LECTURE 1
Announcements
Collaboration policies!

Do want
Tic is over!
Tic feedback

• Use subpackages!
  – Naming a package `engine.shapes` will make a subpackage of `engine` called `shapes`

• Have a splash screen!
  – Dropping players right into the game is jarring

• Nice color schemes!
Tic feedback

• Don’t hand in your entire workspace
  – cs1971_handin should be run from your top level project directory (something like workspace/<project name>)

• If your Tic is incomplete...
  – Don’t panic! Use your standard retry
  – Send an email to your grader when you re-handin
BGD plug

• The Brown Game Developers group is meeting on Saturday, September 19th in CIT 477 (Lubrano) from 2:00PM-4:00PM
• Can be a useful source of advice, within the bounds of collab policy
• “It’s cool :D” – John Tran (class of 2014)
Tac handins...

• Handins should be all set next week (depends on Banner)
• Continue to handin via email/Google Drive until further notice
Lecture 1

Viewports
MOTIVATION
Sometimes screen space is hard

- Theoretically everything can be done in screen space
  - But some things can be very hard
  - Most notably, when the entire game isn’t visible at one time
Game space vs. Screen space

- In nearly all games, it makes sense to think of the game as existing in its own “space”
- The UI in screen space has a “view” into the game world which is just like any other UI element
Space Conversions

• Scale = pixels per game coordinate unit
  - Determines how much of your game can be seen (aka zoom)

• Game point to screen:
  1. Minus game upper left
  2. Multiply by scale
  3. Add screen upper left

• Screen point to game:
  - Do the OPPOSITE of the steps IN REVERSE

Scale: 100 px/unit

1. (1.0, 1.2)
2. (100, 120)
3. (220, 140)
Viewports

IMPLEMENTATION
Implementing viewports

1. Set the clip \(g.clipRect()\)
   a. You will draw out of bounds otherwise
2. Set the “transform”
3. Draw the game-space in its own coordinates
4. Restore the “transform”
5. Restore the clip (if you set it)
The “transform”

• This could be implemented as follows:
  – Create a wrapper for Graphics2D with a method like setViewport()
  – When drawing a viewport, toggle a flag inside your Graphics2D wrapper that you need to transform game- >screen coordinates whenever a shape is drawn
  – Do separate game object transforms in each object’s own onDraw() call
  – Unmark your wrapper when finished drawing the viewport

• That’s a lot of math to do, and a lot could go wrong…
Affine Transforms

• Java’s `AffineTransform` keeps track of geometric transformations for drawing
  – Uses concepts from linear algebra
  – Haven’t taken it? No problem!
• A `Graphics2D` maintains an internal transform
  – Use `setTransform()` and `getTransform()` to modify the current `AffineTransform`
  – Check out `translate()`, `scale()`, and others
  – Works just like old OpenGL (though you’ll have to write a `Graphics2D` wrapper if you want `push()` and `pop()`)
QUESTIONS?
Warnings!

• Viewport is essential to the rest of the class – every assignment from here on will depend on using your viewport!
  – Hard to test your game if you can’t display it correctly
  – Design well
  – Test thoroughly
  – Don’t push bugs until later weeks

• The TA staff STRONGLY recommends the use of AffineTransforms over custom methods
  – If you do not feel comfortable since you haven’t taken Linear Algebra, come to hours!
Lecture 1

Content Management I
WHAT IS CONTENT?
Content

• Types of content
  – Sprites, images, textures
  – Music, sound effects
  – Level/map files
  – Scripts
  – Dialogue

• A single logical piece of content is called an “asset”
WHY NOT HARDCODE ASSETS?
Hardcoding

- Extreme executable bloat
- Large games cannot fit in memory
- Have to recompile entire program every time an asset changes
Solution: break into files

• Engine/game can load and unload assets as necessary or desired
• Non-programmers don’t need to compile
  – Level designers only need to touch map files
  – Artists only need to touch image files
  – Programmers compile builds for everyone else
• More maintainable in general
LEVELS – MAP FILES
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: 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
• We’ll show you how to handle this in a bit
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!
LEVELS — MAP GENERATION
Procedural Generation

- Algorithmically generate your own maps
- Game side - 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 methodically to generate seemingly-hand designed content
- Much different than randomly generated!
Constraint-based Generation

- Not just any random map will work
- Generated maps need to follow game-specific constraints
  - A dungeon crawler might require a path from entrance to exit
  - An RTS might require every area of the map accessible
  - Puzzles must be solvable
- Design your generation algorithm around your constraints
- Then consider soft constraints
  - What looks good, what’s fun, etc
Simple Generation Algorithms

- Perlin noise
- Spatial partitioning
- Exploring paths (random/drunken walk)
- Lots of resources online
  - Can you make your generation engine specific?
Perlin Noise

Named for its creator, this guy, Ken Perlin.

It’s a great way to make smooth, natural noise which can be used to create terrain, cloud patterns, wood grain, and more!

But you’ll probably use it for terrain...
We will be implementing a slightly more intuitive version called Value Noise.
What is noise?

• Randomness
• e.g. From 0 to 14 take a random number between 0 and 1
• By itself, it is jagged and not useful
Wave Functions

![Wave Function Diagram]

Amplitude

Wavelength

frequency = \frac{1}{\text{wavelength}}
Wave Functions

\[ \sin(x) \]
Wave Functions
Wave Functions
Wave Functions

• For \( \sin(x) \) we can change the amplitude and frequency using:

\[
\text{amplitude} \times \sin(\text{frequency} \times x)
\]
Adding Wave Functions
Adding Wave Functions
Adding Wave Functions

\[ \sin(x) + \frac{1}{2}\sin(2x) \]

\[ \frac{1}{4}\sin(4x) \]
Adding Wave Functions

\[
\sin(x) + \frac{1}{2} \sin(2x)
\]

\[
\frac{1}{4} \sin(4x)
\]

\[
\sin(x) + \frac{1}{2} \sin(2x) + \frac{1}{4} \sin(4x)
\]
### Adding Noise Functions

<table>
<thead>
<tr>
<th>Freq.</th>
<th>1</th>
<th>2</th>
<th>4</th>
<th>8</th>
<th>result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Amp.</td>
<td>1</td>
<td>$\frac{1}{2}$</td>
<td>$\frac{1}{4}$</td>
<td>$\frac{1}{8}$</td>
<td></td>
</tr>
</tbody>
</table>

- An frequency of 1 means that there is a spike at every single tile, so don’t use much bigger freq’s in your algorithm!
Value Noise

• Great, so we can add noise functions
• Now we need a smooth noise function that will always return the same value given the same seed and same \((x,y)\) pair.
• Let’s focus on the smooth part first, and the randomness given a seed/\((x,y)\) part second

Coherent noise, more useful...

Non-coherent noise, less useful...
Smooth Noise

- Most interpolation functions take three arguments.
  - \(a\) and \(b\), the value to interpolate between.
  - \(x\), a value between 0 and 1.
    - When \(x\) is 0, function returns \(a\)
    - When \(x\) is 1, function returns \(b\)
Smooth Noise

• Option 1: linearly interpolate between points
• For any $a$, $b$, and $x$ on this graph:
  \[ v = a \times (1 - x) + (b \times x) \]
• This doesn’t look so great
Smooth Noise

• Better option: cosine interpolation

\[ f = (1 - \cos(x \times \pi)) \times .5 \]

\[ v = a \times (1 - f) + (b \times f) \]

• Looks much better

• Slightly slower, but worth it for the results
So what do we know?

- We know how to smooth our noise function
- We know how to add big and small noise functions together to get more interesting noise
- So now we just need a noise function
A Good Noise Function

• What does our noise function need?
  – Given an \((x, y)\) pair and a seed, returns the same value between 0 and 1 every time

• Random.setSeed() only takes a single seed as an argument
A Good Noise Function

• No problem, I’ll just define some function that takes in x, y, and my seed and use that to seed `Random.nextFloat()`

• Good try, but then every point along that function will have the same value, and it will be obvious when looking down on your map
A Good Noise Function

• TA suggestion: use the Vec2f.hashcode() method
  – Returns a single integer that is unique to each pair
  – Will return the same integer every time

• Add this number to your seed and use this result to seed your Random.nextFloat() call
A More Detailed Explanation

• What follows is a lot of pseudocode that contains concepts that we haven’t discussed
  – Persistence, octaves, etc.

• Use this website as a reference:
  – http://freespace.virgin.net/hugo.elias/models/m_perlin.htm
  – It’s actually about Value Noise, not Perlin Noise as it claims
Value Noise

class NoiseGenerator {
    int _baseSeed;
    int _currentSeed;
    Random _rand;

    // feel free to make your own noise function
    private float noise(Vec2i vec) { … }
    private float smoothNoise(Vec2i vec) { … }
    private float interpolatedNoise(Vec2f vec) { … }
    public float valueNoise(Vec2f vec, float freq,
                            float persistence, int num_octaves) { … }
}
Value Noise

// returns a weighted average of the 9 points around the Vec2i v
float smoothNoise(Vec2i vec){
    // four corners, each multiplied by 1/16
    corners = ( noise(vec.x-1, vec.y-1) + noise(vec.x+1, vec.y-1) +
               noise(vec.x-1, vec.y+1) + noise(vec.x+1, vec.y+1) ) / 16
    // four sides, each multiplied by 1/8
    sides = ( noise(vec.x-1, vec.y) + noise(vec.x+1, vec.y) +
              noise(vec.x, vec.y-1) + noise(vec.x, vec.y+1) ) / 8
    // center, multiplied by 1/4
    center = noise(vec.x, vec.y) / 4
    return center + sides + corners
}
// returns an value interpolated between the four corners surrounding the Vec2f v
float interpolatedNoise(Vec2f vec){
    integer_x = Math.floor(vec.x)
    fractional_x = vec.x - integer_x
    integer_y = Math.floor(vec.y)
    fractional_y = vec.y - integer_y

    // the four integer corners surrounding the float (x,y) pair
    v1 = smoothedNoise(integer_x, integer_y)
    v2 = smoothedNoise(integer_x + 1, integer_y)
    v3 = smoothedNoise(integer_x, integer_y + 1)
    v4 = smoothedNoise(integer_x + 1, integer_y + 1)

    i1 = interpolate(v1, v2, fractional_x)
    i2 = interpolate(v3, v4, fractional_x)

    return interpolate(i1, i2, fractional_y)
}
// returns a value between 0 and 1
// freq is the initial frequency of the largest “hill”
// persistence is between 0 and .5, determining how large each amplitude will be
// in relation to the previous one
float valueNoise(Vec2f vec, float freq, float persistence, int num_octaves) {
    total = 0
    amp = .5
    for (int i = 0; i < num_octaves; i++) {
        _currentSeed = _baseSeed + i // so we use a modified seed for each octave
        total = total + interpolatedNoise(vec.x * freq, vec.y * freq) * amp
        amp = amp * persistence
        freq = freq * 2
    }
    return total
}
So what now?

- We have a 2D array of cells that each have a “height” which is between 0 and 1
- Now we can set some threshold for each “layer”
  - 0 – 0.2 is water
  - 0.2 – 0.6 is sand
  - 0.6 – 0.9 is dirt
  - 0.9 – 1.0 is forest
- Each layer can have different properties!
Space Partitioning

- Basic idea – keep splitting the map up into smaller subsections to create rooms
- Not as good as Perlin noise for creating mountainous terrain, used to simulate the insides of structures
Space Partitioning

• Start with an empty rectangular grid.
Space Partitioning

- Pick a random index on which to divide the space along the x axis.
Space Partitioning
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 or height of the space they occupy.
Space Partitioning

• Connect sister leaf nodes of the tree.
• Rooms must either take up more than half their space's width and height or will have 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...
Space Partitioning

- Instead of naively checking depth, have some branches of the tree stop early so you end up with more variation in room size.
Space Partitioning

- To prevent rooms from being too small and weirdly shaped, keep your dividing index with a certain range.
Space Partitioning

- Teams!
- When you do your first split, assign one side of the tree to Team A and the other to Team B.
Space Partitioning

- At the bottom of the tree, assigning one room to each team as a base will ensure that the bases are on different halves of the map and no one is locked in.
QUESTIONS?
Viewports

• Zooming is multiplicative, not additive
  – Rolling mouse wheel should * or / the scale factor

• Some zoom levels don’t make sense
  – Set zoom in/out limits!

• Most maps don’t go on forever
  – Set pan limits!
Content Location

• External content should be referenced relative to your top level project directory
  - resources/maps/myMap.txt

• Anything like below WILL NOT WORK!
  - ~/course/cs1971/tac2/sprites.png
  - /home/student/course/cs1971/tac2/sprites.png
Tips for Tac 1

JAVA TIP OF THE WEEK
Keep asset integrity

• Information is being read from outside sources potentially not created by you.
• It is good practice to ensure that content requirements are being met.
Exceptions!

• Exceptions allow you to provide feedback for content formatting failures

• Lets say that you are reading your map files, and want the first line to always say BEGIN.

• If it doesn’t then you can throw an exception, letting the designer know where the error happened and in what file.
Define your exception

```java
public class MapFileException extends Exception {
    public MapFileException(int line, String file) {
        super("There was a problem with the map file " + file + " on line " + line);
    }
}
```
//read first line from myMap
Scanner s = new Scanner(new File("myMap.txt"));
String first = s.next();
if (!first.equals("BEGIN")) {
    throw new MapFileException(1, "myMap.txt");
}
More Exceptions!

• Exceptions also allow (read: force) you to think about exceptional runtime circumstances
• Too often we see this in handins:
• What problems does this pose?

```java
try {
    // some code
} catch (IOException e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
}

// business as usual
```
Right ways to do it

• Catch and rethrow
  – Logging, cleaning up state (though finally exists)
• Wrap in higher-level exception
  – Map reader shouldn’t throw NumberFormatException, should throw MapParseException with original exception as the cause
What can be thrown?

- **Throwable**
  - Base class, catch-all
  - Never instantiated directly
- **Error**
  - Thrown by VM, unchecked
- **Exception**
  - Normal exceptions, thrown by and handled by programmers
- **RuntimeException**
  - Indicate coding error, thrown by programmers, unchecked
Common unchecked exceptions

**RuntimeExceptions**
- NullPointerException
- IllegalArgumentException
- IllegalStateException
- ConcurrentModificationException
- IndexOutOfBoundsException
- SecurityException
- UnsupportedOperationException

**Errors**
- StackOverflowError
- OutOfMemoryError
- AssertionError
- UnsatisfiedLinkError
- NoClassDefFoundError
- NoSuchMethodError
- InternalError
Tips for Tac I

QUESTIONS?
GAME DESIGN 0

Introduction
Video game creation is an art, a science, and a feat of engineering.

-Andy van Dam
Supplement to 2D Games

• More “high-level” concepts, less technical
• Visualize game creation from a more artistic perspective
• Inspire ideas and concepts for the final project
• Perhaps influence coding style and engine design
Ultimate Goal

Help create better Final Projects
Topics Covered

• History + Genres (Today!)
• Intro to Game Design
• Controls
• Juice
• Difficulty
• Level Mechanics
• Story
• Aesthetics
GAME DESIGN 0
History + Genres
Overview

• A brief look at the history of video games
• An examination of genres in 2D games
• The ultimate goal: familiarization with video games 😊
70’s: Arcade

• *Pong*, the first commercially successful video game (1972)
• Atari 2600, the first major home console (1977)
  – 8 bit CPU, 128 bytes of RAM, 4 KB game cartridges
• Golden Age of Arcade (1978-86)
  – Began with *Space Invaders*
70’s: Arcade

• Common themes:
  – Limited story and graphics
  – Heavy use of score as a gameplay element
The Video Game Crash of 1983
80’s: Home Consoles

• Nintendo Entertainment System (1985)
  – 8 bit CPU, 2KB of expandable RAM, 8KB–1MB game cartridges
• Super Mario Bros. (1985)
• The Legend of Zelda (1986)
• Metal Gear (1987)
80’s: Home Consoles

- Common themes:
  - More variety due to removal of technological limitations
  - High difficulty for replay value
  - Side-scrollers and platformers
90’s: Large Scale Gaming

• Game Boy (1989)
  – First “true” mobile game console

• Rise of PC Gaming
  – *Dune II* (1992) set the standard for the RTS genre

• 3D Graphics
  – *Super Mario 64* (1996) set the standard for 3D platformers
90’s: Large Scale Gaming

• Common themes:
  – Development by large game studios
  – Shift to 3D games on home consoles
  – Handheld becomes the domain of 2D games
2000’s: Innovation and Expansion

• Rise of online gaming
• Expansion of casual audience
  – Nintendo Wii (2006)
• Beginning of indie and modding scenes
  – Counter-Strike (1999)
Genres

• What is a genre?
  – In video games: categorization based on common gameplay elements

• More like labels than hierarchical categories

• Each genre has countless subgenres
  – Example: Metroidvania is an action-adventure subgenre of Platformers that focuses on open world exploration where the player obtains upgrades to gain access to more of the world
Camera genres in 2D

Top-Down

Side-Scroller
Big gameplay genres in 2D

- Puzzle
- Platforming
- Shooters
- Fighting
- Role-playing games
- Strategy
Puzzle games

- Emphasis is on solving some kind of puzzle
- Minesweeper (1960’s)
- Tetris (1984)
- Angry Birds (2009)
Platformers

• Player must jump between platforms to progress

• Super Mario Bros. (1995)

• Today, often mixed with puzzle elements
  – Braid (2008)
Shooters

- Gameplay involves shooting some sort of weapon to defeat enemies
- Incredibly popular!
- *Galaga* (1981)
- *Contra* (1987)
Fighting games

- Player controls an avatar and engages in close-quarters combat with opponents
- **Street Fighter II** (1991)
- **Mortal Kombat** (1992)
  - Fun fact: responsible for the ESRB!
- **Super Smash Bros.** (1999)
Role playing games (RPG’s)

- Player assumes the role of a character
- Typically features character stats and leveling up
- *Final Fantasy* (1987)
- *Pokemon Blue* (1997)
- *Diablo* (1997)
Real-Time Strategy

• Player controls units in battle against AI or other players
• Usually involves an economy system for constructing a base and units
• Modern concept defined by *Dune II*, built upon by Blizzard’s *Warcraft: Orcs & Humans*
• Some debates that this genre has remained largely static since establishment
Warcraft: Orcs and Humans (1994)
Starcraft (1998)
Multiplayer Online Battle Arena

• Started by Warcraft III mod
  – Dota
  – League of Legends

• Combines elements of RTS and RPG

• Main goal is to destroy enemy “headquarters”

• Strong competitive scene
Turn Based Strategy

• Player controls a team of units with the objective of defeating some enemy team
• Each unit takes turns performing actions: moving, attacking, etc.
• Typically grid-constrained
Final Fantasy Tactics (1997)
XCOM: Enemy Unknown (2012)
Things to see

• http://www.dmoz.org/Games/Video_Games/History/
Game Design: Genres

QUESTIONS?
Tic Playtesting!

Yay!