Lecture 2
Announcements
Alc 1 is done!

- Initial feedback for Alc 1
  - Viewports are important – refactor and/or fix them!
  - Remember the goal of OOP: *encapsulate* functionality within objects
  - Organize your projects well! If you’re unsure about your design, talk to a TA or message the Slack

- This week your game will really start to take form!
QUESTIONS?
WHAT’S A SPRITE?
This is a Sprite
Sprites as Bitmap Data

- “Raster” graphics
- Pre-constructed images dynamically placed on the screen
- Designed to represent one type of object in a game
  - Objects may reference different sprites depending on state

This is a raster

This is a vector
Sprites as Animation Data

- Sprites as a filmstrip
- Designed to represent frame-by-frame snapshots of a single game object
- Standardized padding, size, and spacing allows for easy drawing
Typical Sprite File Format

- Multiple sprites per file
- Additional information often (but not always) in config files:
  - Padding
  - Size
  - Locations of a particular object’s sprites
Keep in Mind

- The GameObject’s position/size info and sprite info should be separate
- But keep in mind that they will need to coordinate with each other
  - If you move a GameObject around, it’s sprite should move along with it
  - In some sense then, the sprite is “anchored” onto the GameObject
IMPLEMENTING SPRITES
Sprite Loading

- You should only load a sprite sheet image once
  - Pass this SAME image object around to components that need it
  - Otherwise your game will be very slow!
- Consider making a Resource class which loads in sprite sheets
  - Load in image
  - Handling image index for different sprites
  - Generalizable to other assets like maps, sounds, text, etc…
- If you run into an “Internal graphics not initialized yet” error…
  - Load images on the first tick, not app launch
Drawing Sprites

- **About `g.drawImage(...)`**
- **Rare exception to the no JavaFX rule:**
  - You're going to need to make a JavaFX image
  - Pass in the RELATIVE file path
  - Definitely have a separate directory for all your sprites
- **Your drawing routine should handle different paddings and formats**
Relative Paths

● For All Resource Files:
  ○ Don’t use absolute paths
  ○ `/Users/<username>/Documents/School/cs1971/alc/resources/spritesheet.png` is bad
  ○ `resources/spritesheet.png` is good
  ○ Absolute filepaths won’t work when we try to compile your project
Drawing Sprites

- Draw rectangular chunks from sprite sheet to the canvas
- Don’t cache sub images
  - It isn’t worth the space/time tradeoff
- Remember to draw from your single sprite sheet reference
Implementing **SpriteComponent**

- Has a reference to sprite sheet resource
- Should implement `draw(GraphicsContext g)`
- Once it has a `GraphicsContext` object, it can draw itself
TLDR: Sprite Sheet “Wrapper” Class

● Goal: Give your engine the ability to understand a sprite sheet image file in a way that can be communicated to java classes and functions

● Use the “grid” like nature of sprite sheets to calculate the size of each sprite
  ○ If a sprite sheet is 512 x 256, and there’s 16 x 4 sprites, then each sprite has a size of $(512/16, 256/4) = (32, 64)$

● GameObjects are only going to care about a particular sprite in your sprite sheet, so give that to them
  ○ You could store the positions of the individual sprites (positions within sprite sheet)
  ○ You could treat the sprite sheet like a 2d array
    ■ Getting the sprite in position (3, 4) would look like $(3 \times \text{SpriteWidth}, 4 \times \text{SpriteHeight})$
QUESTIONS?
Lecture 2

Collision Detection
Collision Detection

MOTIVATION
Collisions have consequences

- Collision detection is central to the vast majority of games
- They’re very important
What do we want to collide?

- Points
- Circles
- Axis-Aligned Boxes (AAB)
  - Fancy word for rectangle
- Convex polygons
  - Coming soon™
- Other shapes
  - Not covered

\[
\begin{align*}
&\text{dim} = (w, h) \\
&P_1 P_2 P_3 P_4 P_5
\end{align*}
\]
Collision Detection

DETECTION ALGORITHMS
Point-Circle

- Check if the distance between the point and the center is less than or equal to the radius. If this is true, there is a collision.

\[ \left\| P - C \right\|^2 \leq r^2 \]
Circle-Circle

- Check if the distance between the two centers is less than or equal to the sum of the radii

$$\|C_1 - C_2\|^2 \leq (r_1 + r_2)^2$$
Point-AAB

- Check if the point is within range on each axis

\[ \min_x \leq p_x \leq \max_x \quad \text{AND} \quad \min_y \leq p_y \leq \max_y \]
Circle-AAB

- Find the point on the AAB that's the closest to the circle, then see if that point collides with the circle
  - Closest point: clamp \((c.x, c.y)\) to \([\text{min}x, \text{max}x], [\text{min}y, \text{max}y]\)
  - Then just do point-circle collision with the closest point

\[
P = C \quad \text{max}
\]

\[
P \quad \text{min}
\]
AAB-AAB

- Ensure overlap on each axis
- Project each box onto x and y axes
- If the both pairs of *Intervals* overlap, then there is a collision
Projection

- Imagine a light source with parallel rays
- Shape is between light source and axis
- “Shadow” cast on axis is shape’s projection onto that axis
Creating Projections

- Find the axis you want to project onto
  - This should be a normalized vector (length 1)
  - Vec2d has a normalize method
- `xAxis = new Vec2d(1, 0);`
- `yAxis = new Vec2d(0, 1);`
Creating Projections

- To project a point, take its dot product with the projection axis
  - `double projection = point.dot(axis)`
  - Store `projection` for later
- `Vec2d` has a dot product method
Creating Projections

**Example**: to project an AAB onto the x-axis:
- Project the top-left and bottom-right points onto the x-axis
- Store the two **doubles** in an **Interval**
- This **Interval** is the projection (shadow) of the AAB on the x-axis
Projections $\Rightarrow$ Collisions

- For each axis, check if the corresponding *Intervals* overlap
  - There should be two *Intervals* for each axis
- *Intervals* A and B overlap if and only if:
  - $A_{\text{min}} \leq B_{\text{max}}$ AND $B_{\text{min}} \leq A_{\text{max}}$
- If both axes overlap, the shapes are colliding
**Interval Class**

- Stores two points, and represents the range between them

```java
public final class Interval {
    private double min;
    private double max;
    public bool overlap (Interval other) { // definition here }
}
```
Collision Detection

COLLISION BEHAVIOR
Shapes

- AAB and Circle classes inherit from the same abstract class
  - Shape attributes
  - Implement collision checks
    - Point collisions are only for the mouse; no separate class needed
CollisionComponent

- Contains collision information for a GameObject
- Holds the specific Shape that defines the GameObject’s hitbox
CollisionSystem

- Keeps track of all GameObjects that can collide
- Loops through all pairs of registered objects
- Checks if each pair is colliding
- If there is a collision, both are notified—make sure to only go through each pair once!
Optimization

-Looping through every possible pairing of objects can be inefficient
-Some pairs might never collide
  - Static wall tiles
  - Friendly projectiles
-Add objects to a specific collision layer, and only check layers against each other if the user enables it
QUESTIONS?

Collision Detection
Collision Debugger

- Easy way to test collisions, and completely separate from your project
  - Included as part of the stencil code from Tic
  - You fill in the math, and copy it over to your project after debugging
- Fill in Week2.java
- To launch, run the main method in Display.java
Lecture 2

Tips for Alc II
Removing Units

- Beware the `ConcurrentModificationException`!
  - Happens when you unsafely add/remove objects in a `Collection` during iteration
- Consider a removal queue
- Or, use an `Iterator` to remove objects safely
- This can be generalized to multiple phases of ticks
Sprites

- You’ll need to have sprites in your game to make it pretty!
- Lots of sprites on the internet
- Stealing IP is fun and easy!
  - We do it every lecture
  - Be sure to call it fair use
Tips for Alc II

JAVA TIP OF THE WEEK
Double Dispatch

- If you have a `Circle` and an `AAB` but only know that they're `Shapes`, how do you determine which method to call?
- **The bad way:**

```java
void testCollide() {
    Shape s = new Circle();
    Shape s2 = new AAB();
    s.collides(s2);
}

interface Shape {
    collides(Circle c);
    collides(AAB aab);
    collides(Shape o);
}

boolean collides(Shape o) {
    if (o instanceof Circle) {
        return collides((Circle) o);
    } else if (o instanceof AAB) {
        return collides((AAB) o);
    } else {
        throw new IllegalArgumentException();
    }
}
```
Double Dispatch

- Instead, implement the shape interface

```java
interface Shape {
    collides(Shape o);
    collidesCircle(Circle c);
    collidesAAB(AAB aab);
}

public class Circle implements Shape {
    collides(Shape o) {
        return o.collidesCircle(this);
    }
    collidesCircle(Circle c) { /*code*/ }
    collidesAAB(AAB aab) { /*code*/ }
}
```

Much cleaner!

```java
public class AAB implements Shape {
    collides(Shape o) {
        return o.collidesAAB(this);
    }
    collidesCircle(Circle c) { /*code*/ }
    collidesAAB(AAB aab) { /*code*/ }
}
```

```java
void testCollide() {
    Shape s = new Circle();
    Shape s2 = new AAB();
    s.collides(s2); // Now this just works!
}
```
Tips for Alc II

QUESTIONS
‘Til Next Week!

- Alc II released today!
- Remember to upload your demos :)