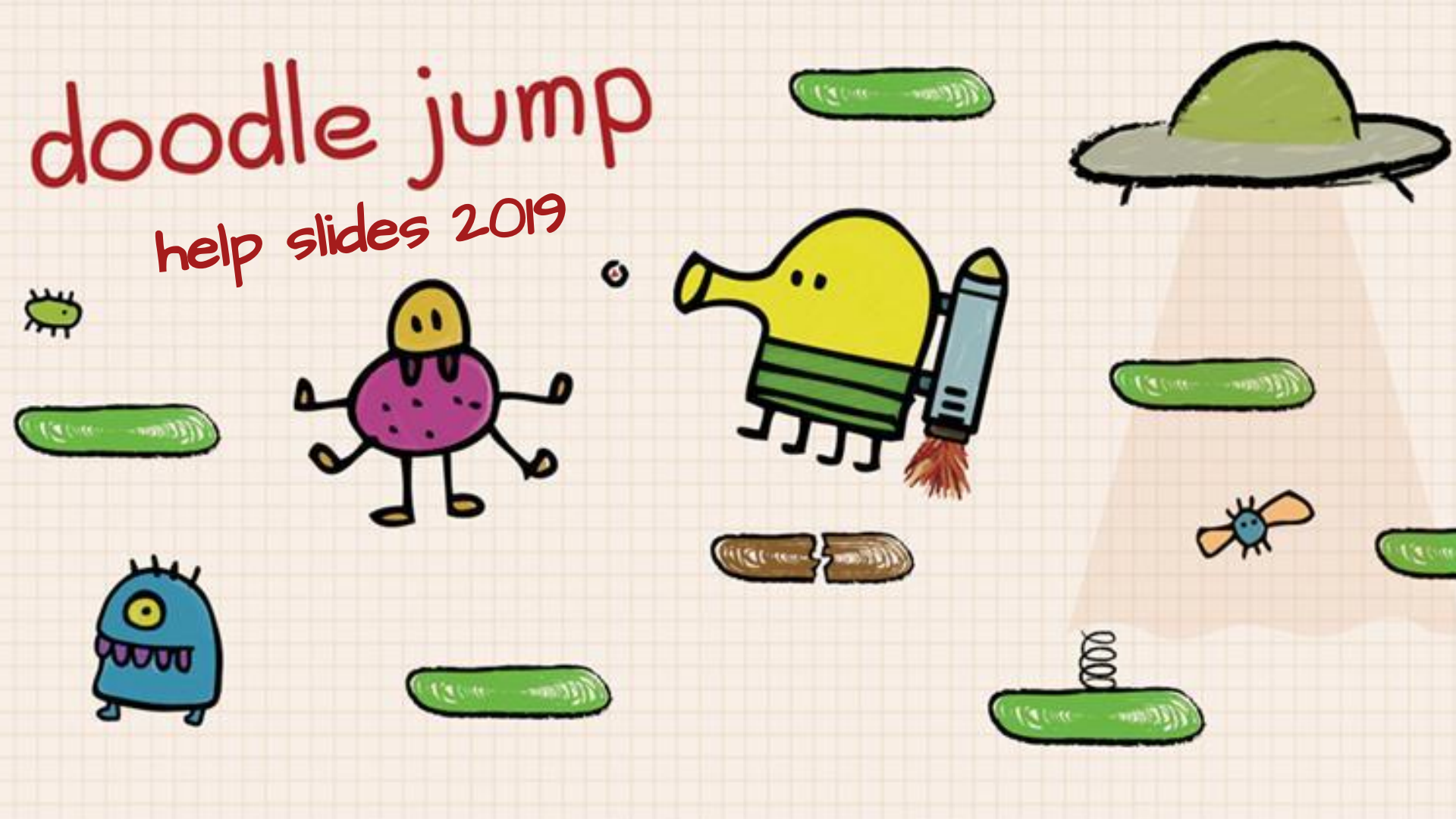
doodle jump

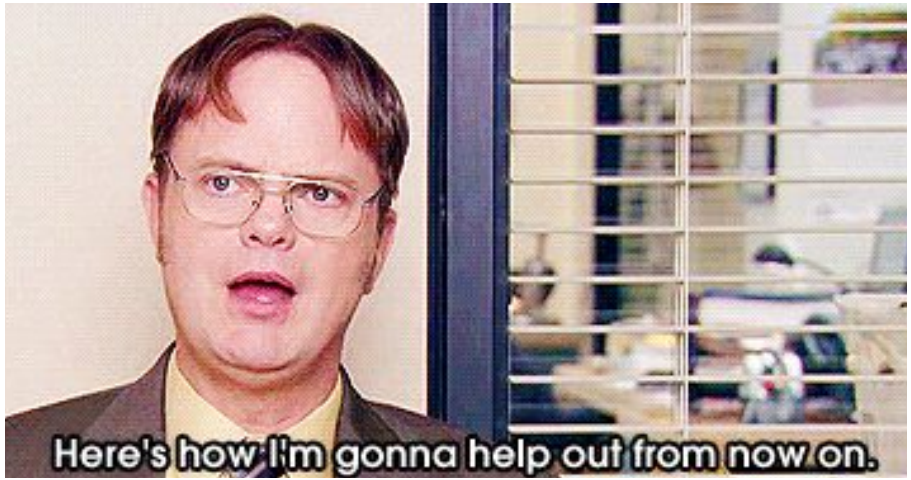
help slides 2019



Help the office members learn basic trampoline skills during safety training!

Help Session Topics

- Design & Structure
- Incremental Coding
- Physics Simulation
- Key Input
- ArrayLists
- Platform Generation
- Vertical Scrolling
- Common Bugs

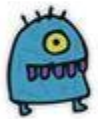


Game Class 

Why use a `Game` class?

- Effective object-oriented programming is focused on **modeling**
  - How can we model properties and functionality as a system of classes?
- Imagine an incredibly complicated game like chess.
  - Would we want the thousands of lines of game logic to all live in the `PaneOrganizer` class? Probably not 😬
  - `PaneOrganizer` has a clear modeling job — it should handle the overall organization of `Pane`s and the relevant **graphical** components.
  - Important application of *abstraction* — a separate class `Game` should handle **logical** game play

Code Incrementally!!! (please)



- DoodleJump is a larger project than Cartoon — it will be much easier if you can break it down into more **manageable pieces**!
    - Eclipse will catch syntax errors — we **HIGHLY, HIGHLY** recommend using Eclipse for the remainder of your CS15 projects! Using Eclipse lets you quickly catch and fix compiler errors.
    - Test **each** method that you write and make sure that it works as intended before writing the next one!
        - **Isolate** each method — test bouncing on a single platform before trying to integrate with scrolling and key input, for instance.

Code Incrementally!!! (please) 

- **Here is a basic outline for working on projects with a GUI component!**

1) Get your `Stage`, `Scene` and main `Pane` to show up
2) Start adding your other JavaFX `Node`s, additional `Pane`s, etc.
3) Add game logic that controls those `Node`s method by method
4) Put the finishing touches on it - add labels, a quit button, etc.

- Check the **handout** for a DoodleJump-specific set of **incremental coding steps**

Physics Simulation

- Your Doodle shouldn't just move up and down at a constant speed, it should *accelerate* or *decelerate* under the influence of gravity!

- **So what does this actually mean?**
  - You will utilize a `javafx.animation.Timeline` to update the **position** and **velocity** of your Doodle, as determined by the equations of motion in the handout and the provided `GRAVITY` constant.

- **Remember!** Since the positive y-axis runs **down** on a computer screen, the `GRAVITY` constant will be **positive**.

Physics Example (1/3) 

- Let us assume that your Doodle initially starts at rest, with it's **position** set to (0,0) and its **velocity** set to 0.

- At the end of your `KeyFrame`, you need to calculate a new position and velocity for your doodle using the following equations:

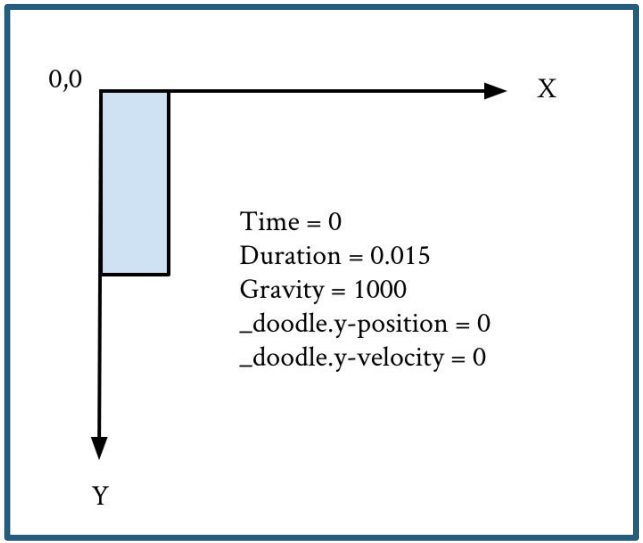
updatedVelocity = currentVelocity + ACCELERATION * DURATION

updatedPosition = currentPosition + updatedVelocity * DURATION

Physics Example (2/3)

In the image example:

currentPosition = 0
currentVelocity = 0
GRAVITY = 1000*
DURATION = 0.015



*Remember our acceleration constant is **positive** (approximately 1000 pixels/s^2)

Physics Example (3/3) 

- **Using the previous equations, we get:**

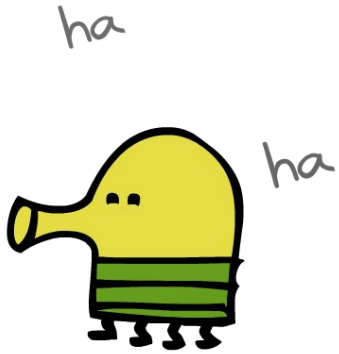
  updatedVelocity = 15
  updatedPosition = 0.225

- **Now we update Doodle's variables:**

  currentVelocity is set to 15
  currentPosition is set to 0.225

And that's it! Just store the new values for **currentPosition** and **currentVelocity** and repeat!

Collision (1/2)



How do we make the Doodle jump when it hits a platform?

- By using the `boolean intersects(double x, double y, double width, double height)` method of the `Node` class, that's how!
  - This method returns **true** if the `Node` that calls the method is overlapping a point in the "box" of size `width`, `height` that is sitting at the location `x`, `y`. This will work well if the `x`, `y`, `width`, and `height` correspond to the `Platform` we're checking for intersection with.
    Remember: a `Shape` is a subclass of `Node`!
- Is it enough to just check if your Doodle is intersecting a platform? Or should your Doodle rebound only under certain conditions?
  - For example, should your Doodle rebound if it is moving up? **No!**

Collision (2/2)

Now that we know how to check if our Doodle is colliding with one platform, how can we check to see if it's colliding with any platform?

- By using a loop!
- As you check all your platforms for a collision, you just have to set your Doodle's **y-velocity** to the `REBOUND_VELOCITY` constant if a collision is detected!

Key Input: Overview 

- **How will you make your Doodle move right and left?**
  - You will be using an `EventHandler`, just like in Cartoon, with an `Event` type of `javafx.scene.input.KeyEvent`.
- **They work in the following way:**
  1. You decide **where** you will have your listener.
     What JavaFX node will listen for key input?
  2. You create your **inner class** that implements the EventHandler interface.
  3. You **specialize** how your class `handle`s `Event`s.

Key Input: KeyCodes 

- **`KeyCode`**s are **enums** that represent the keys on a keyboard. Enums are special data types that represent predetermined values. You can compare them and use them in **switch statements**!

- Calling the method **`getCode()`** on the **`KeyEvent`** passed into your **`handle`** method will return the **`KeyCode`** of the event.

- You can use the **`KeyCode`** enums to check the value returned by the **`getCode()`** method. For example, to check for up arrow button:

```
public void handle(KeyEvent e) {
    if(e.getCode() == KeyCode.UP){ … }
}
```

Key Input: Focus

If your doodle is not responding to key input, try the following steps:

- Call the `consume()` method on the `KeyEvent`
  - One way to make sure the program only executes what you indicate in the `EventHandler` is to "consume" the event, which you can think of as throwing away the event after it has done everything you need it to do. To do this, call the `consume()` method on the `KeyEvent` at the end of the method you use it in. You can read more about this **here** (the section under "Consuming of an Event" at the bottom of the page)

- Call `setFocusTraversable(true)` on the relevant `Pane` right after you create it.
  - Just as in Cartoon, to make sure no other node grabs focus inadvertently, you can call `setFocusTraversable(false)` on any other node (e.g. buttons) that is grabbing focus. You can read about this method **here**.

Arraylists and Platforms

- **How many platforms will you have to create?**
  As many as needed to fill the screen!

- **Where will you store all of your platforms?**
  `java.util.ArrayList`

- `ArrayList` is a built-in Java class. You do **NOT** need to make your own ArrayList class. Use **Javadocs** to find methods this class has.
- When using an `ArrayList`, just like arrays, you need to specify the type of element your arraylist will store. For DoodleJump, you will be storing your platforms in the `ArrayList`. So you will specify **the name of your platform class** in the literal < > brackets!

Arraylists

Question: How will you go through **ALL** of the elements in your `ArrayList`?
Answer: Use loops!

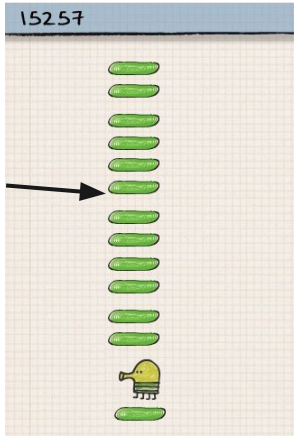
- Look back at the **Arrays lecture** for a review on using loops to iterate through a list of items.
- **HINT:** When platforms are removed, consider how the indices of each platform changes within the ArrayList… If you have a for-loop which should iterate through an ArrayList, think about how the index in the loop could skip indices in the ArrayList as items are removed.

Platform Generation

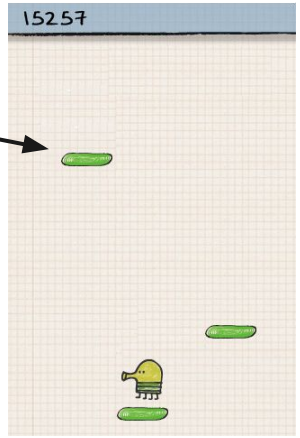
You must generate platforms for your character to hop on and your platforms should be generated “semi-randomly”!

Remember: It’s important that platforms are reachable for the character

Too boring



Can’t be reached!



Platform Generation Steps

1. When the program starts, create enough platforms to cover the entire screen using a `while` loop.

2. When the Doodle jumps to a higher platform, create new platforms!
   - We'll outline the `generatePlatform()` method (next slide) for generating one new platform.

3. When the Doodle is above the center of the panel, move all the platforms down and remove any offscreen platforms.

Platform Generation Pseudocode 

```
generatePlatform() {
    create new platform
    find min x & max x for the new platform
    calculate a random x position between min x and max x
    find min y & max y for the new platform
    calculate a random y position between min y and max y
    set new platforms location to the calculated positions
}
```

Think: What are your constraints on **`min x`** and **`max x`**? How about **`min y`** and **`max y`**?

Random Numbers 

How do you come up with a random location for the new Platform?

- `Math.random()` returns a double value greater than or equal to 0.0 and less than 1.0!

- **Example:**

```
min = minimum reachable x location
max = maximum reachable x location
randomVal = min + (int)((max - min + 1) *
Math.random())
// randomVal is a random value between min
and max inclusive
```

Vertical Scrolling 

When the Doodle passes a certain height, it should stop moving, and all the platforms move downward.

- The Doodle should be moved back to the **midpoint** of the game
- The platforms should **move down** by the amount that the doodle was above the midpoint

> **Example:** If the doodle is one pixel above the midpoint, instead of keeping it there, we move the doodle back to the midpoint and move all platforms down by one pixel. This gives the illusion that the doodle “climbed” one pixel!

Scrolling Pseudocode

After we've updated the Doodle's position according to gravity:

```
if Doodle is above screen midpoint:
    calculate and store how far Doodle is above the midpoint
    set Doodle's position to the panel midpoint
    lower platforms by how much Doodle was above the midpoint
    while the platform list is not empty and still has platforms
    off screen:
        remove those platforms logically and graphically
```

Common Bugs (1/2)

- Quit button stops working when doodle falls below the bottom of the screen
  - Check to see if your panes are overlapping!

- Doodle wiggles/infinite scrolls
  - Check that you are correctly resetting Doodle's position to the middle of the screen

- Make sure that platforms are both *logically* and *graphically* removed! Think about what type of loop would be best for platform removal (HINT: you do not know how many platforms will need to be removed and therefore do not know how many times you should be looping)

Common Bugs (2/2)

- Check that you index in for loops properly! (Printing out the index is a good place to start).
- If you have a class called “Platform,” make sure you don’t use Platform.exit(). Instead, use System.exit(0).
- Too many things being generated?
  - Check out your loops and make sure they’re correct!

Start early…start today…start yesterday!

GOOD LUCK! YOU'VE GOT THIS!