Lecture 5

Announcements
Nin: Your last (solo) project

- Platformer game
  - Physics
  - More physics
  - Externalizing game logic
QUESTIONS?
Lecture 5

Final Project Pitches
This Week

- Today in class: pitch final project ideas!
- Afterwards: talk to your classmates on Slack! Just use #general.
- Form groups (2-4 people per group)
  - Reach out to the TAs about working alone
- By Thursday midnight, fill out this [Google Form](#) with your group members
  - Also on the website
- We’ll reach out to you about scheduling a design check
Final Projects

QUESTIONS?
Lecture 5
Collection Detection II

Technically, you have access to the outside.
Polygons

- Lots of places that could collide (overlap)
- Irregular shapes
- Can’t test every screen point
  - What if both polygons are huge?
  - Can get false positives if pixels are close together
Polygon Definitions

- **Polygon**
  - Bounded by 3 or more straight line segments (edges)
  - Edges meet at vertices

- **Convex Polygon**
  - Every interior angle < 180°
  - Any line through the shape crosses only twice
  - Boundary never crosses itself

- **Concave Polygon**
  - Has an interior angle > 180°
  - There exists a line through the shape that crosses more than twice
Point in Polygon

- Think of the border of a polygon as a path of vectors
- Counterclockwise order!!!
- For convex polygons, points in the interior will be on the same side of all the vectors
Point in Polygon

- To determine what side of a vector \( \mathbf{v} \) a point is on:
  - Draw another vector \( \mathbf{p} \) from the base of the vector to that point
  - Take cross product \( \mathbf{v} \times \mathbf{p} \)
  - If result is negative, it’s on the left

[Diagram showing a vector with points of interest and cross-product signs showing direction]
Point in Polygon Algorithm

1. Iterate over the edges (counterclockwise)
2. Construct a vector $\mathbf{v}$ along the edge
3. Construct a vector $\mathbf{p}$ to the point
4. Take the cross-product $\mathbf{v} \times \mathbf{p}$
5. If all cross-products are negative, point is inside
6. If any cross-product is positive, it’s outside
Recap: Projections

- 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
Shape-Polygon

- What about Circle-Polygon, AAB-Polygon, and Polygon-Polygon collisions?
- Can all be handled with the same algorithm
- Based on the Separating Axis Theorem
Projection

- Projection axis not necessarily horizontal or vertical
- Dashed line is the axis
- Red and blue lines are projections of the shapes
Intersection

- Two shapes intersect when ALL possible projections overlap
  - No matter what direction you look from, you can’t see between the shapes
- But there are infinitely many projections!
  - We’ll show we only need to project against each shape’s edge’s perpendicular.
Separating Axis Theorem

- If two convex shapes are not overlapping, there exists an axis for which the projection of the shapes will not overlap.
- If we can find an axis along which the projection of the two shapes does not overlap, then the shapes aren’t colliding.
Caveat: Only Convex Shapes

- SAT only applies to convex shapes
  - Can’t draw a line between concave shapes, therefore no separating axis
- In this class, you don’t need to support concave polygons
- You can make compound shapes that are concave, but each component is convex
Separating Axis

- Consider an axis on which projections do not overlap
- There’s a line perpendicular to it that goes between the shapes
- This is the line of sight
Separating Axis

- Line of sight = a line that can be drawn between two separate shapes
- Separating Axis = axis perpendicular to that line, onto which shapes are projected
Finding Separating Axes

- If shapes are very close to colliding, a line parallel to an edge will always get through.
- Minimal separating axis is always perpendicular to a shape’s edge.
Finding Separating Axes

- Lines perpendicular to each shape edge are all the separating axes you need
  - Sometimes called “edge normals”
- Consider each edge a vector, and take the perpendicular
- Each green arrow here represents a separating axis
General SAT Algorithm

1. Create a vector for each edge of each shape
2. Take the perpendicular vector to get a separating axis
3. For each axis, project both shapes onto it
4. If there exists an axis without an overlap, there is no collision
5. If the projections overlap on every axis, the shapes are colliding
Special Cases: Circles

- Take the vector from the center to the closest vertex of the polygon
  - A line of sight would have to pass between them
- No perpendicular – this vector is already a separating axis
Special Cases: AABs

- Four axes perpendicular to the edges of an AAB
- Two of them are parallel, why test four?
- For an AAB, separating axes are just x and y axes
Implementation Notes

- To construct vectors for polygon edges, iterate around points counter-clockwise
- Direction of the vector matters!
  - There are two kinds of perpendicular: (-y, x) and (y, -x)
  - Make sure you’re consistent
Implementation Notes

● Remember to check BOTH polygons’ separating axes
  ○ Otherwise false positives

● Checking for overlap
  ○ $\text{min}_1 \leq \text{max}_2 \land \text{min}_2 \leq \text{max}_1$
  ○ Should be the same from your AAB
    Interval overlaps
QUESTIONS?
Collision Detection II

POLYGON MTV
Recap: MTV in one dimension

- In 1D, convex shapes are line segments (intervals)
- These have a 1D MTV
  - Similar to overlap
  - But it has a sign
- Write a method that computes this
- Use it to find shapes’ MTV
Computing MTV

1. For each axis, project both shapes onto that axis, and find the 1D MTV of their projections
   ○ Make sure to normalize the axis before projection
2. Find the axis with the minimum 1D MTV
3. The 2D MTV is the minimum 1D MTV times that (normalized) axis

Note: normalizing and projection are in the Vec2d source code!
Computing intervals’ MTV

// use Double instead of double so we can return null
Double intervalMTV(Interval a, Interval b):
    Double aRight = b.max - a.min
    Double aLeft  = a.max - b.min
    if aLeft < 0 || aRight < 0:
        return null
    if aRight < aLeft:
        return aRight
    else:
        return -aLeft
Computing polygons’ MTV

```python
Vec2d shapeMTV(Shape a, Shape b):
    Double minMagnitude = +infinity
    Vec2d mtv = null
    for Vec2d axis in allAxes:
        axis = norm(axis)
        Double mtv1d = intervalMTV(a.proj(axis), b.proj(axis))
        if mtv1d is null:
            return null
        if abs(mtv1d) < minMagnitude:
            minMagnitude = abs(mtv1d)
            mtv = axis.smult(mtv1d)
    return mtv
```

Recap: Special Cases

- **Circle vs Polygon**
  - Use these axes:
    - Polygon’s edge normals
    - Vector from circle center to closest vertex

- **AAB vs Polygon**
  - Use these axes:
    - Polygon’s edge normals
    - x axis and y axis
QUESTIONS?
An interesting aside on SAT

- The SAT is actually N-dimensional...
- To check two N-d convex shapes:
  - Find separating axes (N-1 dimensions)
  - Project all points onto this hyperplane
  - Use convex hull to get a (N-1)-D polygon
  - Run (N-1)-D SAT on these
  - Two N-d shapes overlap if all (N-1)-d projections overlap
Lecture 5

Physics II

![Diagram of a chicken with forces labeled](image.png)
Physics II

VELOCITY & ACCELERATION
Velocity

- Rate at which position changes
  - $\Delta x / \Delta t$
- Can’t increment position each tick
  - Frame rates vary
- Multiply by time elapsed
  - pos += vel * t
Acceleration

- The rate that velocity changes
  - \( \Delta v / \Delta t \)
- Useful for gravity, springs, wind, etc.
  - `vel += acc * t`
- Fun fact: the next derivatives of position are called jerk, snap, crackle, and pop
Which order to update?

**Position first (Euler)**

\[
\text{pos} += \text{vel} \times \text{time} \\
\text{vel} += \text{acc} \times \text{time}
\]

**Velocity first (Symplectic Euler)**

\[
\text{vel} += \text{acc} \times \text{time} \\
\text{pos} += \text{vel} \times \text{time}
\]

- more stable, use this
Changing velocity for collision

- Just reverse the object’s velocity?
  - \( \text{vel} = -\text{vel} \)
- Reverse the y component?
  - \( \text{vel.y} = -\text{vel.y} \)
  - Doesn’t work if we bounce against a wall!
Two moving-object collisions

- Reverse both velocities?
- Doesn’t always work
- **Apply equal and opposite impulses**
  - An impulse is an instantaneous force (also known as the change in momentum)
  - Objects will change their own velocities based on mass, so heavier objects will see less acceleration
  - Simulate real-world physics
Units (throwback to AP Physics C or IB HL Physics)

Without mass

- position \( \vec{x} \)  
- velocity \( \vec{v} = \Delta \vec{x}/\Delta t \)  
- acceleration \( \vec{a} = \Delta \vec{v}/\Delta t \)  
- (no equivalent)

With mass

- (no equivalent)
- momentum \( \vec{p} = m\vec{v} \)
- force \( \vec{F} = \Delta \vec{p}/\Delta t \)
- impulse \( \Delta \vec{p} \)
Implementing force and impulse

- applyForce() accumulates force
- applyImpulse() accumulates impulse
- onTick() applies force and impulse, clearing them for next frame
- Static (immovable) objects shouldn’t be affected by applyForce() and applyImpulse()

```java
class PhysicsComponent {
    double mass;
    Vec2d pos, vel;
    Vec2d impulse, force;

    void applyForce(Vec2d f) {
        force += f;
    }

    void applyImpulse(Vec2d p) {
        impulse += p;
    }

    void onTick(float t) {
        vel += t*force/mass + impulse/mass;
        pos += t*vel;
        force = impulse = 0;
    }
}
```
Building a gravity component

- Use `applyForce()` to apply a gravitational force
  - Newton’s second law: $F = ma$
  - $F_g = \text{mass} \times \text{gravitational constant}$
- Don’t apply as an impulse; otherwise, fall speed will be dependent on the frame rate
- Play around with g: if things are falling too slowly, crank it up
  - Watch videos of other platforming games to get a sense of natural movement
Impulse collision response

- Translate objects out of collision
  - Each by MTV/2
  - Or proportional to mass in direction of MTV

- Apply some impulse proportional to MTV to each object
  - Details in next section
Physics II

QUESTIONS?
Note about Velocity

- When working with collisions, we only care about the velocity in the direction of the collisions.
- Your engine is 2D, so your velocities are 2D vectors.
- For all the math in the next slides, we’ll need our velocity as a scalar.
  - We only want the component parallel to the MTV.
- To do this, take the dot product of the velocity and the normalized MTV.
Physics II

RESTITUTION
Restitution

- Property of physical entities
- Amount of energy lost in collision
- Value between 0 and 1
  - 0 is perfectly inelastic (objects stick together)
  - 1 is perfectly elastic (kinetic energy entirely preserved, e.g. pool balls)
- The coefficient of restitution (COR) between two entities is the geometric mean of their restitutions: $\sqrt{r_1 r_2}$
Correct Collisions

- How do we find the physically correct collision response?
- i.e. given $u_a$ and $u_b$, what are $v_a$ and $v_b$?
  - $u$: initial velocity
  - $v$: final velocity
  - $a$ and $b$: objects
- Use physical definition of the COR:
  - $\frac{v_b - v_a}{u_a - u_b}$
Final Velocities: finding $v_a$ and $v_b$

- Conservation of momentum:
  \[ m_a u_a + m_b u_b = m_a v_a + m_b v_b \]

- Solving for $v_a$
  \[ v_a = \frac{m_a u_a + m_b u_b - m_b v_b}{m_a} \]

- Substitute COR equation for $v_b$
  \[ \text{COR} \cdot (u_a - u_b) + v_a = v_b \]
  \[ v_a = \frac{m_a u_a + m_b u_b - m_b (\text{COR} \cdot (u_a - u_b) + v_a)}{m_a} \]
Final Velocities: finding $v_a$ and $v_b$

- Solve again for $v_a$:
  
  \[
  v_a = \frac{m_a u_a + m_b u_b - m_b \text{COR} \cdot (u_a - u_b) - m_b v_a}{m_a}.
  \]

  \[
  v_a \left(1 + \frac{m_b}{m_a}\right) = \frac{m_a u_a + m_b u_b - m_b \text{COR} \cdot (u_a - u_b)}{m_a}.
  \]

  \[
  v_a = \frac{m_a u_a + m_b u_b - m_b \text{COR} \cdot (u_a - u_b)}{m_a + m_b} \cdot \frac{m_a}{m_a + m_b}.
  \]

  \[
  v_a = \frac{m_a u_a + m_b u_b - m_b \text{COR} \cdot (u_a - u_b)}{m_a + m_b}
  \]

  \[
  v_a = \frac{m_a u_a + m_b u_b + m_b \text{COR} \cdot (u_b - u_a)}{m_a + m_b}.
  \]
Final Velocities: finding $v_a$ and $v_b$

- Similarly for $v_b$
  $v_b = \frac{m_a u_a + m_b u_b + m_a \cdot \text{COR} \cdot (u_a - u_b)}{m_a + m_b}$
Final Velocities

- Can’t just set velocities directly to $v_a$ and $v_b$!
  - Might interfere with other collisions (that happen later on the same tick)
- Use impulse instead
  - Instead of calculating $v_a$ directly, we just find the impulse that needs to be applied
Velocity Difference

- Impulse causes a change in velocity: we want to change from $u$ to $v$
  \[ I_a = m_a(v_a - u_a) \]

- To find the change in velocity, use our previous equation for $v_a$
  \[ v_a - u_a = \frac{m_a u_a + m_b u_b + m_b \text{COR} \cdot (u_b - u_a) - u_a (m_a + m_b)}{m_a + m_b} \]
  \[ v_a - u_a = \frac{m_b u_b + m_b \text{COR} \cdot (u_b - u_a) - u_a m_b}{m_a + m_b} \]
  \[ v_a - u_a = \frac{m_b (u_b + \text{COR} \cdot (u_b - u_a) - u_a)}{m_a + m_b} \]
  \[ v_a - u_a = \frac{m_b (u_b - u_a) (1 + \text{COR})}{m_a + m_b} \]
Final Impulse

- Multiply the mass back into our impulse equation:
  \[ I_a = \frac{m_am_b(u_b-u_a)(1+\text{COR})}{m_a+m_b} \]

- Same process for \( v_b \):
  \[ I_b = \frac{m_am_b(u_a-u_b)(1+\text{COR})}{m_a+m_b} \]

- Note that neither equation depends on final velocities; we can calculate them directly from the **known values** mass, initial velocity, and \( \text{COR} \sqrt{\frac{r_1}{r_2}} \)

- Implement these equations and apply these impulses upon collision!
Static Shapes

- If \( a \) is static, then you can treat it as if it had infinite mass.
- Take the limit of \( I_b \) as the mass of \( a \) goes to infinity:
  \[
  I_b = m_b (u_a - u_b) (1 + \text{COR})
  \]
- Vice-versa if \( b \) is static.
- You should special-case this.
Putting it all together

Physically correct collision response:

1. Calculate COR with the restitution properties of the shapes
2. Project velocities onto MTV (which become $u_a$ and $u_b$)
3. Apply impulse formula to calculate impulses
4. Apply corresponding impulse to each shape’s PhysicsComponent
Reminders

● Directions matter! Be consistent about which way is positive/negative.
● Our `projectOnto()` method gives a vector
● Our `dot()` method gives a scalar
● For physics equations, you want a scalar velocity, so use `dot()`
● To use `dot()` for projection, you need to give it a normalized vector
QUESTIONS?
Lecture 5
Tips for Nin I
More Accurate Sprites

- Use polygons to give units a “bounding shape” that mimics the visible outline of sprite
Behavior Trees/GOAP

- Give the enemies an AI instead of hardcoded behavior
- Keep track of the player’s current level/difficulty
Axis / Range Classes

- Projection is tedious, can be messy
- An Axis class is a good idea
  - project methods for each shape
- May also want an Interval or Projection class
- Intervals should be able to compare themselves to other Intervals

```java
public class Axis {
    public Axis(Vec2f direction) {...}
    public Interval project(Circle c) {...}
    public Interval project(AAB a) {...}
    public Interval project(Polygon p) {...}
    public Interval project(Compound c) {...}
}
```
Fixed Timestep

- Collisions can break if you have a really long tick
- This might happen, so consider:
  - Give the physics world some constant time on each tick
  - Tick as many times as possible on each game tick
  - Called separately from tick() and lateTick()
Known Bad Cases

- Some things you know will make your code blow up
- Floating point values should never be NaN
- Vectors should never try to divide by zero
- Make sure there are never any NaNs
- Polygons should always be convex
- NaNs will ruin your life
- Vectors have isNaN() methods
- Use assert liberally
Tips for Nin I

QUESTIONS?
Lecture 5

Difficulty
What is difficulty?

- Games are “problem-solving activities approached with a playful attitude”
- The difficulty of solving the problems in the game determines how hard it is
Why does it matter?

- Too easy: your players get bored and quit
- Too hard: your players get frustrated and quit
Components of difficulty

- **Learning curve**
  - Depends on complexity of controls and mechanics
- **Punishment for failure**
- **Difficulty of sub problems**
  - How do I defeat this enemy?
  - How do I clear a line in Tetris with these blocks?
What makes a problem difficult?

- Clarity of problem statement
- Clarity of having reached a solution
- Transparency of solution
- Difficulty of executing the solution
Fair vs. Unfair difficulty

In a fair game…

- The player is responsible for failure
- The player clearly understands the objective
- The player knows what they are capable of doing

In an unfair game…

- Random factors determine failure
- The player doesn’t know they’re trying to do
- The player doesn’t know what they can do
Gauging the difficulty of your game

- As the programmer and designer, you know your game inside and out
- General rule: actual difficulty is always at least one step up than what you think it is
- Playtesting is the best way to test how hard your game is
- No playtesters? Play your game with various levels of effort (from lazy to tryhard)
- Remember: TAs need to be able to beat your game in order to grade it
Adjusting difficulty

- Play with the following:
- Learning curve
  - How hard is it to learn to play the game?
- Degree of punishment for failure
  - How much is the player set back for messing up?
- Difficulty of subproblems
  - How hard is it to do things in the game?
- Number of distinct subproblems
  - How many different problems need to be solved?
- Clarity of problems (only if you know what you’re doing!)
  - How well does the player know what they’re supposed to be doing?
QUESTIONS?
Lecture 5
Final Project Pitches
Daniel Spencer

- **A roguelike bullet hell based on Enter the Gungeon**
  - Variety of challenging enemies with unique AIs and visuals, including bosses
  - Unique weaponry
  - Player improves over time through skill (rather than improvements to character stats etc.)
  - Nice aesthetic style
  - Fluid world traversal and rewarding gunplay

- **Potential features**
  - Lighting - support for game world lights such as wall and ceiling lights
  - Sprite rigging - extend animations to enable sprites to be linked/pinned together in the game world. For example, guns should remain fixed to the characters’ hands throughout all movements.
  - Fixed rooms - boss rooms or treasure rooms, for example, can be injected into the map generation algorithm (e.g. force the algorithm to include some number of these rooms).
  - Custom key binding - this may be more on the game side
  - Co-op - stretch goal
Timothy Wang

● A Puzzle RPG Game
  ○ Personally, I love the idea of progression with a character, and dungeon enemies are a great aesthetic. I really liked games like Path of Exile where you build up a character incrementally by finding better gear, leveling up, and so on. Combining this idea with puzzles sounds like an engaging and exciting game (at least for me :D)!

● Potential features
  ○ I imagine the hardest part about this project would be the puzzle-creation (I'm not even sure what the puzzle game will be yet), but a generator for puzzles will be necessary. Other features would include health bars and enemy AI.
Ethan Mullen

- **Puzzle Game like Ballz**
  - My game is fun because of the strategy needed and the satisfying visuals after shooting the balls to hit the bricks.

- **Potential features**
  - Bouncing collision behavior, random shape and number generation for blocks
Point and Click Adventure

- Heavily inspired on Kingdom Ka. Potentially also will draw inspiration from Gorogoa. It will contain an intro sequence, then a map (like a hub) from which you can decide to venture to 4 locations of choice. Each has a character or event to traverse through and perhaps obtain an item, once 3 are obtained, the last one (which you could enter before, but not go through) will unlock itself to the ending, which might be like 3 rooms.
- Will be surreal and fun to look at and walk through

Potential features

- Will need to incorporate a sort of GameWorld coordinate - > "3D world" coordinate that will allow me to move the player. Strong capabilities in click and understanding clicking events. Facility to create different worlds to traverse to, and to remember information upon returning to them. Save features. Dialouge boxes and the ability to traverse through them, also having to pause player actions while dialouge occurs (atleast as an option)
Rhythm Game

- Inspired on Rhythm Heaven. About a honey bee that is instructed to kill off other species of live bees by its commander.
- Will basically be one big mini-game that throws rhythm based cues to respond to.
- Will have an increasing level of difficulty as you progress. Will involve some way of randomizing inputs to react to so that even same levels will never be the same.

Potential features

- Level Generator. Rhythm recognizer.
A Wiz and Alc Mashup

- The game that I currently have in mind builds off of Wiz, and sort of combines it with Alc to allow users to keep inventory and have more interaction within the game. While the vision in my head is pretty blurred right now/needs a lot of structuring, the goal would be to make Wiz slightly resemble old-school Runescape layout wise (although I could be a little off on that after not playing it in about a decade). This would involve hints/text being found within the game world as guides, more interaction options with game objects/enemies, etc.

- In addition to being just a maze with enemies, I want to add a strategy component that involves more user/character interaction in-game (ex: having to collect and combine certain items to make a key to the next level, playing around with level difficulty/variety of types of levels, bosses, etc.)

Potential features

- While I'm not sure exactly everything that I'll be adding to my engine vs what will be part of the game itself, I know that I would like to add a type of physics component to deal with different speeds and game objects possibly handling a gravity setting, and potentially a text component that allows text to appear on the screen when certain objects are interacted with by the character.
Multiplayer 2D Soccer
- It will probably have power ups, loot, and a shop in the real time game. The shop/power ups will give the player special powers.
- It will be a twist on the regular/realistic soccer games. Also multiplayer helps with competitive interest.

Potential features
- Saving and loading
- Multiplayer/Networking
- Minimap
- Player controls switching between game objects
- Customizing player controls
- Sounds
Sohum Gupta

- **System-based Sandbox Game**
  - I think it would be really interesting to make a sort of system-based sandbox game – think Terraria, but likely more specific and thematized. I don't really have a super fleshed out idea of what the game would consist of exactly, but I think I'd definitely want there to be some sort of crafting/inventory system, as well as making the world destructible. It would also be interesting to explore something that uses procedural generation in the way that a game like Noita does (though on a much simpler scale, of course). I really like the idea of a lot of game systems interacting with each other, and I think it would be super fun to implement a lot of the systems.
  - Ideally, the multiple systems would be set up and procedural generation would be used such that the game would be relatively complex simply by the interactions of the multiple systems. I think sandbox games end up being fun based on the amount of unique things you can do in the game and the complexity of the world, and I think it’d be a really good opportunity to use procedural generation in many different ways.

- **Potential features**
  - I think there would likely be some noise-based map generation that would need to be implemented, but more than that would be the game systems, specifically inventory and crafting systems, which would definitely be large engine features. Also, the way combat is handled in my engine right now is quite clunky, and I think setting up a larger, more generalizable combat system would also be a part of the project.
Juan Mina

- Platformer
  - akin to super meat boy, (fast pace) and limbo, (style)
  - a challenging but rewarding platformer

- Potential features
  - Some sort of fog/mist, just particles in general, sound support, potentially controller support.
Lecture 5

Paper Prototyping
How do you turn an idea into a game?

Making a game is hard

- Animation
- Systems
- Playtesting
- Music / SFX
- Gameplay loop
- Etc.

These all require a locked in foundation of what the game is -- ie if you want to change the player’s moveset, think of the cascade of things you’d have to change in the game to make that a reality
Iteration is important

Design is all about iteration, but for games, iteration is costly

Prototype vs final version of Breath of the Wild
Solution:

Paper prototype as much as possible so you have a clear direction of the game you want to make before making it (it’s a lot easier to draw a character in paper then it is to make a 3d model in blender)
General framework / idea

- You won’t be able to make a fully detailed and realize version of your game in paper (it will be ‘lofi’), but you can illustrate a general idea in paper that is communicate-able.
- Offload as much design work as possible so when you’re working on the real game, you minimize core changes to the game as much as possible.
- Validate that the game is worth making in the first place, you don’t want to be in the position where after making the game, you realize it’s not as fun as originally intended.
How to paper prototype

The objective is to iterate on an idea, and then validate ‘is it fun?’ or ‘is it interesting’

When prototyping, try to answer the following:

- What is the game loop? What will the player be doing for most of the game
- What are the mechanics available for the player? Movement, attacks, etc.
- What’s the win state / lose state? How does the player reach both?
- What’s the scale of the world? How much does it take to design 1 level?
- Other questions that come up, put yourself in the perspective of the player!
What to do with a paper prototype

Iterate, Test, Evaluate, Repeat (ITER)

- After an idea is translated into paper, get feedback from others, does the idea make sense in the context of the game?
  - “The player currently has a sword, but I’m thinking if the player interacts with this character, the player receives a shield, which will let them do [stuff]. What are your thoughts?”
- Don’t be afraid to throw out / throw in new ideas
  - It’s just paper!
- Write out the general story beats or progression of the game
  - What would the player generally experience in a full playthrough of the game
- Write out your thoughts on the work that lies ahead
  - Very important in terms of scope of viability for the game
  - Establish what parts of the game you want to prioritize
This time is for you guys, paper prototype with others!

If you guys are interested, stick around and start talking with others on the game ideas you guys presented

If there is an idea that really interests you, paper prototype it!

Let us know if you have questions or want our feedback, we are here to help!
'Til Next Week!

- Nin I released today
- Final groups due by **Thursday (10/28)**