Lab Exercise 11
CS 2334

November 2, 2015

Due: 7:00 pm on Saturday, November 4, 2017

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 use the arrow keys to move through a set of obstacles and capture a villain. The player must stay within the unoccupied area of the screen. If the player cannot catch the villain before time is up, then the game will end in failure.

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`

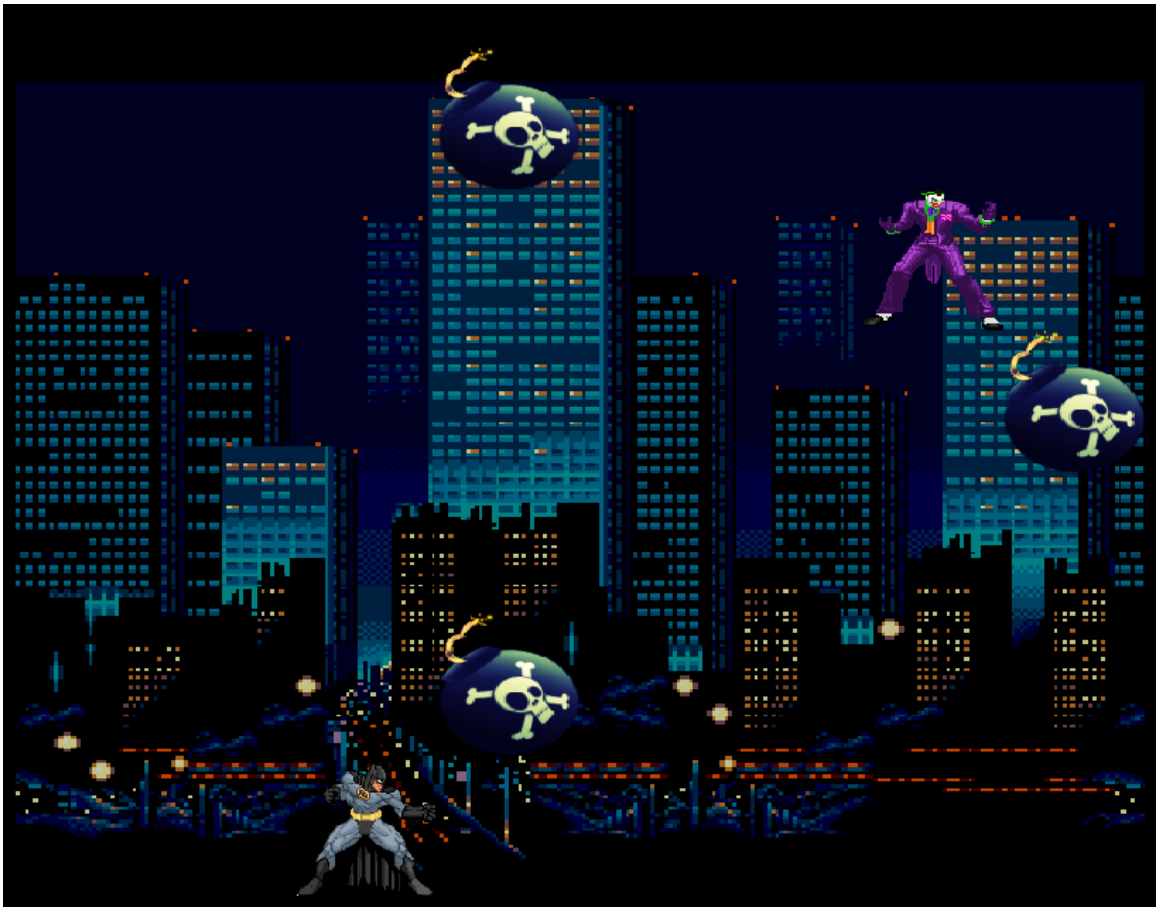
   (b) In Eclipse, select *File/Import*

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

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

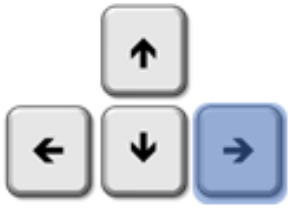
Game: Super Showdown

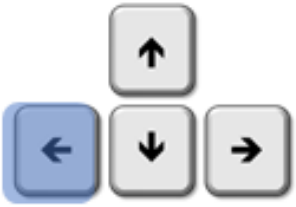
Below is an image of the graphical user interface for the game that we are creating.



The Batman image is one of four avatars that the player can use. The bombs are obstacles that the player must move around. These bombs appear in random locations at the start of each game. The villain will move randomly throughout the game, and the hero must catch him before time runs out. This is done by moving the player right, left, up, or down using the corresponding arrow keys on the keyboard.

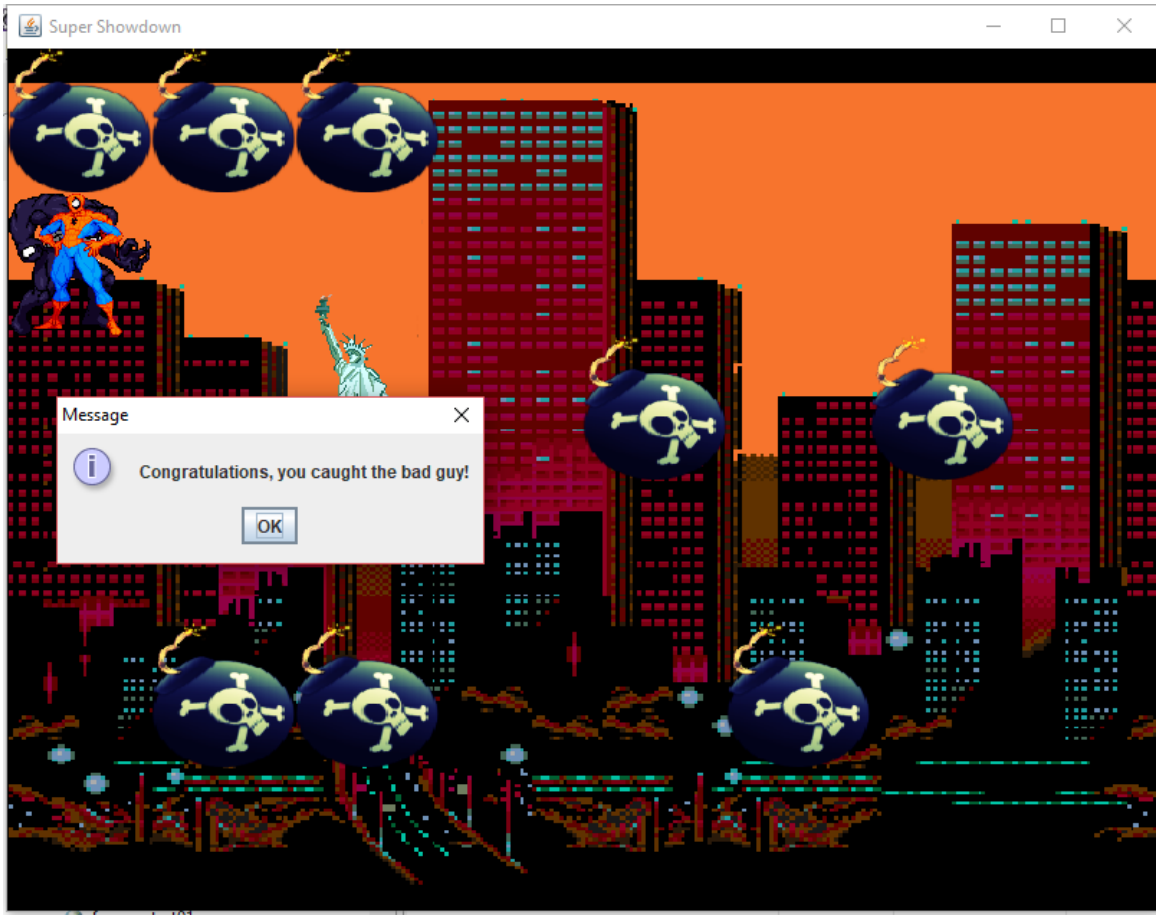
The player (and villain) can wrap around the window to get to the other side, when going left or right. Wrapping **does not** happen when going up or down. The modulus operator (i.e. %) can be used to conveniently make circular/wrapping operations. See the code for the villain movements for examples. Below is an example of wrapping:



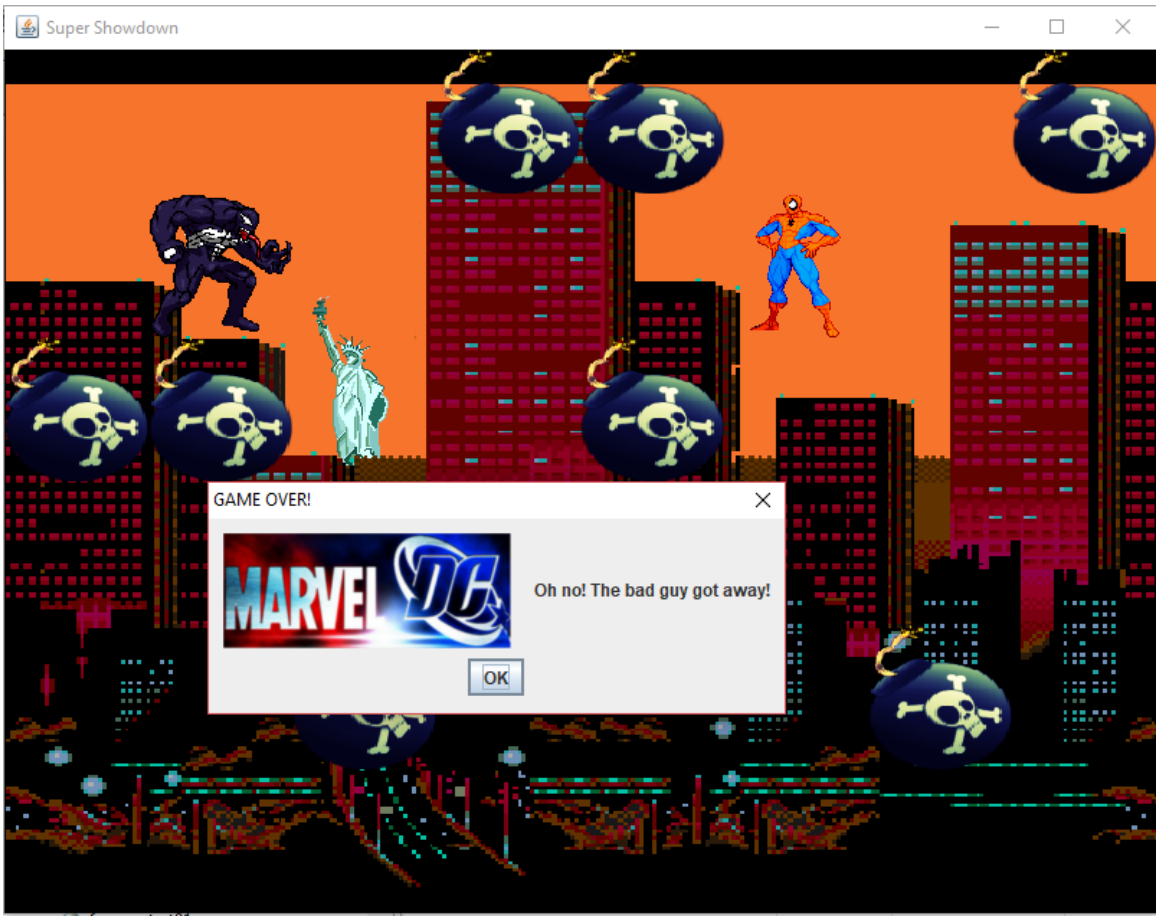


The player may never occupy the same space as an obstacle. If a player attempts to move onto an obstacle, the position of the avatar will not be changed. As a result, the avatar will never be drawn on top of a bomb icon.

When catching the villain, it is okay for the player to be drawn over the same tile as the villain.

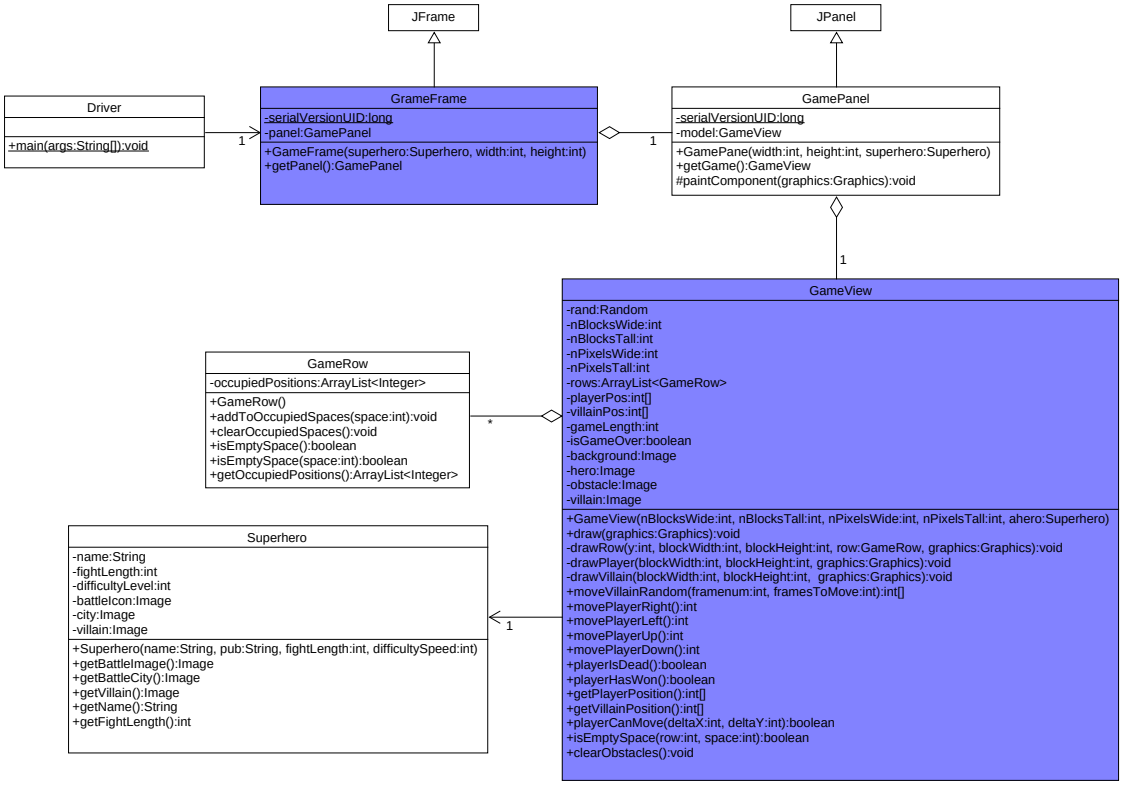


If the player takes too long to catch the villain, the villain gets away and the player loses the game.



UML

Here is the design of your classes. You are responsible for completing the implementations of the classes highlighted in blue.



Lab 11: Specific Instructions

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

1. Most of the classes are implemented. We want you to implement the graphics, not the logic of the game, by finishing the implementation for the classes in blue in the UML. However, you need to analyze and understand the game logic to implement the graphics accordingly.

    - **Driver**, **GamePanel**, **Superhero** and **GameRow** have been fully implemented. Read and understand these classes before moving on. The

GameRow class keeps track of the obstacles (i.e., the bombs) within the frame using an ArrayList of indices. The row location (an integer) of each obstacle is stored in this array.

- Complete the implementation of the **GameFrame** by adding a **KeyListener** to the constructor.
  - 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.
  - You will need to use the *repaint()* method **of the frame** to redraw the player after moving, to force the frame to update itself.
- Complete the implementation of **GameView**
  - Most of the logic occurs in this class. Fully analyze and understand the code before moving on.
  - You will need to complete the draw methods for the villain and the player, and the movement methods for the player. Additionally, complete the *playerCanMove(int,int)* method.
  - Read the comments and follow the TODOs to help figure out what needs to be implemented.

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

3. You will not need JUnit tests for this lab. You will need to test your code manually through interaction with the GUI. Nonetheless, JUnit tests will be waiting on Web-Cat to ensure that your code is performing as expected.

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 7:00pm on Saturday, November 4th. You will not need JUnit tests for this lab. You will need to test your code manually through interaction with the GUI.

- Submit your code to *Lab 11: Interactive Graphics* on WebCat using 1 of the 2 methods described from lab 1.

Rubric

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

Correctness/Testing: 40 points

The Web-Cat server will grade this automatically upon submission. Your code will be compiled against a set of tests (called *Unit Tests*). 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. This grade component is a product of the fraction of **our tests** that your code passes and the fraction of **your code** that is covered by *your tests*. In other words, your submission must perform well on both metrics in order to receive a reasonable grade.

Style/Coding: 25 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)

If you do not submit compiled Javadoc for your project, 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. For a submission to be considered *on time*, it must arrive at the server by the designated minute (and zero seconds). For a deadline of 9:00, a submission that arrives at 9:00:01 is considered late (in this context, it is one minute late).

After 15 submissions to Web-Cat, you will be penalized one point for every additional submission.

For labs, the server will continue to accept submissions for three days after the deadline. In these cases, you will still have the benefit of the automatic feedback. However, beyond ten minutes late, you will receive a score of zero.

The grader will make their best effort to select the submission that yields the highest score.