Lecture 9

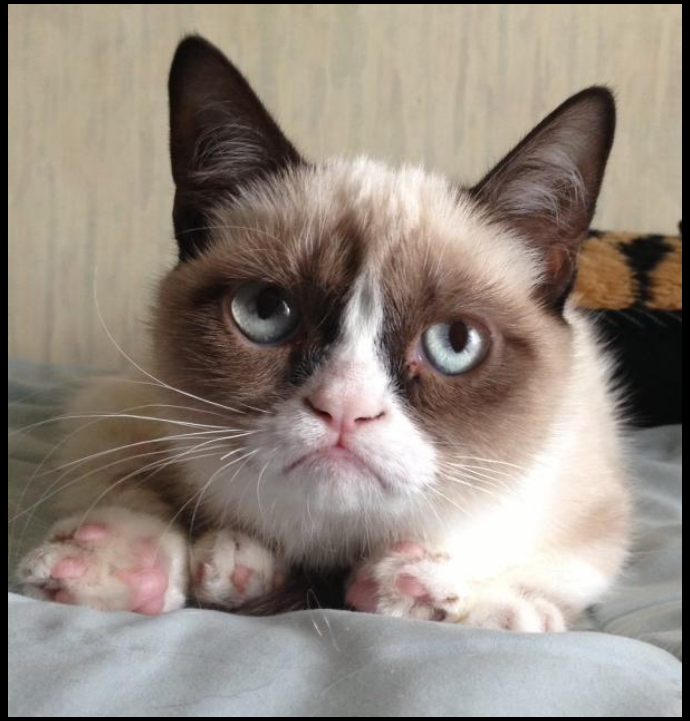
Announcements

Congrats!

- No more engine requirements!
- From now on everything you work on is designed by you

Don’t forget about old projects!

- Don’t forget, you need a working version of each project’s primary engine requirements



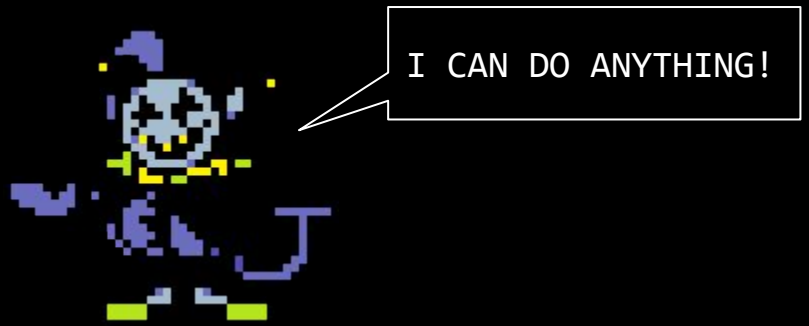
Next Week

- No Lecture
- Instead, open hours for your final and primary engine reqs

Announcements

QUESTIONS?

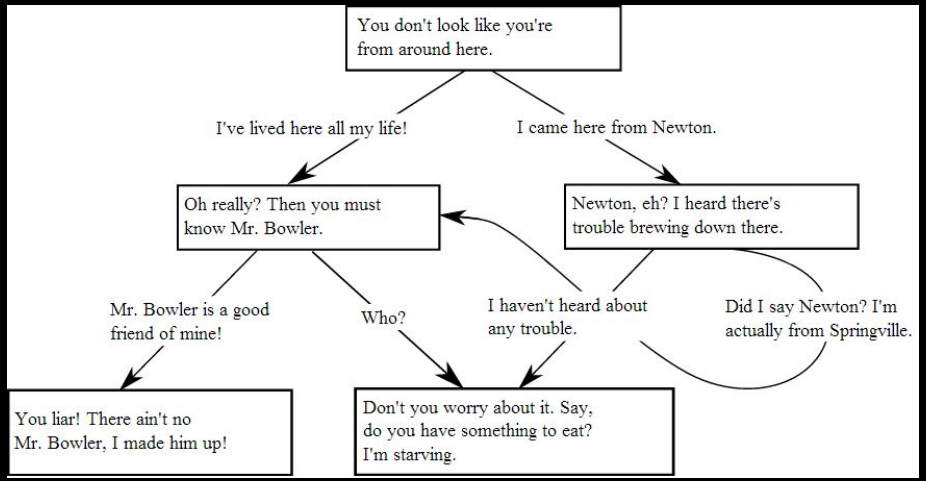
Lecture 9



Text Boxes

Dialogue

- Technically, these are UI Elements
- Requires communication between your Game World and your Scene



Shared Functionality

- All text boxes must be able to progress to another box, or back to the game

```
public class TextBox extends UIElement {
    public void advance() {
        screen.removeElement(this);
        if (this.nextBox != null) {
            screen.addElement(this.nextBox);
        } else {
            screen.setPause(false);
        }
    }
}
```

Types of Boxes

- However, some boxes might advance in different ways
  - A Single, Standalone Box
    - Says one thing and is done
  - Dialogue Option Boxes
    - Presents choices
  - Dialogue Sequence Boxes
    - Keeps yapping

Dialogue Options

- NPC Asks a question, or the world presents the player with a choice
- Requires the player to choose which Dialogue Option/Dialogue Sequence will play next (if any)
- Your UITextBoxes will need to handle input to progress

Dialogue Sequences

- Displays text, but has more to say
- Player prompts the next text box to show up (key press, click, etc.)
- Can end with an Option Box, or a Regular Box

Pause The Game

- Enemies are not fond of conversation

Text Box Pipeline

- Player interacts with the world, prompting a text box to appear
- World notifies screen
- Screen stops ticking world, presents text box
- User prompts through textboxes until reaching the end of a dialogue sequence
- Ticking is restored to the Game World

Lecture 9

Tips for Final II

Tips for Final 2

EFFECTIVE PLAYTESTING

Playtesting Reqs

- You have game requirements now!
- So now you have a game that can be played by other people
- For Final 2 onwards, each person is responsible for turning in playtesting feedback from 3 different people per a checkpoint

Finding Playtesters

- CS students are easy targets, try the sun lab or MS lab
  - Ask nicely
  - Don’t ask busy people
- Keep your audience in mind, however
  - It probably isn’t all CS students

Don't Interrupt!

- Say as little as possible
- Don't offer instructions, hints, etc.
- **Your game should speak for itself**
- The only exception is if a player gets stuck; **make a note of it** and give them a nudge to keep them moving

Keep a Log

- This is what you'll turn in!
- Make note of:
  - What the player gets stuck on
  - What the player enjoys
  - What the player ignores

Tips for Final 2

JAVA TIP OF THE WEEK

Constructors are Lame

- Superconstructors put restrictions on the order complex calculations can be performed
- Leads to duplicate code in constructors
- We'd also like to have initialization near the variable source

```
public class MyClass {
      private int i;
      private String str;

      public MyClass(int i) {
            this.i = i;
            str = "I'm number " + i;
      }

      public MyClass(OtherClass other) {
            // We can't put code before this
            this(0);
            for (...) i += 1;
            str = "I'm number " + i;
      }
}
```

Initializer blocks!

- Unlabeled blocks of code directly in the class body
- Initializer blocks solve problems with duplicated constructor code and allow initialization to be performed at the variable declaration
- Executed from top to bottom when the class is instantiated

```
public class InitBlockExample {

    public static final String s;
    static {
        String temp;
        // complicated logic here
        s = temp;
    }
}
```

Field initialization shorthand

- Field initialization is just shorthand for initializer blocks

```
public class MyClass {
    private static int i = 12;
    private String str = "";
}
```

=

```
public class MyClass {
    private static int i;
    static {
        i = 12;
    }

    private String str;
    {
        str = "";
    }
}
```

Good uses

- Immutable final collections
  - Lists, maps, etc.
- Keeping complicated initialization code near field
- Debugging!

```
public class GoodUses {
    static final Map<String, String> m;
    static {
        Map<String, String> t = /*...*/;
        // lots of puts here
        m = Collections.immutableMap(t);
    }

    int complicatedInit;
    {
        // complicated init code
    }

    GoodUses(int ap) {}
    GoodUses(int ap, String s) {}
    GoodUses() {}
}
```

Other Fun Stuff

- When you specify a main class to run, the JVM:
  - Loads class via reflection
  - Calls `main()` via reflection
- Thus, static initializers are actually run before main()
  - Can `System.exit(0)` at the end of the static initializer to exit gracefully rather than crash with `NoSuchMethodException`
- Don't ever do this

```
public class Mainless {
    static {
        String s = "Look, ma! ";
        s += "No main!";
        System.out.println(s);
        System.exit(0);
    }
}
```

Tips for Final II

QUESTIONS?

* LET'S TALK ABOUT THE FINAL!