Assignment 02: Scrolling Game  

Learning Outcomes

This homework is an application of *object-oriented programming*, the programming paradigm we are learning in COSC102. Using provided code that defines a scrolling game platform, you will design your own game, by choosing a narrative, creating the visual display and implementing its game play—controls and update. This game development project relies on an OOP organization of the code base, which separate features among many classes. Your task im- mersed in this complex code structure is to integrate together the components to create your own fully functional game.

Doing so you are learning the power of OOP based on the following principles

1. **abstraction**—generalising an objet, and its use in game development,

2. **cohesion**—being one thing or doing one thing well, which relies on grouping together code that contributes to a single task, and,

3. **encapsulation**—information hiding. The implementation (the internal workings) of an object is hidden from the rest of the program.

Inheritance and (de)coupling—separating concerns—which implies that an object doesn’t directly change or modify the state or behavior of another object, two additional OOP principles, are not explicitly used in this project. To find out more about these principles as they apply to game development read to the following blog posts written by Steven Lambert.  

http://blog.sklambert.com/introduction-to-oop-for-game-development/

1 Introduction

The above screenshot shows the provided demo version of the scrolling game in action. The user is controlling the U object and is able to move up and down while the other objects are moving towards the user (right to left).

Aside from the user, there are two different types of images:

**G’s** (which stand for *GET*) representing an object which the user wants to try to collect to increase their score; and

**A’s** (which stand for *AVOID*) representing an object which the user does not want to collide with as they can ultimately lead to some kind of failure state (i.e. a “Game Over”).
The window displays a game board based on an underlying grid. The placement and movement of all the objects (including the user, U) is tied to moving to different "cells" on this grid.

The location of any given object can be expressed as a coordinate on this grid, with (0, 0) representing the spot in the upper-left corner of the window. In your code, you will be able to dictate the dimensions of the grid (and thus the playing area of the game, press D to toggle the grid on the demo). The window size is irrelevant, the code that draws the images will scale them appropriately as the window is resized.

While playing the game, the user earns a score, shown in the title bar next to the name of the game. The game ends when some defined win or loss condition is reached. The definition of those conditions—and the rest of the game design—is up to you!

Some of the elements you will need to design include:

- the game’s theme/story including its title, splash-screens, cell images and anything else that determines the game’s theme and playability;
- the scoring and win condition;
- the failure states/game over conditions; and
- any additional features (rules and interactions) you add to the design to make the game more interesting.

You can be as creative 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 as creative as you may think.

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.**

## 2 Provided Code

The starting code for this assignment, a zip file on Moodle, consists of several Java source-code files (.java) and one Java bytecode file (DemoGame.class) an unthemed and incomplete demo version of the game showcasing some of the capabilities of the provided code. The images for the symbols U, G, and A are also provided. The only source-code file you should modify is ScrollingGame.java; however, you should examine the other files to learn how the game works.

### 2.1 Setup

1. Download ScrollingGame.zip from Moodle, which contains the files provided for this assignment.
2. Unzip them in a new folder hw2 in your 102 directory.
3. Open ScrollingGame.java: **all the code you write for this assignment will go in this file.**
4. Rename the file `ScrollingGame.java` to be `YourFirstNameLastNameGame.java` replacing with your actual name `YourFirstNameLastName` part.

5. Open `GameLauncher.java` to run the demo game. The class constant `boolean DEMO` permits to run the `DemoGame.class` provided, which is the basic game example you have to replicate first, or your own game implementation you are writing in the `ScrollingGame.java` file. The setup should remain you of the one used in the SilverCoin lab.

6. Compile (`javac *.java`) and run the program (`java GameLauncher`) after having changed the class constant and made some change on lines 39 and 42 in `GameLauncher.java` to refer to different filename you used for your class.

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

### 2.2 Files Structure

There are four source-code `.java` files that work together to implement the side-scroller game.

- **GameLauncher** contains the main method to start the `ScrollingGame` or the provided demo version.
- **ScrollingGame** contains the logic for the game itself, and uses methods from `GameGrid` to change the game’s graphical rendering.
- **GameGrid** does most of the work in maintaining and rendering the window/game board. Its state contains a two-dimensional array of Cells. The Cell object is defined by an inner class (see the bottom of that file) that represents the state of each given “spot” in the game grid, including what object (if any) is being drawn there.
- **Location** is an object signifying the coordinates of a given spot on the game grid. The upper-left spot is position (0,0).

At this point you may want to look through the files to see all the methods available in them. Many of the files are commented with explanations of what the various methods do.

### 2.3 ScrollingGame Description

The `ScrollingGame` has constants (check them) and five instance variables declared for now; the instance variables include

- **grid** is an object that represents the game’s graphical view—methods called on this object will display different things on the screen. The graphical view maintains the grid used to store and display images on the screen.
- **userRow** stores in which row the user is located on the left edge of the grid. Use it to keep of the user-controlled image (U).
- **msElapsed** stores the number of milliseconds since the game main loop started.

... and so on. You are free to change any of these variables and you should add others, as you are responsible for designing how to represent the information for an entire game.

The initialization happens in the `init` method of `ScrollingGame`. After the game object is initialized, the `play` method is called on it, which starts a gameloop.

Its rough structure is outlined in the provided code but much of the interesting functionality is missing. At a high level, here is how the main gameloop works:

- A while loop keeps iterating until the game is over (determined by the `isGameOver` method, which you will implement later).
- On each iteration of the loop, the game sleeps (as in, stops and waits) for a number of milliseconds determined by the variable `delayTime`. Without it, the game loop will be faster than the player can possibly react.
• After the sleep, both handleKeyPress() and handleMouseClick() methods are called to deal with any last keyboard press and mouse click the user has made since they were previously called.

• On every third iteration of the gameloop (as governed by the FACTOR variable) the game scrolls the grid. Specifically,
  – all the non-player objects (i.e. not the User) scroll to the left and
  – the rightmost edge of the grid is populated with some sampling of new objects (A’s and G’s).

• Finally, the game updates the title bar at the top of the window using the updateTitle method.

You will need to implement all the methods called by play, although you are free to add and change the specific method calls as long as the basic structure of the gameloop is maintained.

There is already code to end the game when the q key is pressed. Unlike console programs, this is an interactive graphics program. It doesn’t stop and wait for user input; it continues to run and it polls for messages from the keyboard and the mouse. The handleKeyPress method checks for a key press and retrieves it as an int value using the line

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

Your completed implementation of the game will check for other key presses, as indicated in the specification below and similarly you can retrieve location of mouse clicks.

### 3 Basic Game Features

At a minimum, your game must support the following features:

• It must scroll the game and populate the right edge of the game with at least two different types of objects: one representing a good object that the user wants to get (a G), and one representing a bad object that the user wants to avoid (an A).

• It must update its state (either by changing a score or some other player feature) when the player runs into the different types of objects.

• It must respond to several additional key presses:
  up, down keys are used to move the player to a different row (without moving the user off the top or bottom of the grid or generating an exception/error);
  right, left keys are used to move the player to a different column (without moving the user off the screen grid or generating an exception/error);
  P is used to pause and resume the game;
  S is used to take a screenshot of the games screen;
  D is used to turned on/off grid lines;
  comma, period are used to slow down and speed up the animation of the game.

### Suggested Plan for Implementation

Here is one way to incrementally add and test the basic functionality for your game:

1. Start with the constructor(s) and the init method. What should you see when running your game given the code executing? Add lines if necessary, try different constructors to understand the flexibility provided to the client (i.e. main and run static methods). You should be able to have the user image (U) appears.

The lame images, “user.gif”, “get.gif”, and “avoid.gif”, have been provided for testing. Use constants to facilitate their replacement with images that are appropriate for your game’s theme later on.
2. Write code that moves the player up and down. Change its corresponding state in `ScrollingGame` by calling `setCellImage` on `grid` to change the image displayed in the correct game cells. (Use null to clear an image from a position.) There is an example of `setCellImage` in the initialization of `ScrollingGame`.

The goal is that when the player 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 player pressed the down arrow. Either way, make sure to correctly update the value of `userRow`.

Test this feature by compiling and running your game every 5 lines you write!! Find a way to check your progress, `println` is useful.

3. Complete the `populateRightEdge` method, which should randomly place in the rightmost column of the game the images for the objects you want the user to get (“get.gif” for now), and those they should avoid (“avoid.gif”). Of course, you get to choose what “randomly” means here; so you should come up with some algorithm that places the objects making the game both engaging and challenging. Perhaps `populateRightEdge` should sometimes place nothing, sometimes place several images. (If you’re unsure, just choose some arbitrary rule for now, and tune it later.)

Don’t forget to use our `Random` class variable `rand`. (You should not instantiate a new object each time you want a random number.) could sometimes place nothing and sometimes place several images.

Note that your placement of objects on the right edge of the screen should never put players in an unwinnable scenario! The player should always have some path through the game board without colliding with an “avoid” object. Therefore, you should never generate a column that is completely filled with “avoid” objects. Patterns of previous columns should always have at least one path the player can pass through without hitting an “avoid” object.

Test and debug!!

4. Write code to scroll the game to the left one column at a time. For example, if on the next page the image on the left represents the state of the grid before `scrollLeft` is called, than the screenshot on the right would be the board after the method is called. In this example, the G that was in the leftmost column has disappeared, and an A has moved into the leftmost column. The rightmost column is empty, waiting for placement of new objects. Finally, the user-controlled U does not move (that only happens in response to key presses, which is handled separately).

**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. Add some code to detect that, either because of user movement or because of scrolling, a collision between the user and an object in the game grid has occurred. In this case, the graphics should be updated to only show the user image, not the image of the object “collected” by the user. You may decide that colliding with different types of objects will have different effects on the game, which can provide for interesting customizations for your assignment; in these cases, consider what game state and graphics state will need to be updated.

You may add code to the `handleCollision` definition to keep your implementation organized.

6. Add some code that tracks positive and negative object collisions, for example, updating a score or other player features. This will probably be tied to the condition that the game ends in a win or loss (for example, the player might win after collecting a certain number of G’s). Once you have established these rules, make sure to
implement the `isGameOver` method so that the animation can stop once the game has finished. Also, add some code to reflect useful state information (like score, etc.) to the player in the title bar of the game window (use the `setTitle` instance method of the `GameGrid` object).

7. Write code that moves the player right and left now. It is another chance to see if collision is working as intended.

8. Write code to handle key presses for pausing (a toggle), saving the screen, turning the grid lines (toggle) speeding up, and slowing down the game animation.

   To implement pause you do not want to stop or block the game loop: stopping the game animation and the player movements will suffice to freeze the game; there is one particular event that you need to listen for.

   To change the game speed, remember that the loop driving the game in the `play` method should still be scrolling the game on every third, i.e. `FACTOR`, iteration. Changing the game speed simply means that the delays are longer or shorter.

### 4 Game Design: Improvements

Now you are ready to work on the theme of your game, which must include a game title, introduction splashscreen (best if rules/keys are explained), corresponding images, a scoring mechanism and a final splashscreen given the outcome of the play. Your game design is worth 22% of your grade; your final code design (described below) is worth 8%.

There are many components of the game that you can customize. The creative part of the assignment involves going beyond the demo version of the game to incorporate a unique theme and graphical style as well as some new rules/game mechanics to make the gameplay more interesting. In particular, you must do the following:

- Tune your game to make it fun to play. The speed and difficulty should make the game playable, which means both winnable and challenging. Other than timing, you may also want to consider the distribution of “get” and “avoid” objects, the contrast between images and the background, the rules for winning/losing the game, etc. Your game should have some kind of definitive win and lose state(s) (i.e. your game should be “beatable”).

- Enhance the appearance of your game. It is important to have a theme that drives your design. Choose a story and find or create images that support it, including splashscreens. Simple and abstract design often pays off. For the best look, use transparency in your images. You may use any images you find as long as you cite them in your `readme` file (more on this below).

- Implement one or more unique game mechanics. In other words, add a new wrinkle to the existing game’s design via a new rule or feature that impacts the way the game is played. Some examples include new powerups, different types of objects, different win/lose conditions, new abilities for the player, etc. For full credit, the new features you add to your game should be creative and interesting.

**Splashscreen hint:** To implement a static screen you should use a similar `while` loop than the one for the game loop (without most of the gameplay update). The `grid.pause(pauseTime);` is required to slow the refresh rate, such that user events can get picked up. (For the pause feature of the basic game you should not use this `pause` method of the `Grid` object, but implement the feature with your own logic—a boolean suffices.)

If you have design ideas that you are unsure about, feel free to ask us. You are free to use your creativity here, but remember that you should not modify any of the provided classes except for `ScrollingGame`. (If you have been granted the permission to change the other classes, that’s fine. You may if you have a good reason.)

The image type that works best with this game is a GIF file. Some advanced software you may want to look into for editing images include:

- **Gimp** [https://www.gimp.org/downloads/](https://www.gimp.org/downloads/)
- **Inkscape** [https://inkscape.org/en/download/](https://inkscape.org/en/download/)
- **YouIDraw** [http://site.youidraw.com/](http://site.youidraw.com/), which seems simple and powerful
Sketchpad [https://sketch.io/sketchpad/](https://sketch.io/sketchpad/), which seems interesting but may not handle transparency directly.

There are many free tools online that you may find simple to use, and there are several free icon and image galleries from which you can obtain images. Don’t hesitate to talk to us and classmates if you have difficulties. I encourage you to recommend software to your peers and solve image software problems collaboratively.

5 Code Design

You are required to do the following:

- Provide a plain text file, readme (as in A1) that outlines your theme, customizations, and overall vision for the game. Explain the design for the implementation of your extra feature(s): the methods and algorithms modifying data state (variables, maybe structure(s) or class(es)). Indicate the implementation you are the most proud of.

- Document the variables and methods in your code. This is a complex program, and it should be easy for a reader to understand where different pieces of important information are located, and where important functionality is implemented. You don’t have to comment an index variable in a for loop, but certainly document the purpose of things in your class definition using high-level language.

- Your code should be of a quality that it is easy to understand. For example, if in 20 seconds you can explain the purpose of each method to a classmate such that are assured of the validity of your approach considering the small amount of lines of code, that’s great. If you are unsure, it probably means you need to redesign that part of the code.

6 Submission

The submission will include:

- Final source code for your game. Submit all the .java files, including YourNameScrollingGame.java, code for any helper classes, and the .java for the provided code.

- Image files required to implement your theme.

- Two screen captures you took with the 'S' key to showcase your game features.

- A readme file

Submit your assignment by creating a ZIP archive of these items and uploading the archive to Moodle. The above items are due **Friday October 19th, 2018 at 5:00PM**.

Credit  This assignment is adapted from a supra cool Nifty Assignment designed by Dave Feinberg (see [http://nifty.stanford.edu/2011/feinberg-generic-scrolling-game/assignment.html](http://nifty.stanford.edu/2011/feinberg-generic-scrolling-game/assignment.html)).