') pauseTime increased and the game slows down.
   - Similarly period ('.') accelerates the game.
   - Make sure not to go overboard, i.e. a negative value is inappropriate. Check the demo to observe the behavior of those key presses.

   Add a pause feature activated by the key press ('p'). It should toggle between the game pausing (no scrolling happening) and the game restarting. You have to slightly modify the `play` method to provide this functionality.

Improvement

Now you are ready to work on the theme of your game, which requires a title and rules, images, etc. consistent with it: it is worth 30% of your grade.

In particular, do at least the following.

- Tune your game to make it fun to play: for example the timing should make the game playable, which means both winnable and challenging. (Other than the timing that affect the playability of the game, include the size of the images, the contrast between images and the background, which you may want to consider.)

- Enhance the appearance of your game. It is important to have a theme that drives your design. Choose a story and create or find `gif` images that support it. Simple and abstract design often pays off. (Use transparency in your `gif` images.)
- Implement a scoring mechanism with feedback in the title bar.

Name your `gif` files with **`yourFirstNameLastName`** appended to the filename so they can easily be shared without overwriting other people images.

If you have ideas that our suggestions did not cover, feel free to come talk to me or Matt. It would be nice if the user could move a few cells horizontally to allow more challenging patterns through the images.

Advanced software to manipulate images includes

- Gimp and Inkscape (free and open source) and
- Photoshop and Illustrator from Adobe

There are many tools online that are simpler to use. A quick search leads me to suggest

- YouIDraw which seems simple and powerful,
- Sketchpad seems interesting but may not handle transparency directly or
- any from that list could be a good idea.

Don't hesitate to use our Homework/Reading/Lab Forum to recommend software and ask any questions.

Submission

You have to submit two things.

- On Moodle submission folder: **upload a zip** named `hw4` that includes
  - `YourFirstNameLastNameGame.java` and possibly additional `.java`files for classes you implemented or modified (submit those only if you changed them)
  - your `.gif` files and
  - a `readme` file **in plain text** (`.txt` see instruction below) that describes
    a. your themes briefly
    b. the improvements you implemented to support it
    c. the software you used to make your images or their origin (website in particular)

    Don't forget **to credit your sources and collaborators**, i.e. anything you used or borrow and people who helped you. **It is the honor code.**
- On the dedicated Moodle forum called `Hw4 game release`: **post a a zip** containing
  - your `YourFirstNameLastNameGame.class` (ONLY **`.class`**) and
  - your specific **`.gif`** files, correctly named: don't forget to append your name to the image filenames

  The submitted `YourFirstNameLastNameGame.class` should have been compiled with those image names, located in the same directory than your program.

It is essential to properly follow these submission instructions, as penalty will apply for not respecting them.

Plain text

Word processing is one of the most frequent computer tasks, and most people have used Microsoft Word to process documents. But there are plenty of times when you don't want a formatted document nor the

proprietary Word format. Code is plain text and `readme` should also not use any formatting so to be easily read. Portability is optimal using ASCII and visual style isn't the point of a `readme`!: we care about the content only.

Every system has a variety of editors that let you create plain ASCII text with no formatting or extraneous content.

On **Windows**, you can use `Notepad++` or `Textpad`, or you can use `Wordpad` if you save your files as ".txt" format. You can usually find these programs from the Start menu in places like `All Programs | Accessories` or `All Programs | Basic Applications`. Do not use Word to edit plain text files.

On **Mac OSX**, you can use `TextEdit` (in `Applications`). Unfortunately `TextEdit` is sometimes too smart for its own good. The goal is to have no style toolbar in the window where you enter text. Use `Preferences | New Document` to select the radio button "Plain Text" instead of "Rich text" (.rtf). Under `Open and Save`, uncheck the box that says "Add '.txt' extension to plain text files" under **`When saving a file and select Unicode UTF-8 for Saving files under`** `Plain Text File Encoding`. Finally under the `Edit` menu of TextEdit toggle `Make Plain Text` (which is the opposite of `Make Rich Text`).

Alternatively install

- SublimeText: Windows, Mac and Linux available. SublimeText is a cross-platform text and source code editor which are editor to code (not any language specific). Open the application, write your readme and save it as `readme`
- TextWrangler: Mac only
- Notepad++: Windows

Credit This assignment is adapted from a Nifty assignment, which is supra cool.