DoodleJump

Help Slides Released: Monday, October 22nd

Design Section: Tuesday, October 23rd - Thursday, October 25th
Bring Mini-Assignment to your section!

Early Handin: Tuesday, October 30th, 11:59 pm
On-Time Handin: Thursday, November 1st, 11:59 pm
Late Due Handin: Saturday, November 3rd, 11:59 pm
Reminder: the SunLab closes at 10pm on Fridays and Saturdays

To run the demo: `cs0150_runDemo DoodleJump`
To run a snazzy demo: `cs0150_runSnazzyDemo DoodleJump`
Demos may not work well over ssh! Try FastX or the Sunlab.

To install: `cs0150_install DoodleJump`
To hand in: `cs0150_handin DoodleJump`

Silly Premise

Phineas, Ferb and Perry are bouncing on a trampoline, and Phineas decides that it would be cool to have trampolines everywhere. Meanwhile, Doofenshmirtz too has an idea to create a Tell-the-Truth-inator. Phineas and Ferb create a Anti-Gravity Ray to shoot the trampolines and bounce with their friends. To everyone's surprise Doofenshmirtz comes with his Tell-the-Truth-inator. Where is Perry to save the day? They now hope they can jump faster to escape the rays.

New Concepts Covered

- `for` loops
- `java.util.ArrayList`
- Physics simulation
- Representing parts of your program graphically and logically
- More Java generics
- More JavaFX and GUI!

Collaboration Policy Reminder

From the collaboration policy:

Collaboration on project design is not allowed, except for during the design discussion for this project under the supervision of your discussion leader TAs. Otherwise, **no collaboration is allowed on project-specific details.** You are not allowed to discuss the classes you will be using in your project, what methods you will be writing, inheritance hierarchies, the design discussion mini-assignment, or any other design components of the program. You may not discuss implementation or debugging of code for projects with anyone except the course staff.

Importantly, though, you may absolutely discuss general (i.e., not assignment-specific) CS15 concepts with anyone, including other current students. The following falls into this category:

- Going over CS15 lecture slides, our (non-assignment) handouts, Javadocs, etc.
- Discussing object-oriented programming concepts, such as polymorphism
- General syntax questions. For example, “How do I declare an instance variable?”
- How to work remotely, and how to move and hand in files

Note that in each case, any examples used must be from the lectures or your own creativity – you may not discuss how even broad design concepts like containment pertain to a specific assignment.

Mini Assignment

Before starting to code your program, you should think through your design carefully by completing the design section mini assignment! This will help you think through your containment and inheritance relationships as well as some of the larger algorithms that you will code in this project. Please bring a printed copy of your mini assignment to your section. Remember, though, that you'll also need to email a PDF to your section leaders by the listed due date to receive credit.

Assignment Specifications

Before you read this section, make sure you play around with the demo! Better yet, have it open while you read this and play along. Doing so will make understanding the assignment much easier.

In this program, you will write the CS15 version of the game Doodle Jump. This game features a "doodle" whose goal is to hop along a series of never-ending platforms without falling to the ground. The doodle will fall according to the laws of gravity (more on this later) until it hits a platform, at which point it will bounce up until gravity brings it down again. The doodle's horizontal position can be controlled with the left and right arrow keys.

As the doodle jumps higher and higher, the game will need to scroll vertically. By scroll vertically, we mean that when the doodle hits a certain height (say the midway point of the window), rather than have the doodle move higher on the screen, all the platforms should move lower (giving the illusion that the doodle is still climbing up). See the demo for an example.

As the game scrolls upward, old platforms need to disappear off the bottom of the screen and new platforms need to be regenerated at the top of the screen so that the doodle will have more places to jump.

The game ends when the doodle falls past the bottom of the screen because it missed all the platforms as it fell. A message should be displayed to tell the user that the game is over, and at this point the doodle can no longer be moved left or right.

Your game should also include a quit button that closes the game window.

Helpful Design Hints

This program is large and complex, so it is more important than ever to **design all your classes before you write any code**. We mean it!

Well-chosen relationships between your classes will be very important. You have some experience with making a GUI from Cartoon, so remember what you have learned when creating the GUI components. We have provided you with an `App` class that takes care of the mainline, but it is your job to create a layout using `javafx.scene.layout` classes as well as any other graphical components you might need. Once you have an idea of how you want your GUI to look, you should think about the communication between the GUI, the `Doodle`, and the platforms. Furthermore, you have some experience with `Timeline`s in Java from Cartoon, which will come in handy for animation and physics simulation.

Another important design decision will be choosing how to generate your platforms. The game would be pretty boring if the same platforms were always generated, so you'll want to incorporate some randomness into your platform generation. At the same time, your platforms should be generated in such a way that the doodle will definitely be able to get from one

platform to the next, so making platform generation completely random won't work either. Your platforms should be generated "semi-randomly," meaning that their location should be chosen randomly given a certain set of constraints. Don't worry! We have some resources that will help you with this part.

So how will you keep track of your platforms? By using a `java.util.ArrayList`, that's how! An `ArrayList` is a data structure that allows you to store and access a dynamic number of objects in a resizable list, which is exactly what we're looking for: a way to keep track of all the platforms in our game.

But wait! Why can't we use an array for this? Since the horizontal and vertical positioning of platforms is based on a degree of randomness, varying numbers of platforms may fit on the screen at any one time. Remember — arrays always have a set capacity. The `java.util.ArrayList`'s dynamic capacity lends itself very well to handling this type of dynamic storage, whereas an array of fixed capacity would not. `ArrayList`s also allow you to easily add and remove elements.

The Javadocs for the `java.util.ArrayList` data structure highlight some of the important methods that have already been defined for `ArrayList`s, so you should become familiar with these before starting this project.

For a review of `ArrayList`s, look over the later slides in the Arrays lecture. This will also help you understand how to declare and instantiate an `ArrayList` that holds a unique type of object. Lab 5 also provides extra practice with `ArrayList`s.

Physics Simulation

Just like in Cartoon, your doodle needs to be "moving". Remember that this can be done by using a `Timeline`, and updating its position at every `KeyFrame`. Additionally, your doodle should fall under the influence of gravity. This means that your doodle won't have a constant velocity -- its velocity will update together with your doodle's position!

The exact calculations surrounding how your doodle should follow gravity involve a few kinematic equations, none of which are necessary for CS15. The constant for gravity (explained more in-depth below) combined with the duration of your `KeyFrame` can be used to calculate the updated velocity and position of your doodle at every `KeyFrame`. Here's some pseudocode for making this calculation each time the `Timeline` cycles through to the end of a `KeyFrame`:

```
    updated velocity = current velocity + GRAVITY * DURATION
updated position = current position + updated velocity * DURATION
```

Essential Constants

The constants `GRAVITY` and `DURATION` you see above are already defined for you in the `Constants` class, as well as suggested constants for dimensions of the Doodle and the platforms, though you are welcome to change them. Be sure to review the Math/Making Decisions lecture for more information on constants!

`GRAVITY` corresponds to the downward force on your doodle. On Earth, gravity points toward the center of the planet with a force of *-9.8 m/s*^{2}. This isn't super helpful, though, in the world of JavaFX. Because the JavaFX coordinate system defines the positive y-direction as downward, our acceleration constant is positive (approximately 1000 pixels/s^2). The `DURATION` constant corresponds to the amount of time, in seconds, that elapses for each iteration of your `KeyFrame`, and thus should also be used to initialize your `KeyFrame`.

Another constant you will see in your `Constants` class is `REBOUND_VELOCITY`. `REBOUND_VELOCITY` is the initial velocity the Doodle achieves right after jumping on a platform. Set your doodle's velocity to `REBOUND_VELOCITY` whenever your doodle hits a platform while falling to make it jump.

Take a second to think about why this works. Each time your doodle bounces, including at the very start of the game, it has an upward speed of `REBOUND_VELOCITY`. As we have seen, though, each tick of the `TimeHandler` corresponds to the passage of a discrete unit of time, during which `GRAVITY` acted on your doodle. `GRAVITY`, which is a constant downward force, *reduces* your doodle's non-constant upward speed by a particular amount (which you have the equations to calculate) at each of these ticks. Once, after several ticks, `GRAVITY` has reduced your doodle's *upward* speed to 0, it continues to push down on your doodle, increasing its *downward* speed until your doodle either drops off the screen, or intersects a platform.

Now you may be wondering—how do we detect when the doodle collides with a platform? You can do this by using the `javafx.scene.Node` `intersects` method! Be sure to check the Javadocs to understand how it should be implemented. Importantly, make sure to only check for platform collisions when the doodle is falling—you don't want it to bump into anything on the way up!

Timeline Updates

By this point it should be clear that your `Timeline` is going to have to do a bit more than it did in Cartoon. To make your lives easier, we have provided a list of steps that need to be taken at the end of each `KeyFrame`. Remember that your `EventHandler` doesn't need to do all of this directly. It can (and should) delegate appropriate tasks to other methods and objects. Inside your `handle(...)` method (or its helper methods!), you should:

1. Calculate a new y-velocity for the doodle using your physics simulation equation

2. Calculate your doodle's new potential position based on the new y-velocity. Note that you shouldn't change the doodle's position at this point
3. Check to see if the doodle will be above the horizontal middle of the window based on its new potential position. Recall that the top left of the window is at position (0, 0), and that the y-position increases as you move down the window. Assuming that your window has a height of 400 pixels, if the doodle's y-position is less than 200 pixels, then it is above the horizontal middle of the window
    a. If the doodle is above the midpoint of the screen
        i. Calculate and store how far the doodle is above the midpoint
        ii. Set the doodle's position to the screen midpoint
        iii. For each platform
            1. Lower by how much the doodle was above the midpoint
    b. If it isn't above the midpoint of the screen, set the doodle's position to the calculated potential position.
4. If the doodle intersects with a platform AND the doodle is falling
    a. Set the doodle's y velocity to `REBOUND_VELOCITY`
5. Check to see if any of the platforms are below the bottom of the window
    a. If a platform is below the bottom of the window, remove it both graphically and logically (more on this later)

Keyboard Interaction

Keyboard input in DoodleJump should be similar to the key handling that you wrote for Cartoon -- the Cartoon handout has more information on specifics of handling key input, but here are some helpful reminders on handling key input in JavaFX:

- Make sure to `consume()` your `KeyEvent`s after you are finished with them to prevent them from triggering other actions in your program.

- For the keys to receive input, you need to "request focus" on the `Pane` that the key handler is added to. "Focus" deals with which graphical element is currently "selected." To register and handle key input, we want to give focus to the `Pane` associated with the key handler. This can be accomplished by calling `Pane`'s `requestFocus`() method **after** the `Pane` is visible (a.k.a. after `stage.show()` is called). Alternatively, you can call `setFocusTraversable(true)` on your `Pane` after you add the key handler to your `Pane` using `setOnKeyPressed`, and call `setFocusTraversable(false)` on all other `Pane`s and buttons.

Platform Generation and Removal

Begin by creating the bottommost platform so it is just below your doodle's start position. Next, you'll want to "semi-randomly" generate the platform above that one. Platforms cannot be

generated completely randomly because the doodle needs to be able to reach the next platform. Therefore, you should decide on minimum and maximum x and y distances between two consecutive platforms to ensure that the doodle is able to jump from one to the other.

Here's an example of how to use this information to generate the platform's location:

> *Assume: Platform p has x-y coordinates (300, 600)*
> *Assume: Doodle can move up 100 pixels, and left or right by 200 pixels per jump, before starting to fall.* ***(This will differ based on the constants you use, make sure your game is playable)***
> *Note: The second platform must be above the first platform. For this example, say the height of each platform is 10 pixels.*
>
> With the above assumptions, the maximum x-distance between consecutive platforms should be 200, the maximum y-distance should be 100, and the minimum y-distance should be 10. This means the second platform's x-position must be a random value between 100 and 500 and its y-position should be a random value between 500 and 590.
>
> **Note:** Platforms should not generate outside the board. You must also consider the pane's width when constraining your generation to stay within the board.

The third platform will need to be positioned relative to the second platform, and so on. Do you see a pattern?

How should you generate *all* the platforms? A loop! At each iteration of the loop, you are creating a new platform whose position is relative to the platform created in the previous iteration of the loop. Your loop will exit when the last platform created meets a certain condition — more on this below.

Here's an idea of how you might want to implement this:

1. Create your first platform and store it in a current topmost platform variable. Position the platform directly under the Doodle's starting location (probably somewhere near the bottom of the Pane).
2. Add that platform to your ArrayList of platforms.
3. While the current topmost platform's y position is not above the top of the Pane:
    a. Create a new platform and add it to the ArrayList.
    b. "Semi-randomly" position the new platform relative to the current topmost platform.
    c. Update the current topmost platform variable to reference this new platform.

Remember: To graphically add or remove `Node`s, you should do so from the Pane that contains them. To add or remove `Nodes` in your game logic, you should make sure to add or

remove them from the data structure in which they're contained. This is *not* the same as the graphical Pane!

What to Make Sure You Code

There's plenty of room for creativity in this assignment, but **before you do any extra credit, make sure your game meets all of the following specifications!** Note, too, that these are different than the Minimum Functionality requirements. Rather, this is a guideline to help you parse out the important pieces of your program.

1. It has a `Doodle`! (duh…) The `Doodle` should be its own class - this makes it easier for your doodle to store and calculate relevant information. Remember: delegation!
2. You have a `Timeline`. At the end of your `KeyFrame`, the doodle's velocity is calculated. The doodle should move according to motion equations and the `Timeline`.
3. The doodle can be moved left and right with the left and right arrow keys.
4. You have platforms (instances of a `javafx.scene.shape.Shape`, or you can write your own `Platform` class containing a `Shape`). The position of each new platform is semi-random. It should be constrained within a certain distance of the platform below it so it is reachable.
5. When the doodle is falling and intersects a platform, it bounces.
6. The platforms scroll down when the doodle reaches the approximate midpoint of the screen.
7. As the doodle jumps higher and the game scrolls, new platforms are created so that the doodle can continue to climb indefinitely.
8. If the doodle falls past the bottom of the screen, a game over label appears and the doodle can no longer be moved with the arrow keys.
9. The game should display a quit button that calls `System.exit(0)` or `Platform.exit()` when clicked.
10. Collisions with the sides of the frame

Bells and Whistles

Once you've met all of the above requirements, feel free to snazz up your DoodleJump with some bells and whistles! (check out the snazzy demo for inspiration). Reminder: you are not eligible for extra credit unless your project meets minimum functionality requirements, defined at the bottom of the handout. Here are a few ideas:

- Different levels of difficulty
- A button to restart the game
- Make the Doodle a composite shape

- Keeping track of the player's score
- Support for multiple players
- 'Smooth' left/right movement with the arrow keys
- Different types of platforms - movable, breakable, high-jump
- Toys to alter the Doodle's velocity - jetpacks, helicopter-hats, etc.
- Venomous black holes and hungry monsters

Keep in mind that most TA's have not implemented every one of these extra credit features, so we cannot guarantee support for them (though they will try send you in the right direction). As with Cartoon, we are unable to give extra credit for sound, as the department computers do not have the proper support libraries.

Coding Incrementally

Figuring out where to begin on this project can be a bit overwhelming. It will make your life a lot easier to get the program working a little bit at a time. Please don't try to write a ton of code and then run it! You'll just cause more headaches for yourself later. **TAs will turn you away from hours if you have not demonstrated that you have carefully attempted each part of the assignment before moving on to the next step.** We suggest following these steps:

1. Make your game window show up.
2. Get your doodle to display.
3. Set up the `Timeline` that will be in charge of updating the doodle's location and displaying the graphical changes. (To test if your `Timeline` is working, you can start with printlines. Afterwards, you may want to move the doodle slightly at the end of your `KeyFrame`.)
4. Set up the key event handler so you can use the left and right arrows to get your doodle to move.
5. Add some physics simulation so that your doodle falls.
6. Start with creating one platform and add and implement/test collision detection with that platform.
7. Generate a whole screen of semi-randomly positioned platforms so that your doodle can jump its way upwards!
8. Add the vertical scrolling so that when the doodle tries to pass a certain height, it stops moving, and all the platforms move downward. (**WARNING!** This step is tricky! Really think about the best way to implement this -- careful design and pseudocode will simplify this step greatly.)
9. As platforms scroll down, be sure to delete them and generate new ones both graphically and logically.

README

In addition to describing your design choices, if you decide to implement any extra credit, please detail it in an EXTRA CREDIT section of your README.

You are expected to create your own README file. Please refer to the README guide for information on what information your README should contain and how you should format it.

Minimum Functionality Requirements

MF Policy Summary: *In order to pass CS15, you will have to meet minimum functionality requirements for all projects. If you don't meet them the first time around, you may hand the project in again until you succeed, but you will keep your original grade. MF requirements are* ***not*** *the same as the requirements for full credit on the project. You should attempt the full requirements on every project to keep pace with the course material. An 'A' project would meet all of the requirements enumerated in the assignment specification section of the handout and have good design and code style.*

To meet minimum functionality for DoodleJump:

- A doodle and platforms appear on screen.
- The doodle's position and velocity are updated continuously with a `Timeline`.
- The doodle can collide with and bounce off of platforms.
- Platforms scroll downward when the doodle reaches some point on the screen.
- The doodle moves left and right in response to key presses.