LECTURE 6

Announcements
Tou II Feedback

• Handins look varied and fun!
• Write a Range or Interval class
  – Makes math code easier to:
    • read
    • write
    • modify
  – Makes M I easier to build on top of Tou
Mid-Semester Feedback

• We only have 3 responses so far 😞
• By doing the surveys you will:
  – Make the class more enjoyable for future students
  – Get to decide what advanced topics we cover first
• Please fill out both surveys!
M: Your last project (kinda)

- Platformer game
  - 3 weeks
  - Physics
  - More physics
  - Externalizing game logic

IMPLEMENT ALL THE PHYSICS
QUESTIONS?
Lecture 6

Physics I
INTEGRATION
Velocity

- Rate at which position changes
  \[ \dot{v} = \frac{\Delta x}{t} \]
- Just increment pos?
  \[ \text{pos} = \text{pos} + \text{vel} \]
- Works fine with constant frame rate...
- But frame rates vary
- So always use time
  \[ \text{pos} = \text{pos} + \text{vel} \times \text{time} \]
Acceleration

• The rate at which velocity changes
  \[ \ddot{a} = \frac{\Delta \dot{v}}{t} \]

• Useful for gravity, springs, wind...

• \( v_{el} = v_{el} + acc \times time \)
Which order to update?

Position first (Euler)
- pos = pos + vel*time
- vel = vel + acc*time

Velocity first ("Symplectic Euler")
- vel = vel + acc*time
- pos = pos + vel*time
Which order to update?

Position first (Euler)

- \( \text{pos} = \text{pos} + \text{vel} \times \text{time} \)
- \( \text{vel} = \text{vel} + \text{acc} \times \text{time} \)

Velocity first ("Symplectic Euler")

- \( \text{vel} = \text{vel} + \text{acc} \times \text{time} \)
- \( \text{pos} = \text{pos} + \text{vel} \times \text{time} \)
- stabler than Euler; use this
Collision response

• You know how to detect whether 2 objects collide
• How do we make objects respond in a physically believable way?
• General strategy:
  1. Move objects out of collision
  2. Change their velocities
Moving out of collision

- Many ways to move the ball out of the wall
- We want the minimum
- “Minimum translation vector” (MTV)
- Same as overlap
- For now, assume we can find MTV
Changing velocity for collision

• Just reverse the object’s velocity?
  \[ v_{\text{el}} = -v_{\text{el}} \]

• Reverse the y component?
  \[ v_{\text{el}_y} = -v_{\text{el}_y} \]

• Reverse the perpendicular component?
  – What if 2 moving objects collide?
2-moving-object collisions

- Reverse both velocities?
- Doesn’t always work
- Apply equal and opposite forces
- An instantaneous force is called an impulse
Units

Without mass

• position  m  
  – \( \ddot{x} \)

• velocity  m/s  
  – \( \ddot{v} = \Delta \ddot{x}/t \)

• acceleration  m/s\(^2\)  
  – \( \ddot{a} = \Delta \ddot{v}/t \)

With mass

• (no equivalent)

• momentum  kg m/s  
  – \( \dot{p} = m \ddot{v} \)

• force  kg m/s\(^2\)  
  – \( F = \Delta \dot{p}/t \)

• impulse  kg m/s  
  – \( \Delta \dot{p} \)
Implementing force and impulse

- `applyForce(...)` accumulates force
- `applyImpulse(...)` accumulates impulse
- `onTick(...)` applies force and impulse, clearing them for next frame
- `applyForce(...)` and `applyImpulse(...)` only take effect on next `onTick(...)`
- “Static” (immovable) objects can override `applyForce(...)` and `applyImpulse(...)` and make them no-ops

```cpp
class PhysicsEntity {
    float mass;
    Vec2f pos, vel;
    Vec2f impulse, force;

    void applyForce(Vec2f f) {
        force += f;
    }

    void applyImpulse(Vec2f p) {
        impulse += p;
    }

    void onTick(float t) {
        vel += t*force/mass + impulse/mass;
        pos += t*vel;
        force = impulse = 0;
    }
}
```
Impulse collision response

• Translate objects out of collision
  – Each by MTV/2
  – Or proportional to velocity in direction of MTV

• Apply some impulse proportional to MTV to each object
  – How much?
  – This week: guess/hack
  – Next week: more realistic
Collision callbacks

- Pass in other Entity
- Separate Collision info object (really a struct)
- Pass in the MTV
- Maybe pass in which shapes collided
  - Enemies with weak points
- Maybe allow callback to prevent solid collision response
  - One-way platforms
- Double dispatch is your friend!

```java
class PhysicsEntity {
    boolean onCollide(Collision collision);
}

class Collision {
    final PhysicsEntity other;
    final Vec2f mtv;
    final Shape thisShape;
    final Shape otherShape;
}
```
Can we do better?

```java
abstract class PhysicsEntity<T extends PhysicsEntity<T>> {
    boolean onCollide(Collision<T> collision);
}

abstract class PhysicsWorld<T extends PhysicsEntity<T>> {
    List<T> physEntities;
    @Override void onTick(float seconds) {
        for (T entity1, entity2: physEntities) {
            // check if entity1 collides entity2, construct a collision object,
            // and pass that object to both entities
        }
    }
}

class Collision<T extends PhysicsEntity<T>> {
    final T other; final Vec2f mtv; final Shape thisShape; final Shape otherShape;
}

// MEntity extends PhysicsEntity<MEntity>, and MWorld extends PhysicsWorld<MEntity> -
done, with all callbacks generated in the engine!
```
Physics I

MINIMUM TRANSLATION VECTOR
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 (normalized!) axis, find 1D MTV of shapes’ projections
2. Find the axis giving minimum 1D MTV
3. 2D MTV is 1D MTV times that (normalized) axis
MTV interactive demo

• Same as last week
• Arrows are potential MTVs for box against triangle
• Purple arrows are the actual MTV
• SAT guarantees that MTV is on a separating axis
Computing intervals’ MTV

Float intervalMTV(Interval a, Interval b)
    Float aRight = b.max - a.min
    Float aLeft = a.max - b.min
    if aLeft < 0 || aRight < 0
        return null
    if aRight < aLeft
        return aRight
    else
        return -aLeft
Computing polygons’ MTV

Vec shapeMTV(Shape a, Shape b)
Float minMagnitude = +infinity
Vec mtv = null
for Vec axis in allAxes
    Float 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
Computing circles’ MTV

• Circle vs Circle
  – Compare \( \text{dist}(\text{center1}, \text{center2}) \) and sum of radii
  – MTV is parallel to line connecting centers

• Circle vs Poly
  – Use the same set of axes you did last week:
    • Poly’s edge normals
    • Vector from circle center to closest vertex
Computing circles’ MTV (ctd)

• Circle vs Box
  – If Box contains circle center
    • Use Box’s axes
  – Otherwise
    • Clamp circle center to Box
    • Compare dist(center, clampedPoint) and radius
    • MTV is parallel to line connecting
Computing boxes’ MTV

• Easy but inefficient way:
  – Box converts itself to a Poly

• Efficient way:
  – Use (0, 1) and (1, 0) as the only 2 axes
MTV pitfalls

• Be careful with signs and argument order
  – Especially when reversing args for double dispatch

• Can use asserts:
  – MTV of shape A to move it out of shape B should point from B to A
  – assert dot(MTV, A.center – B.center) > 0
QUESTIONS?
Lecture 6
Tips for M I
Gravity

• On each tick, for each entity e, e.applyForce(g.smult(e.mass))

• Static objects’ applyForce() does nothing
  – Consider how collision with a static object differs from collision with dynamic object
Player motion

• Set velocity while left or right is held?
  – Too sudden
  – Can interact badly with other physics

• Apply force while left or right is held?
  – Asteroids!
Goal velocity

- **goalVelocity** set directly from arrow keys
- Gradually set velocity to **goalVelocity**
- By applying a force
  - $F = k(\nu_{goal} - \nu_{current})$
Tips for M I

QUESTIONS?
Pitfalls of inheritance

- What if you
  - Change name?
  - Change arg types?
- Might forget to change a subclass (if not using Eclipse’s refactor tools)
- Subclass’ method no longer overrides anything!
- How can we avoid this?
The @Override annotation

- Whenever you override a method, mark it @Override
- If superclass changes, you get a compile error
- Eclipse can insert these for you

```java
public class Entity {
    public void onTick(float t) {
        // ...
    }
}
```

```java
public class Bullet extends PhysicsEntity {
    @Override
    public void onTick(long t) {
        // ...
    }
}
```

Bullet.java:2: method does not override or implement a method from a supertype
Other standard annotations

• Annotations can mark:
  – classes
  – methods
  – fields
  – variables
  – parameters

• Can create your own...
  – See Oracle’s tutorial if you’re curious

• @Deprecated
  – Alternative to removing a method entirely
  – Compile warning when marked method is used

• @SuppressWarnings
  – Tells the compiler not to catch you when you fall
  – Avoid if at all possible
  – Typically an indication of a larger problem
QUESTIONS?
LECTURE 6
Final Project Overview
Overview

• Can be any 2D game
• You should work in groups!
• Each person is responsible for 10 “points” worth of new engine features
  – More members in a group means more engine features
  – More details in the final project handout
Timeline

• 4 main parts:
  – Week 1: Idea
  – Week 2: Form groups and get approved
  – Week 3: Design
  – Weeks 4-8: Code, playtest, polish, present
Week 1: Idea (this week!)

• A ½ to 1 page document
• Describe basic gameplay idea
  – How is your game fun?
  – Why should someone want to help make it?
• Describe engine feature(s) you plan on implementing
• Give a 60-second “elevator pitch” of your game in class
• Everyone must give a pitch, even if you already know your group and which project you’re working on
Week 2: Groups

• Form a group (or decide to work alone)
• Finalize game and engine features
• Each group must meet with the TA’s to present the following:
  – A more polished idea of the game
  – Breakdown of member responsibilities for engine
Week 3: Design

- Research new engine features
- Design the engine and game
- Exact breakdown of member responsibilities
- Choose someone’s engine to use or integrate engines
- For groups of 3 or more
  - Explain how you will use version control
Weeks 4-8

- **Week 4:**
  - Engine should be mostly done
  - Game exists

- **Week 5:**
  - Engine should be done
  - Game is playable
  - 5 playtests *per member* from people not in CS1971
Weeks 4-8

• Week 6:
  – Game should be mostly done
  – 5 more playtests per member from outsiders

• Week 7:
  – Game should be done
  – 5 playtests per member
  – Powerpoint slideshow for postmortem presentation
Weeks 4-8

• Week 8:
  – Polish up your game, bug fixes, etc
  – Create an executable and put it in /contrib
  – Make a video demo of your game from gameplay footage

• It is now December 20th

• And then you’re done!
Final Project Overview

QUESTIONS?
GAME DESIGN 5
(Past Final Projects)
Brief Summary

• Focuses on past video games developed by students in:
  – CS32
  – CS1971

• What went right?

• What went wrong?

• Discuss final project ideas
Sunlab Games

• Fun for the first five minutes…
• …Then you remember you have other work to do
Common Themes

• Overstretching
  – More than what can be created in a 4 week time period

• No polish
  – Focused too much on content development

• NOTE: These are post-mortem summaries; your post-mortems should be more in-depth
Reject Wars (2012)

- Networked platforming shooter
- Java sockets and Threads
- Similar to Super Smash Bros.
Reject Wars – What went right?

• Networking
  – Understood Java Sockets and Threads a lot better
• Basic gameplay – done!
• Character sprites
  – Zach Davis! Lump Space Princess! Magikarp!
Reject Wars – What went wrong?

• Networking
  – Serialized the game world…
  – Scaling issues

• Environmental sprites

• Stage-based strategy didn’t work out
Prismatic (2013)

• 2D top down puzzle game (Castle Crashers theme)
• Based upon interacting with light rays
• 20+ different entities to interact with (enemies, objects, etc...)
Prismatic— What went right?

• 4-man team — excellent division of labor
  — Level editor, gameplay, UI, file system

• Good expectations

• Users have own profiles, can upload levels for each other to play
Prismatic—What went wrong?

• Lack of playtesting
• Some, but not much, polish
• Not much else…
Galaxy Ferret (2013)

- Shoot 'em up inspired by Star Fox
- Player purchases upgrades to make their ship stronger
- Open structure – player chooses which levels to play
Galaxy Ferret – What went right?

• Well-polished gameplay with varied mechanics
  – Barrel rolling, regenerating shields, different weapon types, large enemy variety, fun bosses

• Consistent cartoony-space aesthetic

• Particles, particles, particles

• Built off of a strong Tou 2, so most of the core was done from the start
Galaxy Ferret – What went wrong?

• Lots of little engine features implemented but barely used
  – Sound, parallax, cutscenes, screen transitions
• Open-world structure ended up not making sense and was hard to balance
• Levels were hardcoded and slow to create
  – Should have created some parsable file format
Demos!

- Prismatic
- Galaxy Ferret
- Constant Chaos
CS1971: Past Projects

QUESTIONS
TIPS FOR FINAL
Coming up with a good idea

• All games start with an idea
• Ideas are finicky – they come on their own terms, when they feel like it
• Be prepared to write them down whenever they come to you!
Developing an idea

• Once you have your initial idea, start defining crucial details:
  – Genre?
  – Singleplayer or multiplayer?
  – Ideal team size?
  – Biggest risks?

• Come up with a simple prototype you can build quickly
  – Minecraft example: only placing/removing blocks with just a few distinct block types
  – These will help you immediately find out if your idea is fun
What we don’t see...

- Simulation (*Roller Coaster Tycoon*)
- Strategy (*Starcraft, Civilization*)
- Rhythm (*Dance Dance Revolution*)
- Racing (*Mario Kart*)
- Fighting (*Street Fighter*)
- Horror (*Clocktower*)
- Multiplayer co-op (*Battletoads*)
What we get A LOT of

• Platformers
• RPGs
• Platformers
• Platformers
Avoiding problems…

• Organize; know your abilities
• Be reliable to your teammates
• Settle on a coding convention
  – Follow it!
• Take initiative
  – There’s always that one guy who says: “Doesn’t matter to me, I’ll do anything”
• FIX BUGS!!!
  – Don’t put it off till later!
Making it better...

• Add juice!
  – Screen shake, tweening, mouse effects
• Playtest, playtest, playtest
• Steal ideas from awesome games
  – Even AAA Companies do this
Single Player vs. Multiplayer

**Single player**
- Easier to balance
- No networking
- Harder to keep the player engaged

**Multiplayer**
- Other players essentially become your content
- Increased bang for your buck content-wise
- Networking can be tricky to debug
Tou 2 Playtesting!

Hooray!