The University of Melbourne
School of Computing and Information Systems
SWEN20003 Object Oriented Software Development

ShadowFlap

Project 1, 2021
Released: Friday, 27/08/2021
Initial Submission Due: Wednesday, 01/09/2021 at 8:59pm AEST
Project Due: Friday, 10/09/2021 at 8:59pm AEST

Please read the complete specification before starting on the project, because there are important instructions through to the end!

Overview

Welcome to the first project for SWEN20003, Semester 2, 2021. Across Project 1 and 2, you will design and create an imitation of the classic Flappy Bird game. Flappy Bird is a game where you, as the player, controls a bird by pressing the space bar (or tapping the screen on touchscreen devices) and attempts to fly the bird between pairs of pipes. In this project, you will create the basis of a larger Flappy Bird game that you will complete in Project 2B.

This is an **individual project**. You may discuss it with other students, but all of the implementation must be your **own work**. By submitting the project you declare that you understand the University's policy on academic integrity and aware of consequences of any infringement.

You may use any platform and tools you wish to develop the game, but we recommend using IntelliJ IDEA for Java development as this is what we will support in class.

The purpose of this project is to:

- Give you experience working with an object-oriented programming language (Java),
- Introduce simple game programming concepts (2D graphics, input, simple calculations)
- Give you experience working with a simple external library (Bagel)

Figure 1 shows a screenshot from the game after completing Project 1.

Bagel

The **B**asic **A**cademic **G**ame **E**ngine **L**ibrary (Bagel) is a game engine that you will use to develop your game. You can find the documentation for Bagel here.

Figure 1: Completed Project 1 Screenshot

Graphics Concepts Revision

Coordinates

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.

Frames

The program's logic is updated 60 times per second, each time the screen is cleared to a blank state, and all of the graphics are drawn again. Each of these steps is called a **frame**. Every time a frame is to be rendered, the `update()` method in `ShadowFlap` is called. It is in this method that you are expected to update the state of the game.

Velocity

Moving objects have a **velocity** that can be represented by a **vector** (v_x, v_y). A vector is composed of magnitude and a direction. Often, the magnitude is the step size that the object takes. In this case, we scale the vector to have magnitude 1 (called a unit vector) and simply use its direction component, which was described in detail in Workshop 4. 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. You can refer to Workshop 4 solution for an implementation of 'moving' objects. However, the implementation in Workshop 4 solution does **not** consider acceleration.

Collision

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.

Game Elements

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

The Background

The background should be rendered on the screen and completely fill up your window throughout the game. The image that acts as the background is supplied to you in the skeleton package. There is no need to scale or rotate the image in anyway. Remember that the default window size should be `1024 * 768` pixels.

Messages

All messages should be rendered with the font provided in `res` folder, in size 48. The location at which each message should be rendered will be described sections below.

Start of Game

When the game is run, an instruction message that reads `PRESS SPACE TO START` should be rendered at the **centre** of your window, on top of the background and in the font provided. Note that nothing else should be in the window at this time.

The Bird

To imitate the flapping motion, `birdWingUp.png` should be rendered once every 10 frames. In all other frames where the bird is active, `birdWingDown.png` should be rendered.

Spawning

When the player hits the space-bar for the first time to start the game, the bird should initially spawn at `(200, 350)`. The bird's `x` coordinate should remain constant throughout the game. The image to be used in spawning is `birdWingDown.png`.

Falling

After the game has started, the bird (albeit very briefly) has an initial speed of 0 pixel per frame before gravity comes into play. When **not** flying (see the following section for flying behaviour), the bird's falling speed is **accelerated** by gravity at of 0.4 pixel per frame-squared (not 9.8 like on Earth!). However, the bird **cannot fall faster than 10 pixels per frame**. Remember that every time a frame is rendered, the `update()` method in the main `ShadowFlap` class is called.

Flying

After the game has started, the player can press the space bar for the bird to *fly*. Whenever the space bar is pressed, the bird should start flying upwards by 6 pixels per frame. Then, gravity should pull it down in the same behaviour as described in the **Falling** section above.

The Pipes

In this game, there is one set of pipes that consists of a top pipe (which stems from top of the window and points downwards) and a bottom pipe (which stems from the bottom of the window and points upwards). The gap between top and bottom pipe should be 168 pixels. The set of pipes should initially spawn at the window's right border. Additionally, the top and bottom pipes should stem from the top and bottom borders of the window respectively.

You are provided with an image of a top pipe, which you must rotate to become a bottom pipe when appropriate. The top pipe image you are provided with is long enough to span the entire window height, but remember that you can draw images at negative coordinates.

Once spawned, the set of pipes should start moving from their initial position to the left border of the screen when the player presses the space bar to start the game, at a speed of 5 pixels per frame. If there is no collision with the bird, the pair of pipes should leave the window from the left border.

Score Counter

While the bird is active, a score counter should be rendered in the top left corner of the screen, in the format of `SCORE: k`, with `k` being the correct score at any moment of the game. The bottom left of the score counter message should be located at `(100, 100)`.

Collision Detection

You are required to detect collision between the bird and the pipes. Remember that you can assume the provided images are rectangles and make use of the `Rectangle` class in Bagel. If there is a collision, then the game is over and a message of `GAME OVER` should be rendered at the centre of the window. Additionally, the score should be rendered at 75 pixels below the `GAME OVER` message, in the format of `FINAL SCORE: k`.

Out-of-Bound Detection

If the bird leaves the window from the top border or the bottom border, it is considered out-of-bound. Once an out-of-bound is detected, the game should behave in the same way as when a collision is detected.

Win Detection

Once the bird's `x` coordinate surpasses the x coordinate of a pair of pipes, the score increases by 1. A winning message that reads `CONGRATULATIONS!` should be rendered at the centre of the window. Additionally, the score should be rendered at 75 pixels below the `CONGRATULATIONS!` message, in the format of `FINAL SCORE: k`.

Your Code

You must submit a class called `ShadowFlap` that contains a `main` method that runs the game as prescribed 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 correctly, as well as the effective use of Java concepts. As always in software engineering, appropriate comments and variables/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:

- Draw the background on screen
- Draw the game instruction message on screen
- Draw the score counter on screen
- Make the top pipe and bottom pipe spawn at given coordinates
- Implement logic to make the top pipe and bottom pipe move towards left border of the window
- Make the bird spawn at given coordinates
- Implement 'gravity' logic to make the bird fall when space-bar is not pressed
- Implement logic to make the bird 'fly' when space-bar is pressed
- Implement logic to make the bird's wings flap
- Implement logic for collision detection between the bird and the pipes
- Implement logic to keep scores
- Implement win detection and draw winning message on screen with correct score
- Implement lose detection and draw losing message on screen with correct score
- Implement out-of-bound detection and draw losing message on screen with correct score

Supplied Package

You will be given a package called `project-1-skeleton.zip` that contains the following: (1) Skeleton code for the `ShadowFlap` class to help you get started, stored in the `src` folder. (2) All graphics and fonts that you need to build the game, stored in the `res` folder. (3). The `pom.xml` file required for Maven. Here is a more detailed description:

- `res/` – The graphics and font for the game.
  - `birdWingDown.png`: The image to represent a bird with downward-facing wing.

  - `birdWingUp.png`: The image to represent a bird with upward-facing wing.
  - `pipe.png`: The image to represent a pipe that points downwards.
  - `background.png`: The image to represent the background.
  - `slkscr.ttf`: The font to be used throughout this game at size 48.
- `src/` – The skeleton code for the game.
  - `ShadowFlap.java`: The skeleton code that contains entry point to the Game and `update()` method.
- `pom.xml`: File required to set up Maven dependencies.

Submission and Marking

Initial Submission

To ensure you start the project with a correct set-up of your local and remote repository, you must complete this Initial Submission procedure on or before **Wednesday, 01/09/2021 at 08:59pm**.

1. Clone the `[user-name]-project-1` folder from GitLab.
2. Download the `project-1-skeleton.zip` package from LMS, under Project 1.
3. Copy the contents of `project-1-skeleton.zip` to the `[user-name]-project-1` folder **in your local machine**.
4. Add, commit and push this change to your remote repository with the commit message `"initial submission"`.

After completing this, you can start implementing the project by adding code to meet the requirements of this specification. Please remember to add, commit and push your code regularly with meaningful commit messages as you progress.

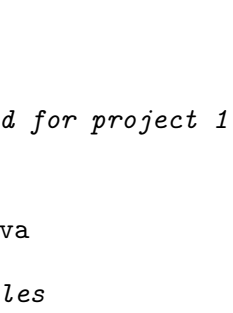
Please try to complete Initial Submission **before** starting your project, so you can make regular commits and push to gitlab since the very start. However, if you start working on your project locally before completing Initial Submission, that is fine too, just make sure you move all of the contents from your project folder to `[user-name]-project-1` in your local machine.

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 GitLab. You are to submit to your `<username>-project-1` repository. An example repository has been set up here showing an ideal repository structure.

At the **bare minimum** you are expected to follow the structure below. You **can** create more files/directories in your repository if you want.



On 01/09/2021 at 9:00pm, your latest commit will automatically be harvested from GitLab.

Commits

You are free to push to your repository post-deadline, but only the latest commit on or before 01/09/2021 8:59pm will be marked. You **must** make at least 5 commits (excluding the Initial Submission commit) throughout the development of the project, and they must have meaningful messages (commit messages must match the code in the commit). If commits are anomalous (e.g. commit message does not match the code, commits with a large amount of code within two commits which are not far apart in time) you risk penalization.

Examples of **good, meaningful** commit messages:

- implemented gravity logic
- fix bird flying behaviour
- refactored code for cleaner design

Examples of **bad, unhelpful** commit messages:

- fesjakhbdjl
- i'm hungry
- fixed thingzZZZ

Good Coding Style

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

- 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 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 Betty at `betty.lin1@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 Betty once you have submitted your project with an extension.

The project is due at **8:59pm sharp**. Any submissions received past this time (from 9:00pm 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 Betty so that we can ensure your late submission is marked correctly.

Marks

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

- **NOTE:** Not completing the Initial Submission (described in the Submission and Marking section here) before beginning your project will result in a **3 mark penalty**!
- Features implemented correctly – **6.5 marks**
  - Background is rendered correctly and all entities spawn at given coordinates: (**0.5 marks**)
  - Bird's flying behaviour is implemented correctly: (**1 mark**)
  - Bird's falling behaviour is implemented correctly: (**1 mark**)
  - Top pipe and bottom pipe moves at the same pace until they leave the window: (**1 mark**)
  - Win detection is implemented correctly with winning message and correct score rendered: (**1 mark**)
  - Collision detection is implemented correctly with game-over message and correct score rendered: (**1 mark**)
  - Out-of-Bound detection is implemented correctly with game-over message and correct score rendered: (**1 mark**)
- Code (coding style, documentation, good object-oriented principles) – **1.5 marks**
  - Delegation and Cohesion – breaking the code down into appropriate classes, each being a complete unit that contain all their data: (**0.5 mark**)
  - Use of Methods – avoiding repeated code and overly long/complex methods: (**0.5 mark**)
  - Code Style – visibility modifiers, consistent indentation, lack of magic numbers, commenting, etc.: (**0.5 mark**)