THE UNIVERSITY OF MELBOURNE
SCHOOL OF COMPUTING AND INFORMATION SYSTEMS
SWEN20003 OBJECT ORIENTED SOFTWARE DEVELOPMENT

Project 1, Semester 2, 2019

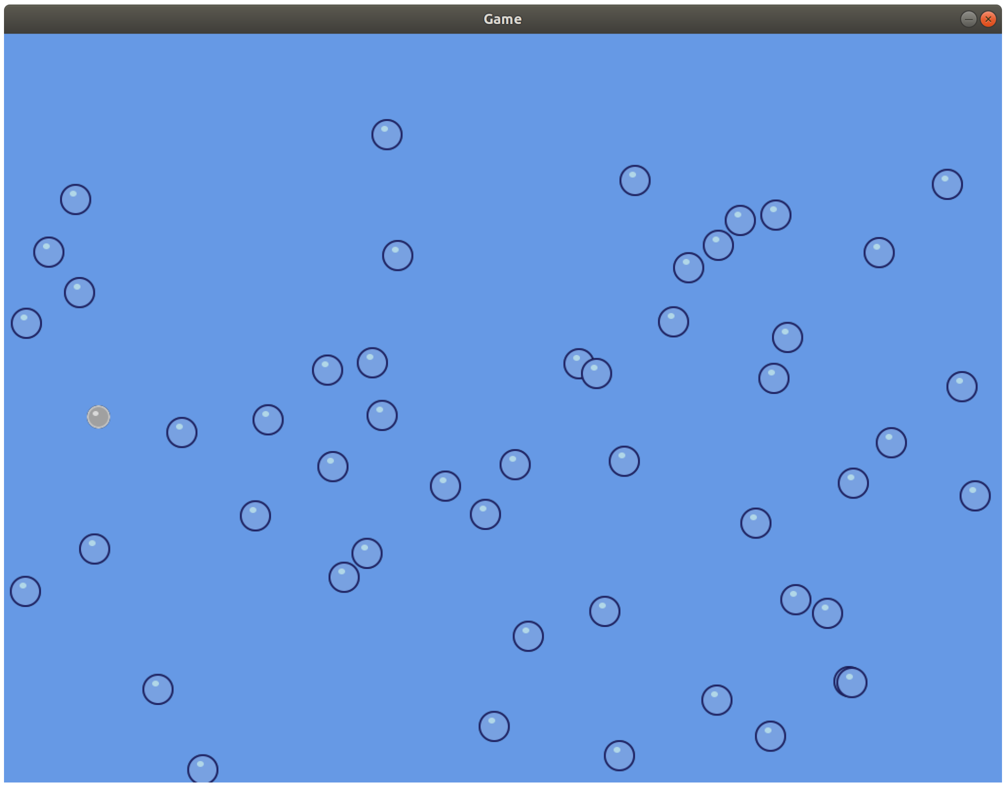
Written by Eleanor McMurtry

Released: Friday 30th of August, 9:00pm
Due: Friday 13th of September, 11:59pm

Overview

Welcome to the first project for SWEN20003! We will be using the Bagel library for Java, available under Projects on the LMS. Comprehensive documentation for Bagel can be found at `https://people.eng.unimelb.edu.au/mcmurtrye/bagel-doc/`. Week 5's workshop introduces Bagel, so refer to the tutorial sheet if you have trouble getting started. This is an **individual** project. You may discuss aspects of it with other students, but all of the implementation must be your own work.

Project 2 will extend and build on Project 1 to create a complete game, so it's important to write your submission in such a way that it can be easily extended. Below is a screenshot of the game after completing Project 1.



This early version of **Shadow Bounce** includes several "pegs" and a ball that can be used to hit and destroy the pegs. The goal of **Shadow Bounce** is to use the ball to destroy all of the pegs in as few shots as possible.

Background on graphics concepts

Every coordinate on the screen is described by an (x, y) pair. $(0, 0)$ represents the top-left of the screen, and coordinates increase towards the bottom-right. Each of these coordinates is called a *pixel*. The Bagel `Point` class encapsulates this, and additionally allows floating-point positions to be represented.

60 times per second, the program's logic is updated, the screen is cleared to a blank state, and all of the graphics are drawn again. Each of these steps is called a **frame**.

Often, we would like to draw *moving* objects. These are represented by a **velocity**: a pair (also called a **vector**) (v_x, v_y) that represents how far the object should move each frame. Calculating the new position can be done via *vector addition* of the position and velocity; Bagel contains the `Vector2` class to facilitate this. You are not required to use this class; it is merely provided for convenience. The **magnitude** (or **length**) of a vector (x, y) can be found via the formula $\sqrt{x^2 + y^2}$.

It is sometimes useful to be able to tell when two images are *overlapping*. This is called **collision detection** and can get quite complex. For this game, you can assume images are rectangles. Bagel contains the `Rectangle` class to help you.

The game

Below I will outline the different game elements you need to implement.

The pegs

The game board is made up of a collection of pegs that appear on the screen. For this project, they will be randomly generated. The game should choose 50 random positions on the screen with x coordinates between 0 and 1024, and y coordinates between 100 and 768. Pegs should be drawn centred at these locations.[1]

The ball

When the left mouse button is clicked, if the ball is not currently on screen, the ball should appear at the position $(512, 32)$. It should start with a velocity towards the mouse, with a magnitude of 10 pixels per frame. Each frame, the velocity should increase by 0.15 pixels per frame downwards, to simulate gravity. When the ball's centre reaches the left or right side of the window, the ball's horizontal movement should reverse so that it stays on screen. If the ball makes contact with a peg, the peg should be "destroyed" and no longer displayed on screen.

Your code

You must submit a class called `ShadowBounce` with a `main` method that runs the game as described above. You may choose to create as many additional classes as you see fit, keeping in mind the principles of object oriented design discussed so far in the subject. You will be assessed based on your code running

[1] It would be nice if these positions are chosen so the pegs don't overlap, but this won't be assessed.

correctly, as well as the effective use of Java concepts. As always in software engineering, appropriate comments and variable/method/class names are important.

Implementation checklist

To get you started, here is a checklist of the game features, with a suggested order for implementing them:

1. Draw a single peg on screen
2. Generate random positions and draw a peg at each of them
3. Draw a ball on screen when the left mouse button is pressed
4. Make the ball fall downwards over time
5. Detect when the ball touches pegs, and destroy the pegs
6. Make the ball start off moving towards the mouse
7. Make the ball "bounce" off the sides of the screen

The supplied package

You will be given a package, `res.zip`, which contains all of the graphics and other files you need to build the game. You can use these in any way you want. Here is a brief summary of its contents:

- `res/` – The images for the game.
  - `ball.png`: the image for the ball object
  - `peg.png`: the image for the peg objects

Submission and marking

Technical requirements

- The program must be written in the Java programming language.
- The program must not depend upon any libraries other than the Java standard library and the Bagel library (as well as Bagel's dependencies).
- The program must compile fully without errors.

Submission will take place through the LMS. Please zip all of your source code into one folder and submit this `.zip` file. **Do not submit a .rar, .7z, .tar.gz, or any other type of compressed folder.** Especially do not submit one of these files that has simply been renamed to have a `.zip` extension, as then we will be unable to open your file.[2]

[2]This may sound ridiculous, but it has happened several times in the past!

We highly encourage you to re-download and check that your submission contains all your code.

Good Coding Style

Good coding style is a contentious issue; however, we will be marking your code based on the following criteria:

- You should *not* go back and comment your code after the fact. You should be commenting as you go. (Yes, we can tell.)
- You should be taking care to ensure proper use of visibility modifiers. Unless you have a very good reason for it, all instance variables should be private.
- Any constant should be defined as a static final variable. Don't use magic numbers!
- Think about whether your code is written to be easily extensible via appropriate use of classes.
- Make sure each class makes sense as a cohesive whole. A class should have a single well-defined purpose, and should contain all the data it needs to fulfil this purpose.

Extensions and late submissions

If you need an extension for the project, please email Eleanor at `mcmurtrye@unimelb.edu.au` explaining your situation with some supporting documentation (medical certificate, academic adjustment plan, wedding invitation). If an extension has been granted, you may submit via the LMS as usual; please do however email Eleanor once you have submitted your project.

The project is due at **11:59pm sharp**. Any submissions received past this time (from 12:00am onwards) will be considered late unless an extension has been granted. There will be no exceptions. There is a penalty of 1 mark for a late project, plus an additional 1 mark per 24 hours. If you submit late, you **must** email Eleanor so that we can ensure your late submission is marked correctly.

The distribution of marks is on the final page.

Marks

Project 1 is worth **8** marks out of the total 100 for the subject.

- Features implemented correctly – **4 marks**
  - Pegs are generated correctly: **1 mark**
  - Ball appears on click: **1 mark**
  - Ball starts moving towards the mouse and falls with gravity: **1 mark**
  - Pegs disappear on contact with the ball: **1 mark**
- Code (coding style, documentation, good object-oriented principles) – **4 marks**
  - Delegation – breaking the code down into appropriate classes (**1 mark**)
  - Use of methods – avoiding repeated code and overly long/complex methods (**1 mark**)
  - Cohesion – classes are complete units that contain all their data (**1 mark**)
  - Code style – visibility modifiers, consistent indentation, lack of magic numbers, commenting, etc. (**1 mark**)