Introduction To Section 2

In this section you will learn about…

- Constant acceleration (e.g. local gravity).
- Variable acceleration (drag, Magnus effect).
- Creating methods for linear forces.
- **F = ma** for turning forces into accelerations.
- Calculating velocity & position from acceleration.

Overview Of Pattern

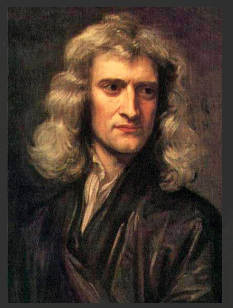
- Unbalanced forces lead to accelerations.
- Accelerations lead to a change in velocity.
- Velocity determines the change in position.

Newton's First Law

In this video…

- Introducing Sir Isaac Newton
- Newton's First Law
- About the **FixedUpdate()** loop
- Updating **transform.position**

Isaac Newton



Newton's First Law

- "When viewed in an inertial reference frame, an object either remains at rest or continues to move at a constant velocity, unless acted upon by an external force."

http://en.wikipedia.org/wiki/Newton%27s_laws_of_motion

Using FixedUpdate()

- Called every 0.02s (20ms) by default.
- That's a **consistent** 50 times a second.
- **Time.deltaTime** returns the current value.

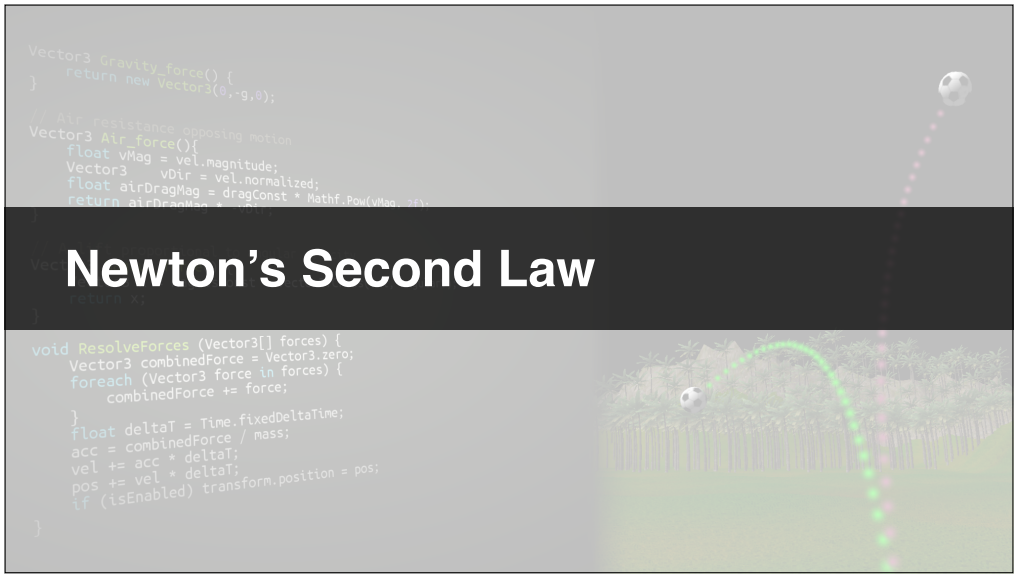
http://docs.unity3d.com/Manual/ExecutionOrder.html

Make Your Ball Move

- **transform.position** = displacement from origin
- **v** = average velocity this FixedUpdate()
- **deltaS** = displacement change
- **deltaS = v * Time.deltaTime**
- Set **transform.position** inside **FixedUpdate()**

Create A Force List

- **using System.Collections.Generic;**
- Create a public list of forces, **forceVectorList;**
- Create **AddForces ()** so sum the forces.
- **Debug.LogError ()** if we have a net force.
- Otherwise continue to update the position.



2nd Law Defined

- "The vector sum of the forces **F** on an object is equal to the mass m of that object multiplied by the acceleration vector a of the object: **F** = m **a**."
- **a** = **F** / m

Write Code For Acceleration

- Publicly expose float **mass**
- Create **UpdateVelocity ()** method
- Modify **velocityVector** every FixedUpdate
- Remove **Debug.LogError()** code

Newton's Third Law

In this video…

- Defining Newton's Third Law.
- Examples of why it's important to consider.
- Introducing our trail drawing code.
- Switching trail direction.
- Recap of Newton's laws

Newton's Third Law

- "When one body exerts a force on a second body, the second body simultaneously exerts a force equal in magnitude and opposite in direction on the first body."

Modify Your Force Trails

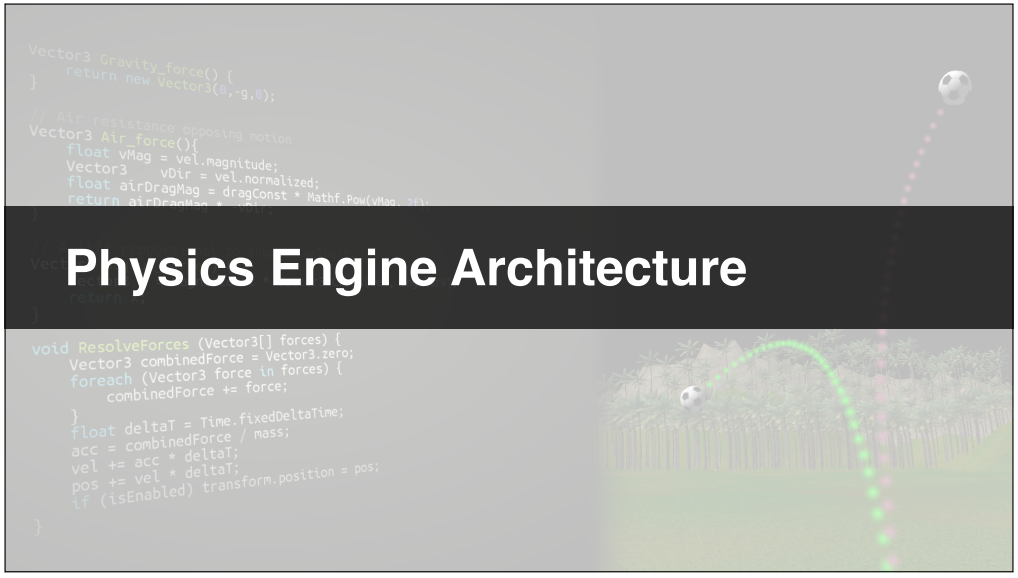
- Turn them round so they face the other way.
- Experiment with particle systems alternative?
- Remember the other force is **always** there.

Recapping Newton's Laws

- **1st Law:** if (**SumForces** == **0**) then **deltaV** = **0**
- **2nd Law: acceleration**= **netForce** / mass
- **3rd Law: forceAonB** = - **forceBonA**

Valid for a huge range of phenomena. Care below 10^-8m or above 10^8m/s (GPS, superconductors, etc)

Physics Engine Architecture

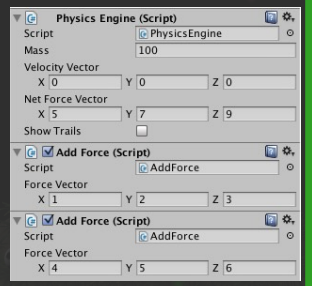


In this video…

- What we're trying to achieve.
- Why it's important to get this "right".
- Re-orgainse list iteration for variable forces.
- Bring DrawTrails.cs into PhysicsEngine.cs.
- Move the forces into separate components.

Refactor Your Code

- Move **DrawForces.cs** into **PhysicsEngine.cs**.
- Create **AddForce.cs** script, add two to ball.
- Try applying forces from scripts.



SI Units & Dimensions

In this video…

- About Units and Dimensions.
- Why it's worth thinking about them.
- Where to annotate our units.
- Prepare our **RocketEngine.cs** class.

The 7 Base Units

- One SI base unit for each dimension.
- We're going to use SI base units (or multiples).
- Both sides of any equation must match.
- 1 meter = 4 feet COULD be right
- 1 meter = 4 kilograms couldn't

Where To Annotate Units?

	Inspector	code readability	IDE pop-up
// comment	None	Excellent	None
/// summary	None	Poor	Excellent
[Tooltip…]	Excellent	Poor	None
suffix _ms^-1	Poor	Poor	None

Annotate All Your Units

- Check all units are annotated neatly.
- Do this for both RocketEngine.cs
- … and for PhysicsEngine.cs

http://en.wikipedia.org/wiki/SI_base_unit

In this video…

- The thrust of a rocket engine.
- Modelling rocket engines and fuel burn.
- An introduction to “delta V”.

http://en.wikipedia.org/wiki/Rocket_engine

Write float FuelThisUpdate()

- Return fuel used during this Time.deltaTime in kg
- Annotate private variables with units.
- Burn 1 metric tonne of fuel in virtual shuttle.
- Check that “delta V” is 2.23 m s^-1.
- Check same dela V at lower thrustPercent

Improving Code Architecture

Modelling Gravity

In this video…

- Review the universal gravitation equation*
- Create a method for it in Unity.
- Check we get realistic force values.

*http://en.wikipedia.org/wiki/Gravitational_constant

Write the equation into Unity

- No simple power operator like ** or ^ I'm afraid.
- Use **Mathf.pow(distance, 2f);** for square.
- Test you get a **force** of around 9.8 Newtons.

Back Down To Earth

```
void ResolveForces (Vector3[] forces) {
    Vector3 combinedForce = Vector3.zero;
    foreach (Vector3 force in forces) {
        combinedForce += force;
    }
    float deltaT = Time.fixedDeltaTime;
    acc = combinedForce / mass;
    vel += acc * deltaT;
    pos += vel * deltaT;
    if (isEnabled) transform.position = pos;
}
```

In this video…

- Setup a football field "on Earth".
- Test gravitation matches real values.
- Explore parabolic flight.
- Set scene for air resistance.

Score A Goal

- Start in the centre of the pitch.
- Use a single rocket engine to land in goal box.
- Burn for no more than 2 seconds.
- Burn only in one direction.

Modelling Simple Air Resistance

In this video…

- Design a simple formula for air resistance.
- Create an Air Drag component.
- Test flight against reference ball.
- Tweak to get similar results to Unity's engine.

http://en.wikipedia.org/wiki/Drag_(physics)

Create The Fluid Drag Component

- Remember **Mathf.Pow()**
- Test against Unity's reference ball

Script Execution Order

In this video…

- Expand on Unity's script execution order*
- Demonstrate our bug at "high" speeds.
- Show how this relates to fixed update time.
- Move Universal Gravity to separate object.

* http://docs.unity3d.com/Manual/ExecutionOrder.html

Script Execution Order

Script Instance	Script Instance	Script Instance
PhysicsEngine on ball 3	PhysicsEngine on ball 1	UniversalGravitation
Awake ()	-	-
-	Awake ()	-
-	-	Awake ()
Start ()	-	-
-	Start ()	-
-	-	Start ()
FixedUpdate()	-	-
-	FixedUpdate()	-
-	-	FixedUpdate()

Create UniversalGravitation.cs

- Create a game object called Universal Gravitation
- Move gravitation code to UniversalGravitation.cs
- Make it work and test at high speeds

Making A Simple Game

In this video...

- Make a simple hit-the-target game.
- Add a cylinder to represent a rocket launcher.
- Write **Launcher.cs** class.

Write Launcher.cs class

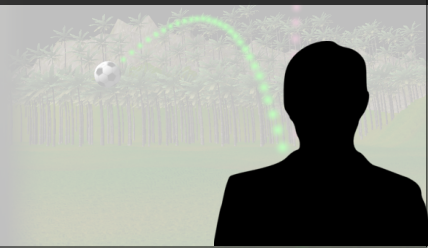
- Take **maxLaunchSpeed** as a parameter.
- Drag-in two sounds, for wind-up and launch.
- Drag-in the ball to be instantiated & launched.
- Launch speed increases while launcher clicked.

Finishing & Tidying Up

In this video...

- Re-introduce Universal Gravitation.
- Review the overall structure of what we've done.

Section Wrap-Up



What You've Learnt

- Basic physics engine architecture / trade-offs.
- A foundation in Newton's Laws of Motion.
- Dimensional checking as a tool.
- How to distill information from Wikipedia etc.
- How to create components for various forces.

Create & Share Rocket Lander

- Give keyboard control over thrust.
- Display fuel, velocity and height to player.
- Challenge to "land" below 5 m s^-1.
- Score is fuel kg remaining.
- Share with us via GameBucket.io.g