the gamedesigninitiative at cornell university

Lecture 18

box2d Physics

Physics in Games

- Moving objects about the screen
 - **Kinematics**: Motion ignoring external forces (Only consider position, velocity, acceleration)
 - **Dynamics**: The effect of forces on the screen
- **Collisions** between objects
 - **Collision Detection**: Did a collision occur?
 - **Collision Resolution**: What do we do?

Physics in Games

- Moving objects about the screen
 - Kinematics: Motion ignoriant or corces
 (Class Body cation)
 Dynamics: The effect of forces on the screen
- Collisions between objects





Body in box2d

- Represents a single point
 - Center of the object's mass
 - Object must move as unit
- Properties in class Body
 - Position
 - Linear Velocity
 - Angular Velocity
 - Body Type
- There are 3 body types
 - **Static**: Does not move
 - **Kinematic**: Moves w/o force
 - **Dynamic**: Obeys forces





Body in box2d

- Represents a single point
 - Center of the object's mass
 - Object must move as unit
- Properties in class Body
 - Position
 - Linear Velocity
 - Angular Velocity
 - Body Type
- There are 3 body types
 - Static: Does not move
 - **Kinematic**: Moves w/o force
 - **Dynamic**: Obeys forces





Body in box2d

- Represents a single point
 - Center of the object's mass
 - Object must move as unit
- Properties in class Body
 - Position
 - Linear Velocity
 - Angular Velocity
 - Body Type
- There are **3 body types**
 - Static: Does not move
 - **Kinematic**: Moves w/o force
 - **Dynamic**: Obeys forces

- Kinematic is rarely useful
 - Limited collision detection
 - Only collides w/ dynamics
 - Does not bounce or react
- Application: Bullets
 - Light, fast-moving objects
 - Should not bounce





Forces vs. Impulses

Forces

- Instantaneous push
 - To be applied over time
 - Gradually accelerates
 - Momentum if sustained

Impulses

- Push with duration
 - To be applied in one frame
 - Quickly accelerates
 - Immediate momentum



Impulse = Force x Time

Forces vs. Impulses

Forces

- Instantaneous push
 - To be applied over time
 - Gradually accelerates
 - Momentum if sustained

Impulses

- Push with duration
 - To be applied in one frame
 - Quickly accelerates
 - Immediate momentum



Impulse = Force x 1 Sec

in Box2D

Force and Acceleration

- What do we need to compute motion?
 - $\Delta p = v\Delta t = v_0\Delta t + \frac{1}{2}a(\Delta t)^2 = v_0\Delta t + \frac{1}{2}(F/m)(\Delta t)^2$
 - So depends on Force, current velocity and mass
- Where does that mass come from?
 - Class Body has a getter, but no setter!
 - It comes from the Fixture class
 - Fixture gives *volume* to body
- Will revisit this later with collisions

Force and Acceleration

- What do we need to compute motion?
 - $\Delta p = v\Delta t = v_0\Delta t + \frac{1}{2}a(\Delta t)^2 = v_0\Delta t + \frac{1}{2}(F/m)(\Delta t)^2$
 - So depends on Force, current velocity and mass
- Where does that mass come from?
 - Class Body has a getter, but no setter!
 - It comes from the Fixture class
 - Fixture gives *volume* to body
- Will revisit this later with collisions

Four Ways to Move a Dynamic Body

• Forces

- applyForce (linear)
- applyTorque (angular)

• Impulses

- applyLinearImpulse
- applyAngularImpuls
 e

• Velocity

- setLinearVelocity
- setAngularVelocity

Translation

11

• setTransform





Four Ways to Move a Dynamic Body





Example: box2d Demo



box2d

at cornell university

Example: box2d Demo



Four Ways to Move a Dynamic Body

• Forces

- applyForce (linear)
- applyTorque (angular)

Impulses

- applyLinearImpulse
- applyAngularImpuls e

• Velocity

- setLinearVelocity
- setAngularVelocity

Translation

15

• setTransform

Must Cap Velocity



Basic Structure of a Update Loop

```
public void update(float dt) {
    // Apply movement to relevant bodies
    if (body above or equal to max velocity) {
body.setLinearVelocity(maximum
velocity);
    } else {
        body.applyForce(force)
        body.applyTorque(torque)
    }
    // Use physics engine to update positions
world.step(dt,vel iterations,pos iterations);
}
16
```

Basic Structure of a Update Loop

```
public void update(float dt) {
    // Apply movement to relevant bodies
    if (body above or equal to max velocity) {
body.setLinearVelocity(maximum
velocity);
    } else {
        body.applyForce(force)
        body.applyTorque(torque)
    }
    // Use physics engine to update positions
world.step(dt,vel iter
                                            ons);
                           Multiple times to
                           improve accuracy
                                             gamedesig
17
                         box2d
```

Basic Structure of a Update Loop



Collision Objects in box2d

Shape

- Stores the object geometry
 - Boxes, circles or polygons
 - Must be convex!
- Has own coordinate space
 - Associated body is origin
 - Unaffected if body moved
 - Cannot be resized later
- Also stores object density
 - Mass is area x density

Fixture

- Attaches a shape to a body
 - Fixture has only one body
 - Bodies have many fixtures
- Cannot change the shape
 - Must destroy old fixture
 - Must make a new fixture
- Has other properties
 - **Friction**: stickiness
 - **Restitution**: bounciness



```
// Create a body
definition
// (this can be reused)
bodydef = new BodyDef();
bodydef.type = type;
bodydef.position.set(posi
tion);
bodydef.angle = angle;
// Allocate the body
bodv1 =
world.createBody(bodydef)
;
// Another?
bodydef.position.set(posi
tion2);
20
                              box2d
body2
         a = \pm a D = a = (l = a = a = f)
```





// Create two triangles
as shapes shape1 = new
PolygonShape().; shape2 = new
PolygonShape(); shape1.set(verts1);
shape2.set(verts2); // Create a fixture fixdef = new
FixtureDef(); fixdef.density = density; // Attach the two shapes to body fixdef.shape = shape1; fixture1 = body1.createFi
xture(fixdef); gamedesigninitiative box2d fixdef.shape = shape2



```
// Create a body definition
```

```
// (this can be reused)
```

bodydef = new BodyDef();

bodydef.type = type;

```
bodydef.position.set(posi
tion);
```

bodydef.angle = angle;

// Allocate the body

body1 =
world.createBody(bodydef)
;

// Another?

```
bodydef.position.set(posi
tion2);
24
body2 =
```

```
// Create two triangles
as shapes
    shape1 = new
PolygonShape().;
    shape2 = new
PolygonShape();
    shape1.set(verts1);
shape2.set(verts2);
    // Create a fixture definition
    fixdef = new
FixtureDef();
    fixdef.density = density;
    // Attach the two shapes to body
    fixdef.shape = shape1;
    fixture1 = body1.createFi
xture(fixdef);
                              gamedesigninitiative
box2d
    fixdef.shape = shape2
```

Observations on Fixture Parameters

- **Density** can be anything **non-zero**
 - The higher the density the higher the mass
 - Heavier objects are harder to move
- Friction should be within 0 to 1
 - Can be larger, but effects are unpredictable
 - Affects everything, even manual velocity control
- **Restitution** should be within 0 to 1
 - A value of 0 means no bounciness at all
 - Unpredictable with manual velocity control

A Word on Units

box2d



• Adjust scale so 1 default



Example: Box2D Demo



box2d

at cornell university

Example: Box2D Demo



How Do We Find the Shape?

- Do not try to *learn* boundary
 - Image recognition is hard
 - Hull will have **many** sides
- Have artists draw the shape
 - Cover shape with triangles
 - But can ignore interiors
 - Keep # sides small!
- Store shape in another file
 - Do not ruin the art!
 - Need coordinates as data





Data-Driven Design

120,2 130,4 125,50 150,65 160,100 150,110 125,80 140,200 130,200 120,110 . . .

character.jpg

character.shape



Custom Collisions: ContactListeners

- Special listener attached to world object
 - Reacts to any two **fixtures** that collide
 - Allow you to *override* collision behavior
 - Or you can *augment* collision behavior
- Two primary methods in interface
 - **beginContact**: When objects first collide
 - **endContact**: When objects no longer collide
- **Example**: Color changing in box2d demo

Collision is About Fixtures!



- Capsule obstacle is two circles and rectangle
 - Allows smooth motion while walking
 - Feet do not get hung up on surfaces
- But may register **multiple collisions**!



Aside: What is an Obstacle?

- GDIAC extensions include the Obstacle class
 - Combine body and fixture into one class
 - Defined as shapes: Box, Wheel, Polygon, Capsule
- Designed to make collisions easier
 - Each fixture has an associated user data object
 - Used to define the **source** of the collision
 - Obstacles assign themselves as this source

Sessentially box2d on "training wheels" gamedesi



Collision Filtering

- FixtureDef has a Filter attribute
 - categoryBits: Defines what can collide with it
 - maskBits: Defines what it can collide with
 - groupIndex: Collision group (overrides bits)

• Example:

- Fixture A category x001, Fixture B category x010
- Mask $\times 101$ or $\times 001$ only collides with A
- Mask x011 collides with both A and B

Collision Filtering

• FixtureDef has a Filter attribute

- categoryBits: Defines what can collide with it
- maskBits: Defines what it can collide with
- groupIndex: Collision group (overrides bits)

• Example:



How about Sort-of-Filtering?

- Want a non-sensor object where
 - We always **detect** the collision
 - But sometimes ignore the **restitution**
- Method beginContact has a Contact parameter
 - Manages the physics while it resolves collision
 - Can call the method contact.isEnabled(false)
 - Turns off collision; **endContact** is never called
- See tutorials for "anatomy of a collision"
 ³⁶ https://www.iforce2d.net/b2dtut/collumburghtumbersity

Recall: Swept Shapes

- False positives happen if:
 - Two objects are moving
 - Swept shapes intersect at different intersection times
- What if only one moving?
 - Swept intersects stationary
 - So no false positives
- Change reference frames
 - Keep one shape still
 - Move other in new coords



the gamedesigninitiative at cornell university

Recall: Swept Shapes



More Collisions: RayCasting



- Method rayCast in world
 - Give it start, end of ray
 - Also a RayCastCallback
 - Executed when call step
- Invoked on all collisions
 - Not just the first on
 - Does not return in order!
 - This is for optimization



The RayCastCallback Interface

float reportRayFixture(Fixture
fixture, // Fixture found

Vector2 point, // Collision point

Vector2 nom, // Collision normal

- Fraction is how far along ray (0 = start, 1 = end)
- fle First collision is one with lowest fraction tion of ray
 - But be prepared for larger fractions first
- Return value is optimization to **limit search**
 - Ignores collisions with fraction later than return

The RayCastCallback Interface



• Ignores collisions with fraction later than return

AABB Queries



• Bounding Box queries

- Find all fixtures in box
- Must be *axis aligned*
- Rotation not allowed
- Similar to raycasting
 - Provide callback listener
 - Call step method in world
 - Prepare for many matches
- Application: selection
 - See Ragdoll Demo



Some Words on Joints

- Joints connect **bodies**
 - Anchors can be offset body
 - Coordinates relative to body
- Are affected by **fixtures**
 - Fixtures prevent collisions
 - Limit relative movement
- Must control with forces
 - Manual velocity might violate constraints
 - Use force or impulse





The Distance Joint

- Extremely common joint
 - Separates by a fixed amount
 - Good for ropes/grappling
- Can be **hard** or **soft**
 - Hard: Strong but very brittle
 - Soft: Stretchy but very weak
- Softness set in the **joint def**
 - Damping, frequency values
 - Turns the joint into a **spring**
 - **Damping**: Use <1 to soften
 - **Frequency**: Spring oscillation



The Distance Joint

- Extremely common joint
 - Separates by a fixed amount
 - Good for ropes/grappling
- Can be **hard** or **soft**

Older versions of box2d have a rope joint. This is **deprecated** in favor of soft distances.

• Turns the joint into a **spring**

- **Damping**: Use <1 to soften
- **Frequency**: Spring oscillation

Hard Distance

Other Joint Types

Revolute



- Joint binds at one point
- Both translate together
- But rotate independently



Weld

- Joint binds at one point
- Both translate together
- Both rotate together



Other Joint Types



- Joint binds with a "track"
- Both rotate together
- But translate along track

- Joint binds through portals
- Pulling one raises the other
- Distance w/ "teleportation"



Making a Rope: The Simple Way





Making a Rope: The Better Way

Web of springy **distance** joints with revolute joints at the end



Keeps rope strong but flexible!



Summary

- box2d support motion and collisions
 - Body class provides the motion
 - Fixture, Shape classes are for collisions
- Multiple ways to control a physics object
 - Can apply forces or manually control velocity
 - Joint constraints work best with forces
- Collisions are managed by callback functions
 - Invoked once you call the world step method
 - Collisions are processed per fixture, not per body

box2d

