Lecture 7
Announcements
Feedback for M 1

- Some of you had stacking shapes
- The stack maybe exploded eventually
- Don’t worry, stacking boxes is hard to do properly
- This week’s lecture will help but not alleviate the problem
Feedback for M 1

• Sometimes shapes would settle overlapping other shapes

• Two possible causes:
  – Not translating enough
  – Doing collision response before integration, hence drawing before translating
Goal Velocity

• For side-scrolling platformers (like M), goal velocity should only apply for the x-component of your velocity

• Otherwise, you get a weird “feather falling” effect when jumping
Mid semester things

• We’ll be sending out mid semester grade reports soon – keep an eye on your inbox

• Most of you have filled out the mid semester feedback form – thanks!
QUESTIONS?

Announcements
Lecture 7
Physics II
M1 Collision Response

- You obtained MTV in M1, how did you use it?
  - Move things out of collision
  - Apply some impulse along MTV
  - It probably worked most of the time, or only if you modeled specific behavior
  - Other times the response is weird
Controlling Impulse

• Want more realistic collisions
• Two things to do:
  – Support a range of collision behavior
  – Determine the physically correct collision result
• Also need to be able to code it

```cpp
void onCollide(Collision col) {
  Shape o1 = col.shape1;
  Shape o2 = col.shape2;
  Vec2f mtv = col.mtv;
  o1.move(mtv.smult(0.5f));
  o2.move(mtv.smult(-0.5f));

  // code to calculate impulse
  applyImpulse(imp1);
  applyImpulse(imp2);
}
```
Restitution

- Property of physical entities
- Value between 0 and 1
- 0 is perfectly inelastic, 1 is perfectly elastic, etc.
- The coefficient of restitution (COR) between two entities is the geometric mean of their restitutions:

\[ COR = \sqrt{r_1 r_2} \]
Correct Collisions

• How do we find the physically correct collision response?
  – i.e. given $u_a$ and $u_b$, what are $v_a$ and $v_b$?
• Use physical definition of the COR:
• $COR = \frac{v_a - v_b}{u_a - u_b}$
Final Velocities

- Conservation of momentum:
  \[ m_a u_a + m_b u_b = m_a v_a + m_b v_b \]
- The COR equation can be rewritten as
  \[ \ast \text{COR} \ast (u_a - u_b) = v_a - v_b \]
- So conservation of momentum becomes
  \[ m_a u_a + m_b u_b = m_a v_a + m_b (v_a - \ast \text{COR} \ast (u_a - u_b)) \]

\[ \ast \text{COR} = \sqrt{r_1 r_2} \]
Final Velocities

- Solving for $v_a$:
  
  $$v_a = \frac{m_a u_a + m_b u_b + m_b COR^* (u_b - u_a)}{m_a + m_b}$$

- Similarly for $v_b$ (reverse $a$ and $b$ basically):
  
  $$v_b = \frac{m_a u_a + m_b u_b + m_a COR^* (u_a - u_b)}{m_a + m_b}$$
Once we determine what the final velocities of the objects should be, how do we apply this?

- Impulse or velocity setting?
  - Setting velocity affects other collision response calculations
Translation to Code

• Impulse > velocity setting
  – Setting velocity affects other collision response calculations
Translation to Code

- Impulse > velocity setting
  - Setting velocity affects other collision response calculations
Translation to Code

- Impulse > velocity setting
  - Setting velocity affects other collision response calculations
- Recall the definition of the impulse: \( m\Delta v \)
- We really want the difference of final and initial velocity
Velocity Difference

• The velocity differences are:

\[ v_a - u_a = \frac{m_b(1+COR)(u_b-u_a)}{m_a+m_b} \]

\[ v_b - u_b = \frac{m_a(1+COR)(u_a-u_b)}{m_a+m_b} \]
So the impulse you want to apply is:

\[ I_a = m(v_a - u_a) = \frac{m_am_b(1+\text{COR})}{m_a+m_b} (u_b - u_a) \]

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

You can see that they are equal and opposite, as we would expect
Static Shapes

- If object $a$ is static, you can show that the equations in the last slide reduces to
  - $I_a = m_b (1 + COR) (u_b - u_a)$
  - $I_b = m_b (1 + COR) (u_a - u_b)$
- Vice-versa if object $b$ is static
- You should special case this
Note about Velocity

- Note that all velocities in these equations are for the one-dimensional case
- But your 2D engine is 2D!
- Velocity only changes in the direction of the MTV
- So the $v$’s and $u$’s in the equations are the projection of velocity onto the MTV
Putting it all together

Physically correct collision response:
1. Calculate COR with the restitutions of the shapes
2. Project velocities onto MTV
3. Apply impulse formula to calculate impulses
4. Apply corresponding impulse to each shape
Lecture 7
Raycasting
What is raycasting?

• Determine the first object that a ray hits
• A ray is like a ray of light, has a source and direction and goes on forever
• Think of it as shooting a laser in a particular direction
Raycasting Uses

• When would we need to raycast?
  – Hitscan weapons
  – Line of sight for AI
  – Area of effect
  – Rendering
The Ray

- A ray is a point (source) and a direction
- Point on ray given by:
  - $\vec{r} = \vec{p} + t\hat{d}$
  - $\vec{p}$ is the source point
  - $\hat{d}$ is the direction
    - This must be normalized!
  - $t$ is a scalar value (length)
Basics

• Raycasting boils down to finding the intersection of a ray and shapes
• Kind of like collision detection all over again
• You want the point of collision as well
Ray-Circle

- If the source is outside
- Project center onto ray
- Check if the projection is positive and the projection point is within the circle
Ray-Circle

- If the source is outside
- Project center onto ray
- Check if the projection is positive and the projection point is within the circle
- Point of intersection?
Ray-Circle

- If the source is outside
- Project center onto ray
- Check if the projection is positive and the projection point is within the circle
- Point of intersection?
- \[ \hat{p} + (L - \sqrt{r^2 - x^2})\hat{d} \]
Ray-Circle

• If the source is inside the circle
• Do the same thing as if outside, but:
  • \( \vec{p} + (L + \sqrt{r^2 - x^2}) \hat{a} \)
• Now \( L \) is allowed to be negative
Ray-Polygon/AAB

• A polygon/AAB is composed of edges
• We can check for intersection of ray by checking for intersection of all edges
• There is no shortcut for AABs this time
Edge is defined by two end points, $\vec{a}$ and $\vec{b}$

We need some other vectors:

$\hat{m}$ is direction of the segment (normalized)

$\hat{n}$ is the perpendicular to the segment (normalized)
Ray-Edge

• Firstly, determine if the segment straddles the ray
• Use cross products
• \((\vec{a} - \vec{p}) \times \vec{d} \text{ and } (\vec{b} - \vec{p}) \times \vec{d}\) must be of opposite sign
• Therefore no intersection if 
  \[ ((\vec{a} - \vec{p}) \times \vec{d}) \left((\vec{b} - \vec{p}) \times \vec{d}\right) > 0 \]
Ray-Edge

- Secondly, determine if the two lines intersect
- Point of intersection
- \( \vec{q} = \vec{p} + t \vec{d} \)
- Solve for \( t \)
- \( t \) must be nonnegative!
Ray-Edge

• Because $\vec{q} - \vec{b}$ lies on the segment
• $(\vec{q} - \vec{b}) \cdot \hat{n} = 0$
• So plugging in:
• $(\vec{p} + t\vec{d} - \vec{b}) \cdot \hat{n} = 0$
• $t\vec{d} \cdot \hat{n} = (\vec{b} - \vec{p}) \cdot \hat{n}$
• $t = \frac{(\vec{b} - \vec{p}) \cdot \hat{n}}{\vec{d} \cdot \hat{n}}$
Ray-Polygon

- Intersect the ray with all the edges of the polygon
- Ray intersects polygon if it intersects at least 1 edge
- Keep track of the point that is closest to the source (the lowest value of $t$)
Putting it all together

Raycasting:

1. Intersect ray with every shape in the world
   1. For circles, use the circle-ray algorithm in the slides
   2. For polygons and AABs, intersect each edge and use the closest

2. Keep track of closest intersection point from the source as well as the corresponding shape
Lecture 7
Tips for M II
Collision Groups

• Groups of objects that only collide among themselves or never collide with each other

• You can represent these groups however you want
  – One way to do it is with a group index and/or group mask
  – See Box2D documentation for inspiration
Raycasting Groups

• Similar to collision groups
• Some objects shouldn’t be raycasted on, just like some objects are not meant to be collided
• Special case: the source
  – If you’re raycasting from a shape, you don’t want to check the shape itself
Raycasting

- Raycasting needs a list of entities, so the world needs to be involved
- Make Raycast a method of world
  - Return RayCollision object containing point of collision and entity collided
Jumping Properly

- Extra requirement this week: player can jump once iff standing on a solid object
- Use information from your collision response object!
- If the MTV points upwards, then the player has hit something solid below it
  - Player is now allowed to jump
Fixed Timestep

• Most of you will have noticed weird behavior when you fullscreen/resize
• It screws around with the nanosSincePrevTick
• Solution: fixed timestep
  – Give the physics world some constant time when ticking
  – Tick as many times as possible on each game tick
Tips for M II

JAVA TIP OF THE WEEK
Methods of Object

- Object has a few methods that you should know/care about:
  - hashCode()
  - equals(Object o)
  - toString()
**toString**

- Called whenever you use the object as a string
- You should override this in your classes to return meaningful strings
  - It’s worth doing if you debug with printlines
- See `Vec2f` for examples

```java
public class Object {
    public String toString(){
        return getClass().getName() + "@" + hashCode();
    }
}

public class PhoneNumber {
    public String toString(){
        return "(" + _areacode + "\n" + _number;
    }
}
```
**equals**

- Determines if object is equal to another object
- The default method compares references
  - Doesn’t work if you want equal but not the same object (e.g. `String`)

```java
public class Object {
    boolean equals(Object obj){
        return (this == obj);
    }
}
```
equals

- To override:
  - Use default check first
  - Check if `instanceof`
  - Cast to object
  - Compare things
- You can access fields directly!

```java
class Book {
    String author;
    String title;

    boolean equals(Object obj) {
        if (this == obj)
            return true;
        if (!(obj instanceof Book))
            return false;
        Book o = (Book) obj;
        return o.author.equals(author) && o.title.equals(title);
    }
}
```
Contract of equals

- A few things must be true about your equals():
  - o.equals(o) must be true
  - o.equals(null) is always false
  - must be symmetric
  - must be transitive
  - o1.equals(o2) changes only if o1 or o2 changes

```java
public class Book {
    String author;
    String title;

    boolean equals(Object obj) {
        if (this == obj)
            return true;
        if (!(obj instanceof Book))
            return false;
        Book o = (Book) obj;
        return o.author.equals(author) && o.title.equals(title);
    }
}
```
hashCode

• Returns the hash of the object
• This value is used by Java’s Map data structures
• You must override hashCode if you override equals
  – Otherwise data structures can do weird things

```java
public class Object {
    public int hashCode();
}
```
Contract of hashCode

- hashCode must return the same value for the same object unless it is changed
- If two objects are equal, they have the same hashCode
- If two objects are unequal, ideally they will have different hashCode values

```java
public class Object {
    public int hashCode();
}
```
Overriding `hashCode`

- Many ways of actually calculating hash of an object
- You can look them up if you want
- Eclipse can auto-generate hash functions based on fields of your choosing

```java
public int hashCode()
{
    int hash = 5; //prime
    int prime = 89; //prime
    hash = prime*hash + x;
    hash = prime*hash + y;
    hash = prime*hash + z;
    return hash;
}
```
Tips for M II

QUESTIONS?
GAME DESIGN 6
Level Mechanics
Brief Summary

- Overview of major level design approaches
- Techniques for getting the most out of your levels
What is a level?

- From Wikipedia: “the total space available to the player during the course of completing a discrete objective.”
- Essentially: the environments that your player interacts with
The level designer

• Level designers must:
  – Design the visual appearance of the level
  – Decide what areas are traversable and accessible
  – Add triggers for scripted events and cutscenes
  – Place enemies, obstacles, power-ups, etc.
  – Create navigation nodes for AI pathfinding
  – Add sounds and music
  – And more!
Level Structure

• Broadest types of level structures:
  – Linear
  – Bottleneck
  – Open-world
  – Hub/Spoke
  – Combination

• It is near impossible to create a game completely devoid of linear elements
Linear
Linear

• Pros
  – Developer has full control of the player
  – Extremely common level layout

• Cons
  – Lack depth, if not done correctly
  – Extremely common level layout

• Examples: *Half-Life, Call of Duty, Limbo*
Bottleneck
Bottleneck

• Pros
  – More player freedom
  – Replay value

• Cons
  – Similar to linear
  – Players may backtrack on a previous path

• Examples: Super Mario Bros. 3, Deus Ex
Branching

START

BRANCH

BRANCH

FINISH 1

FINISH 2

FINISH 3
Branching

• Pros
  – Multiple endings (replay value)
  – More player freedom

• Cons
  – Hard to design
  – More content required

• Examples: The Stanley Parable, Shadow the Hedgehog
Open-world

START

Objective

Objective

Objective

FINISH
Open-world

• Pros
  – Player has complete freedom

• Cons
  – Developer has little control of player
  – If not given any direction, player will get bored
  – Huge amount of content needs to be created

• Examples: *Grand Theft Auto*, *World of Warcraft*
Hub/Spoke

PART 1
(Unlocks 2)

PART 2
(Unlocks 3)

PART 3
(Unlocks 4)

PART 4
(unlocks exit)

Hub A

Exit Path
Hub/Spoke

• Pros
  – Gameplay time
  – More fun than a level select screen

• Cons
  – Hub needs some compelling reason to exist
  – Can lead to boredom if player has to backtrack

• Examples: Super Mario 64, Sonic Adventure
Combination

• Most games use some combination of these structures
• Often, the structures are nested
  – Open-world with linear dungeons (*Skyrim*)
  – Linearly ordered levels that have branches and bottlenecks within them (*Sonic*)
• For large scale games, it can get pretty crazy…
Example: The Legend of Zelda: The Windwaker

Introduction (Linear)
Outset Island → Forsaken Fortress → Windfall Island

The Great Sea

1. Dragon Roost Island
2. Forest Haven
3. Outset Island
4. Tower of the Gods
5. Forsaken Fortress

The Three Pearls (Open-world/hub)

Repairsing the Master Sword (Open-world)

Ganon’s Tower

The Great Sea

Earth Temple
Wind Temple
Triforce Pieces

Every dungeon and island has nested substructures as well...

…but the overall structure is linear!
Level Structure

QUESTIONS?
30 seconds of fun

“In Halo 1, there was maybe 30 seconds of fun that happened over and over and over again, so if you can get 30 seconds of fun, you can pretty much stretch that out to be an entire game.”

-Jaime Griesemer
Flow

- "A feeling of complete and energized focus in an activity, with a high level of enjoyment and fulfillment" – Schell
- Preconditions:
  - Goals are clear
  - Feedback is immediate
  - Balance between perceived challenge and perceived ability
Interest curves

- Used in all kinds of storytelling
- Starts off with a hook
- Includes a climax close to the end

![Interest curve graph](image-url)
Case study: *Left 4 Dead*

- FPS based on a group of 4 survivors in a zombie apocalypse
- Extremely simple goal: get from a starting position to a safe house
- What makes it fun and replayable?
- AI director adjusts pacing based on how you play
  - Procedurally populates world with enemies
Left 4 Dead: AI Director

• Promote "structured unpredictability"
  – Uniform randomness isn't exciting
  – Exact positioning isn't replayable

• Want to move toward a target intensity
  – Model "Emotional Intensity" of each player
  – Take max of intensity of all players
  – If too high, temporarily remove threats
  – If too low, create an interesting population
Left 4 Dead: "Emotional Intensity"

- Model survivor intensity as a number
- Increase intensity when player is damaged, pushed off a ledge, etc.
- Decay intensity over time when player is not engaged
Left 4 Dead: Population modulation

- Use emotional intensity to modulate population
  - Build up, sustain, fade, and relax
  - Sound familiar? All about interest curves!
Case study: *Batman: Arkham Asylum*

- Third-person adventure; player is Batman
- Variety of gameplay types
  - Puzzles
  - Roaming
  - Combat
  - Gargoyle room (sneaking without being noticed)
  - Cinematic (no player interaction)
  - Boss fight
  - Gadget (player given new gadget to play with)
- How to combine all these elements?
  - Pacing through variety
Batman: Arkham Asylum

- Gargoyle room
- Fight
- Cinematic
- Roaming
- Puzzle
- Bossfight
- Gadget

- Introduction
  - Introduction seq.
  - Big grunt

- To Batmobile
  - Batmobile
  - To medical facility

- The medical facility
  - Chen
  - Aaron
  - Girl
  - Scarecrow
  - Bane

- Treatment continued
  - Crocs lair
    - Collect Spores
    - Flee
  - Batcave
    - Climbing
    - Pumps
    - Control room

- Cave cont.
  - To Botanic garden
  - Botanic garden
    - Ivy Bossfight
  - To Penintiary

- Penintiary
  - Final battle
Batman: Arkham Asylum

• Interleaves contrasting gameplay elements
  – Rarely more than 5 minutes of successive combat
  – What does the interest curve look like?

• Steady progress
  – New gadgets awarded roughly every 60 minutes of gameplay
  – Allows short bursts of exploration; an entirely open world is often too overwhelming
    • Anticipation of unlocking new areas
  – Clear objectives and clear rewards
Pitch Your Game!

Impress us!
Presentation Order

1. mthiesme  8. jlkao
2. pkirschn   9. lcamery
3. murbany    10. ah
4. bsheeran   11. jbunce
5. cmgianca   12. urangane
6. hsufi      13. lcallana
7. bweedon    14. jl48
M1 PLAYTESTING!

Hooray!