Lab Exercise 11: Interactive Graphics
CS 2334

November 3, 2016

Introduction

In this lab, you will extend your knowledge of creating graphics in Java. Specifically, you will experiment with using **KeyListeners** and **KeyEvents** to construct graphics programs that react to keyboard button presses that are made by a user.

The game that you are completing requires the player to move through a shifting maze by pressing the right and left keys arrow keys. The player must stay within the unoccupied area of the screen. If the player is caught by a moving wall, then the player loses the game. If the player makes it to the end and collects the Poke Ball, the player wins.

Learning Objectives

By the end of this laboratory exercise, you should be able to:

1. Create event-driven graphics
2. Use a **KeyListener** to update graphics based on **KeyEvents**
3. Read existing code and documentation in order to complete an implementation

Proper Academic Conduct

This lab is to be done individually. Do not look at or discuss solutions with anyone other than the instructor or the TAs. Do not copy or look at specific solutions from the net.

Preparation

1. Import the existing lab11 implementation into your eclipse workspace.

   (a) Download the lab11 implementation:
   `http://www.cs.ou.edu/~fagg/classes/cs2334/labs/lab11/lab11.zip`

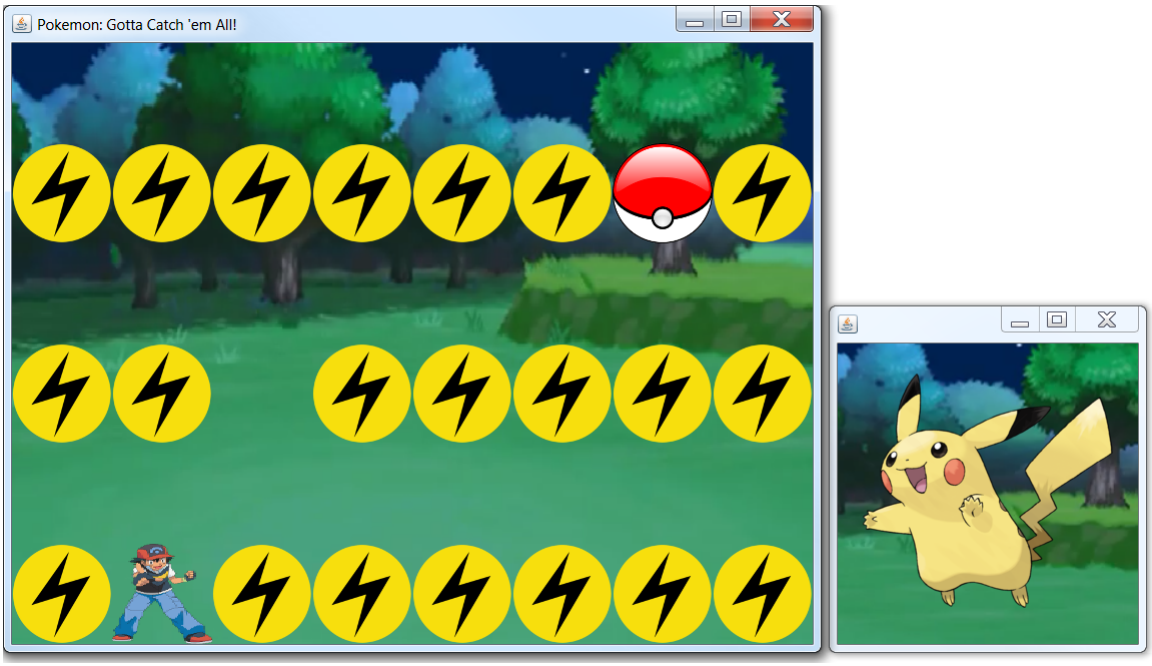
   (b) In Eclipse, select *File/Import*

   (c) Select *General/Existing projects into workspace*. Click *Next*

   (d) Select *Select archive file*. Browse to the lab11-initial.zip file. Click *Finish*

Game: Pokmon: Gotta Catch 'em All

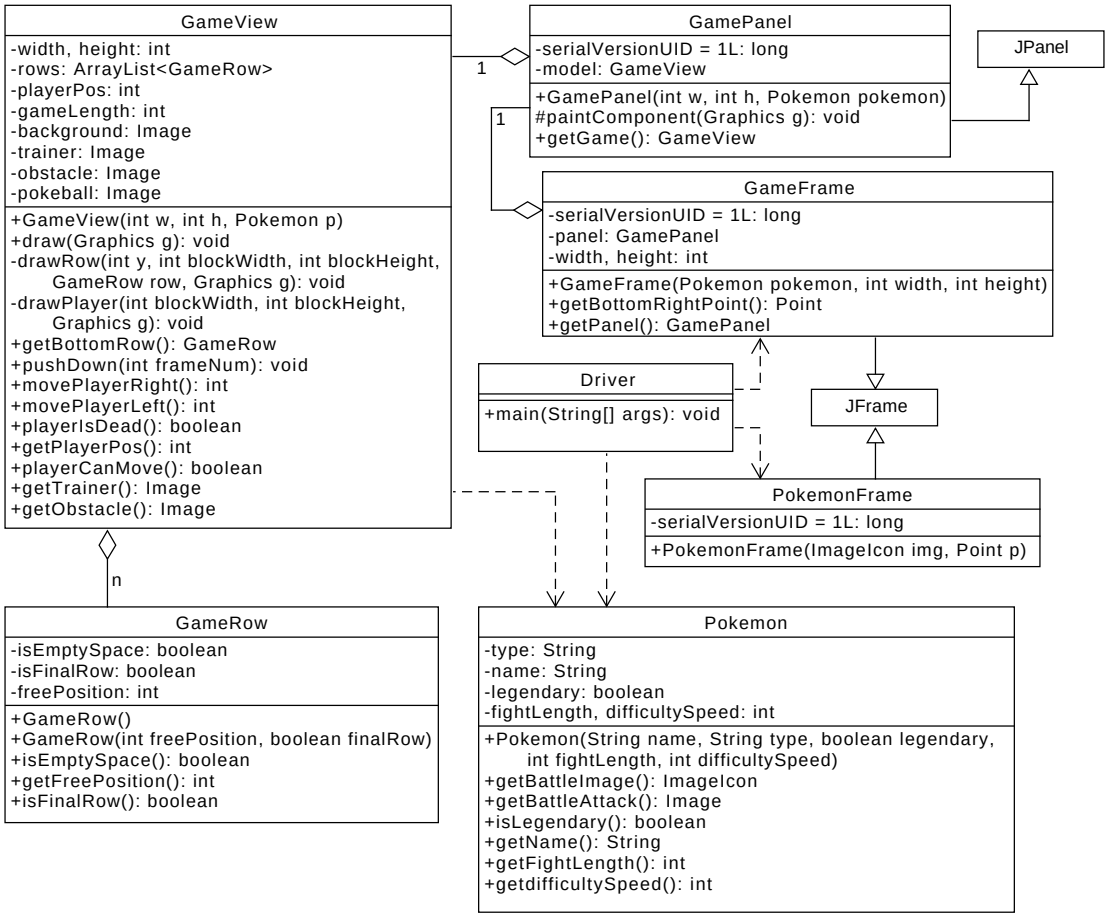
Below is an image of the graphical user interface for the game that we are creating.



Ash Ketchum (the person in the red hat and blue pants) represents the player. The rows of electric symbols are walls through which the player cannot pass. These rows shift downwards at decreasing intervals. The goal is to move the player into a free position before the wall overtakes the player, and collect the Poke Ball at the end of the maze. This will be done by moving the player right or left using the right and left keys on the keyboard.

Note: the player can wrap around the window to get to the other side.

UML



- **Driver.** This class houses the JOptionPane that the user interacts with to select their difficulty level. The Driver then creates the appropriate Pokemon object, PokemonFrame, and GameFrame. The game then runs to completion inside a loop, and exits with a final JOptionPane display.

- **GamePanel.** A simple panel that is used inside the GameFrame to display the GameView object.

- **GameView.** The meat of this project. Here, multiple components come together to produce the Pokemon game. This class is responsible for creating

every component related to the maze, player, and movement. Here is where you will finish implementing *movePlayerRight* and *movePlayerLeft*, as well as *drawPlayer*, *drawRow*, and *draw*.

- **GameRow.** The class representation of a single row of the in-game maze. This row can either be completely empty, or be completely filled with obstacles save for a single empty space the player can safely pass through.
- **Pokemon.** A class that represents a catchable Pokemon for our game. Also handles level dynamics such as game length, difficulty, and graphical resource preparation for each Pokemon.
- **PokemonFrame.** A simple frame that displays the Pokemon you are trying to catch. This frame is spawned at a location dependent on the GameFrame location.
- **GameFrame.** The main frame that holds the gamePanel and, therefore, the entire game. This frame is responsible for handling key inputs from the player and moving the player on screen accordingly. You will need to implement a *KeyListener* to accomplish this task.

Lab 11: Specific Instructions

All of the classes shown in the UML are provided in the initial lab11.zip.

1. Most of the classes are implemented. We want you to implement the graphics, not the logic of the game. However, you need to analyze and understand the game logic to implement the graphics.

    - **Driver**, **GamePanel**, **GameRow**, **Pokemon**, and **PokemonFrame** have been fully implemented. Read and understand these classes before moving on.
    - Implement a **KeyListener** in **GameFrame**
        - When you implement the **KeyListener**, Java will require you to create handler methods for three events types. Since you only need one event type, it is okay to leave the other two methods with no body.
        - Alternatively, you may implement a **KeyAdapter**, for which you only need to override the one method of interest.

- Complete the implementation of **GameView**
  - Most of the logic occurs in this class. Fully analyze and understand the code before moving on.

2. Do not add functionality to the classes beyond what has been specified

3. Don't forget to document as you go!

4. You do not need to implement JUnit tests for this lab. Instead, you should be testing your code interactively. Make sure to try all of the possible conditions to make sure that they player movement and game rules are implemented correctly.

   Note that Web-Cat will be executing a unit test on your code. In most cases, we expect this unit test to pass. However, there will be an additional step of checking that your program is producing the correct output images during game play.

Final Steps

1. Generate Javadoc using Eclipse.

   - Select *Project/Generate Javadoc...*
   - Make sure that your project is selected, as well as all of the Java source files
   - Select *Private* visibility
   - Use the default destination folder
   - Click *Finish*

2. Open the *lab11/doc/index.html* file using your favorite web browser or Eclipse (double clicking in the package explorer will open the web page). Check to make sure that that all of your classes are listed and that all of your documented methods have the necessary documentation.

3. If you complete the above instructions during lab, you may have your implementation checked by one of the TAs.

Submission Instructions

- All required components (source code and compiled documentation) are due at 11:59pm on Friday, November 4th.
- Method 1: Submit through Eclipse
  1. From the *Window* menu, select *Preferences/Configured Assignment*.
  2. Select your project.
  3. From the Project menu, select *Submit Assignment*.
  4. Under *Select the assignment to submit*, select *Lab 11: KeyEvents, KeyListeners, and Graphics*.
  5. Click *Change Username or Password....* Enter your Web-Cat username and password. Click *OK*. You should only need to do this step once per session.
  6. Click *Finish*.
  7. Your browser should automatically open a Web-Cat page that shows your submission being graded. After a short wait, the page will show a report of your submission. See the main class web page for a link that describes the Web-Cat output.
- Method 2: Submit directly to the Web-Cat server
  1. From the File menu, select *Export*.
  2. Select *Java/JAR File*. Click *Next*.
  3. Select and expand your project folder.
  4. Select your *src* and *doc* folders.
  5. Select *Export Java source files and resources*.
  6. Select an export destination location (e.g., your *Documents* folder/directory). This file should end in *.jar*
  7. Select *Add directory entries*.
  8. Click *Finish*.
  9. In your web browser, login to the Web-Cat2 server.
  10. Click the *Submit* button.
  11. Browse to your jar file.

12. Click the *Upload Submission* button.
13. The next page will give you a list of all files that you are uploading. If you selected the correct jar file, then click the *Confirm* button.
14. Your browser will then open a Web-Cat page that shows your submission being graded. After a short wait, the page will show a report of your submission. See the main class web page for a link that describes the Web-Cat output.

Rubric

The project will be graded out of 100 points. The distribution is as follows:

Correctness/Testing: 45 points

The Web-Cat server will grade this automatically upon submission. Your code will be compiled against a set of tests (called *Unit Tests*). **NOTE:** The tests used for this lab will focus on graphics alone - a style of testing different than what we have done before. These unit tests will not be visible to you, but the Web-Cat server will inform you as to which tests your code passed/failed. You do not need to write tests for this lab. This grade component is proportional to the fraction of tests that your code passes (so 22.5 points means that your code passed half of the tests)

Style/Coding: 20 points

The Web-Cat server will grade this automatically upon submission. Every violation of the *Program Formatting* standard described in Lab 1 will result in a subtraction of a small number of points (usually two points). Looking at your submission report on the Web-Cat server, you will be able to see a notation for each violation that describes the nature of the problem and the number of subtracted points.

Design/Readability: 35 points

This element will be assessed by a grader (typically sometime after the lab deadline). Any *errors* in your program will be noted in the code stored on the Web-Cat server, and two points will be deducted for each. Possible errors include:

- Non-descriptive or inappropriate project- or method-level documentation (up to 10 points)
- Missing or inappropriate inline documentation (2 points per violation; up to 10 points)
- Inappropriate choice of variable or method names (2 points per violation; up to 10 points)
- Inefficient implementation of an algorithm (minor errors: 2 points each; up to 10 points)
- Incorrect implementation of an algorithm (minor errors: 2 points each; up to 10 points)

- Non-visible components (2 points per violation; up to 10 points)

If you do not submit compiled Javadoc for your lab, 5 points will be deducted from this part of your score.

Note that the grader may also give *warnings* or other feedback. Although no points will be deducted, the issues should be addressed in future submissions (where points may be deducted).

Bonus: up to 5 points

You will earn one bonus point for every two hours that your assignment is submitted early.

Penalties: up to 100 points

You will lose ten points for every minute that your assignment is submitted late.