Lecture 3
Announcements
Next Week

- No class next Wednesday!
- No project due Tuesday either
- Use this time to catch up, debug, and/or refactor
QUESTIONS?
This week: Wiz 1!

- Real gameplay features!
- Level Loading or Level Generation!
- Actually fun!
Why Sprite Sheets?

- In Alc, we could have a separate sprite sheet for each element
- What about for animation?
- Index different sprites for different movements—but from the same sheet
AnimationComponent

- Like the `SpriteComponent`, but also responds to `tick` calls
  - Can inherit from `SpriteComponent`
- Animate by drawing frames like a flipbook: one after another, then reset
  - Store a list of positions for frames
  - Calculate which frame to use based on elapsed time
- Update frame state on `tick`, draw correct frame on `draw`
  - Use modulo (%) operator when updating the index so it can reset back to 0
QUESTIONS?
Lecture 3

Physics I
Physics I

COLLISION RESPONSE
Collision response

• You know how to detect whether 2 objects collide
• How do we make objects respond in a physically believable way?
• General strategy:
  ○ Move objects out of collision (so they never overlap)
  ○ (also other stuff later)
MINIMUM TRANSLATION VECTOR
Moving out of collision

• Many ways to move the ball out of the wall
• We want the minimum
• “Minimum translation vector” (MTV)
MTV in one dimension

- In 1D, convex shapes are line segments
- These have a 1D MTV
  - Similar to overlap
  - But it has a sign
  - Different for each shape
- To correct their positions, move by half the MTV
Circle vs Circle MTV

- **Magnitude**: $\text{radius1} + \text{radius2} - \text{distance(\text{center1}, \text{center2})}$
- **Direction**: parallel to line connecting the two centers
Circle vs AAB MTV

- If AAB contains circle center
  - Find $p =$ closest point on AAB edge from circle center
  - Length of MTV is $\text{radius} + \text{distance(}\text{center}, p\text{)}$;
  - MTV is parallel to the X or Y axis

- Otherwise
  - Clamp circle center to the AAB (like what we do for Circle/AAB collisions)
  - Length of MTV is $\text{radius} - \text{distance(}\text{center, clampedCenter}\text{)}$
  - MTV is parallel to the line connecting the two points
AAB vs AAB MTV

- Only four possibilities
  - Move up (this.maxY - that.minY)
  - Move down (this.minY - that.maxY)
  - Move left (this.maxX - that.minX)
  - Move right (this.minX - that.maxX)

- Return the shortest
Static Objects

- Some objects shouldn’t move, even when collided
- When a static object collides, it doesn’t move
- When an object collides with a static object, it moves instead by the full MTV
Collision callbacks

- Wrap event information into a separate `Collision` info object (really a struct)
- Pass in the MTV
- Pass in which `Shapes` collided
  - Optional

```java
class CollisionComponent {
    void onCollide(Collision c);
}

class Collision {
    final GameObject other;
    final Vec2f mtv;
    final Shape thisShape;
    final Shape otherShape;
}
```
How MTVs Affect Our Engine

- Collision methods should now return Vec2d MTVs, not booleans
  - Be careful with signs and argument order
  - Especially when reversing args for double dispatch
- Be careful when calculating the MTV between concentric circles, you might end up dividing by 0
MTVs in the debugger

- Fill in the Week methods from the stencil
- Also update the week field in Display.java! (There’s a TODO comment about it)
QUESTIONS?
Lecture 3
Map Generation
MOTIVATION

Map Generation
Hand Crafted vs Procedural

Hand Crafted:
- Straightforward
- Predictable
- Time intensive

Procedural:
- Far more variety
- Can lead to janky edge cases
- Rapidly created
Procedural Generation

- Algorithmically generate your own maps
  - Experiment!
- Typically uses seeded random numbers
  - Ex. `Random r = new Random(seed);`
  - Calling `r.nextIntXXX();` some number of times will return the same sequence of numbers
  - The seed can be used to share or save the generated map
  - Used to generate seemingly-hand designed content
- Somewhat different than randomly generated
Constraint-based Generation

- Not just any random map will work
- Generated maps need to follow game-specific constraints
  - Dungeon crawlers require a path from entrance to exit
  - You might require every area of the map to be accessible
  - What looks good, what’s fun, etc
- Constraints are baked into the algorithm
Simple Generation Algorithms

- Value noise/Perlin noise
- Space partitioning
- Exploring paths (random/drunken walk)
- Lots of resources online
  - Make map generation as generic as possible
Map Generation

SPACE PARTITIONING
Space Partitioning

- Basic idea – keep splitting the map up into smaller subsections to create rooms
- Used to simulate the insides of structures
Space Partitioning

- Start with an empty rectangular grid.
• Pick a random index on which to divide the space along the x axis.
Space Partitioning

Dungeon

A

B
Space Partitioning

- Pick another index on which to divide, this time dividing along the other axis (in this case y).
- Use a different index for each split
Space Partitioning
Space Partitioning

• Keep dividing, switching between x and y until you hit some depth (3 here).
Space Partitioning

- Fill spaces with random sized boxes.
- Make sure boxes fill up more than half of the width and height of the space they occupy.
Space Partitioning

- Connect sister leaf nodes of the tree.
- If rooms don’t take up more than half their space’s width and height, you might get z-shaped hallways.
Space Partitioning

- Connect parent nodes.
Space Partitioning

• Keep on connecting up the tree.
Space Partitioning

• If the halls are too narrow, increase width of hallways to create more open space.
Space Partitioning

- Now you have your series of connected rooms!
- But there's more...
• Instead of always checking depth, have some branches of the tree stop early so you end up with more variation in room size.
Constraints

- Add a minimum width/height
- Prevents rooms from being too small and weirdly shaped
Space Partitioning

- Say you wanted to keep spawn and exit rooms far apart
- During the first split, assign one side of the tree to Spawn and the other to Exit
Space Partitioning

- At the bottom of the Spawn subtree, assign one room to spawn.
- Symmetrically for Exit subtree.
Map Generation

QUESTIONS?
Lecture 3

Level Loading
Important Map Information

- Size of map
- Locations of terrain (grass, desert, trees, etc.)
- Starting location of units, unit types, unit orientation (friendly or enemy)
- Location of sprites, on sprite sheet, for unique objects
File Parsing

- Good news: Mostly game-side
- Bad news: So many things can go wrong!
  - Map file can’t be opened
  - Map file is empty
  - Map file is a directory
  - Map file is a JPEG
  - Is a map file, but has inconsistent data
Parse Safely

• Read in a line, then parse it, repeat
  ○ At least you can report the line count where an error happened

• Recommended classes:
  ○ BufferedReader (for reading lines)
  ○ Scanner + StringReader (for parsing each line)

• Catch exceptions
  ○ Throw your own LevelParseException
  ○ Report useful debugging information

• We require that your parser never crash!
Level Loading

QUESTIONS?
Lecture 3
Tips for Wiz I
Tips for Wiz I

VIEWPORT: CENTER ON PLAYER
Bad Way

- Give the viewport a reference to the player
- On tick, update the viewport center to be the player’s transform
  - Too much game logic handled by the viewport
Better Way

- Give the player a `CenterComponent`
- Give the `CenterComponent` a reference to the `Viewport`
- Each tick, the `CenterComponent` sets the center of the `Viewport` to its object’s transform
Tips for Wiz I

JAVA TIP OF THE WEEK
Asserts

- Asserts:
  - `assert thisFunctionReturnsTrue(); // no error`
  - `assert thisFunctionReturnsFalse(); // error`
  - Can be useful for debugging
  - Particularly MTVs
Throwing Exceptions

```java
public void parse() throws LevelParseException {
    throw new LevelParseException("Uh-oh!");
}
```
Catching Exceptions

try {
    map.parse()
} catch (LevelParseException e) {
    // handle exception
} catch (IOException e2) {
    // handle exception
} finally {
    // continue assuming success
}
Lecture 3

Game Design: Controls
Why are controls important?

• In games, the player interacts with some game world
• Controls are the player’s interface for interaction with the game world
• Can make or break your game!
Good Controls: 3 Core Principles

• Good controls must be:
  ○ Intuitive
  ○ Ergonomic
  ○ Agentive
 Principle 1: Intuitive

- Intuitive controls:
  - Are easy to pick up and learn
  - Make sense for the game being played
  - Require little cognitive effort from the player
  - Follow pre-existing conventions when possible
**Principle 1: Intuitive**

- **General conventions:**

<table>
<thead>
<tr>
<th>Button</th>
<th>RTS</th>
<th>MMO</th>
<th>Shooter</th>
</tr>
</thead>
<tbody>
<tr>
<td>Left Click</td>
<td>Select units</td>
<td>Select target</td>
<td>Fire</td>
</tr>
<tr>
<td>Right Click</td>
<td>Move to/attack units</td>
<td>Attack target</td>
<td>Secondary fire/hold aim</td>
</tr>
<tr>
<td>WASD</td>
<td>Pan camera</td>
<td>Movement</td>
<td>Movement</td>
</tr>
<tr>
<td>Space</td>
<td>Focus on event</td>
<td>Jump</td>
<td>Jump</td>
</tr>
<tr>
<td>Mouse Wheel</td>
<td>Zoom camera</td>
<td>Zoom camera</td>
<td>Cycle weapons</td>
</tr>
<tr>
<td>Number keys</td>
<td>Hotkeyed units</td>
<td>Special skills</td>
<td>Select weapon</td>
</tr>
<tr>
<td>Shift (held)</td>
<td>Modify click actions</td>
<td>Use second skill set</td>
<td>Sprint</td>
</tr>
</tbody>
</table>
Principle 2: Ergonomic

• Ergonomic controls:
  ○ Have a consistent “home-base” position
    - e.g. one hand on mouse, one hand on WASD
  ○ Place most used buttons at or around home-base
  ○ Demand as little movement as possible from the player
Principle 2: Ergonomic
Principle 2: Ergonomic

Blue: Home Base
(No Movement Necessary)
Principle 2: Ergonomic

Cyan: Most Common Actions

(Minimal Finger Movement)
Principle 2: Ergonomic

Green: Common Actions
(Some Finger Movement)
Principle 2: Ergonomic

Purple: Held Actions
(Pinky movement for Shift and Ctrl, Thumb for Alt)
Principle 2: Ergonomic

Yellow: Infrequent Actions
(Some Hand Movement)

*Most players will already have learned muscle memory for ESC*
Principle 2: Ergonomic

Red: Use With Caution
(Full Hand Movement Required)
Principle 3: Agentive

- Agentive controls:
  - Give the player as much control as possible
  - Produce consistently predictable results
  - Respond as quickly as possible to player input
Principle 3: Agentive

• Common player complaints about controls that are not agentive:
  ○ “It feels laggy.”
  ○ “I can’t get it to do what I want.”
  ○ “Why doesn’t anything happen when I press the button?”
Principle 3: Agentive

• Solutions:
  ○ Make response time on every button press as quick as possible
  ○ Provide feedback when a particular action is not available
3 Principles: Recap

• Good controls are:
  ○ Intuitive  (the player can pick them up quickly)
  ○ Ergonomic  (the player can perform them easily)
  ○ Agentive  (the player feels in control)

• These need to be kept in balance!
  ○ Ergonomic controls may not be intuitive and vice versa
Common CS1950N Control Pitfalls

• Not following conventions for no discernible reason
  ○ Movement on IJKL

• Assigning actions to the keys of the first letter they start with
  ○ J for jump, L for laser, F for fire

• Using modifier keys unnecessarily
  ○ Left click to select unit, Shift + left click to move, Ctrl + left click to attack
Game Design Tips for Wiz I

• Follow the conventions of the genre!
  ○ WASD to move
  ○ Left click / arrow keys to shoot

• If you’re not sure, steal from well known games
  ○ Think from the player’s perspective!
‘Til Next Week!

• Wiz I released today!
• Sign up for design checks :)
• No class next week
• Use the time to refactor your code!