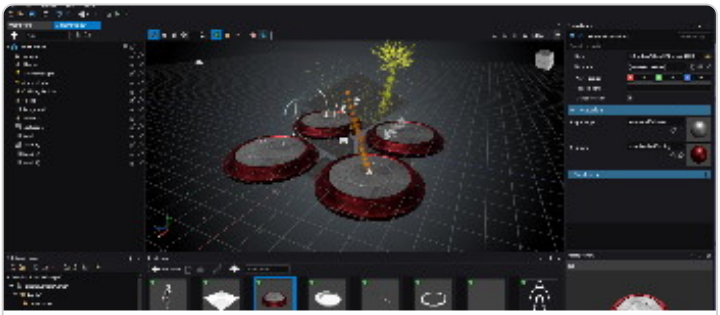
# Stride Game Engine Tutorials

These pages contain tutorials to learn more about the Stride game engine 🎮.

New to Stride? Start with these tutorials to get familiar with the basics of the engine and the Game Studio.

1. 🛠️ Game Studio - The Stride engine comes with an editor called Game Studio, which is the central tool for game and application production in Stride.
2. 🌱 C# Beginner - Covering the beginner principles of using C# when working with the Stride game engine.
3. 📈 C# Intermediate - Diving into intermediate principles of C# programming in Stride, including UI, collisions, and more.



## 🛠️ Game Studio

**12 lessons** **1 hour**

The Stride engine comes with an editor called Game Studio, which is the central tool for game 🕹️ and application production in Stride.

Learn about Stride launcher, main interface, scene management, transforming entities, asset pipelines and more.

🚀 Jump to the Game Studio tutorials

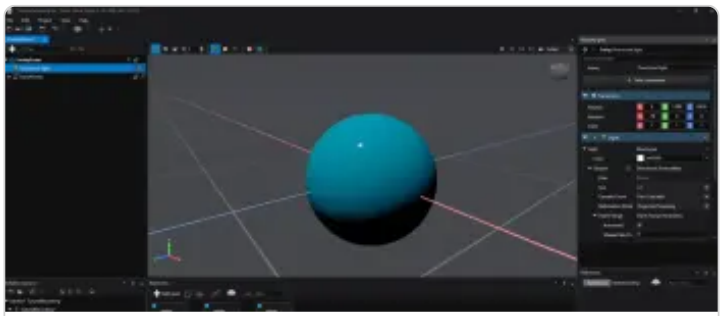

## 🌱 C# Beginner

**15 lessons** **2.5 hours**

These tutorials cover the beginner principles of using C# when working with the Stride game engine 🎮.

Learn about entities, transform positions, editor properties, components, delta time, cloning, keyboard and mouse input and more.

🚀 Jump to the C# beginner tutorials

## 📈 C# Intermediate

**11 lessons** **4 hours**

These tutorials cover various intermediate principles of using C# when working with the Stride game engine 🎮.

Learn more about UI basics, collision triggers, ray-casting, async scripts, scenes, animations, audio, camera and navigation.

🚀 Jump to the C# intermediate tutorials

## ⚡ Quick Tutorials

**1 lesson** **4 minutes**

These quick tutorials provide bite-sized lessons to help you get up to speed with the Stride game engine in no time.

Learn about setting up your first project, basic scripting, simple animations, quick UI tips, and more.

🚀 Jump to the Quick tutorials

# 🛠 Game studio

**12 lessons** **1 hour**

The Stride engine comes with an editor called **Game Studio**. The videos below cover the basics of the UI and how various concepts work inside the editor.

> **NOTE**
> These videos were recorded when Stride was called 'Xenko'. Other than the name and logo change, the UI is almost 100% identical.

## Playlist


## All tutorials

1. Stride launcher
2. Main interface
3. Navigating the scene
4. Scene management
5. Transforming entities
6. Asset pipeline

7. Importing resources
8. Textures
9. Materials
10. Models
11. Physics intro
12. Static colliders

# Stride Launcher 🚀

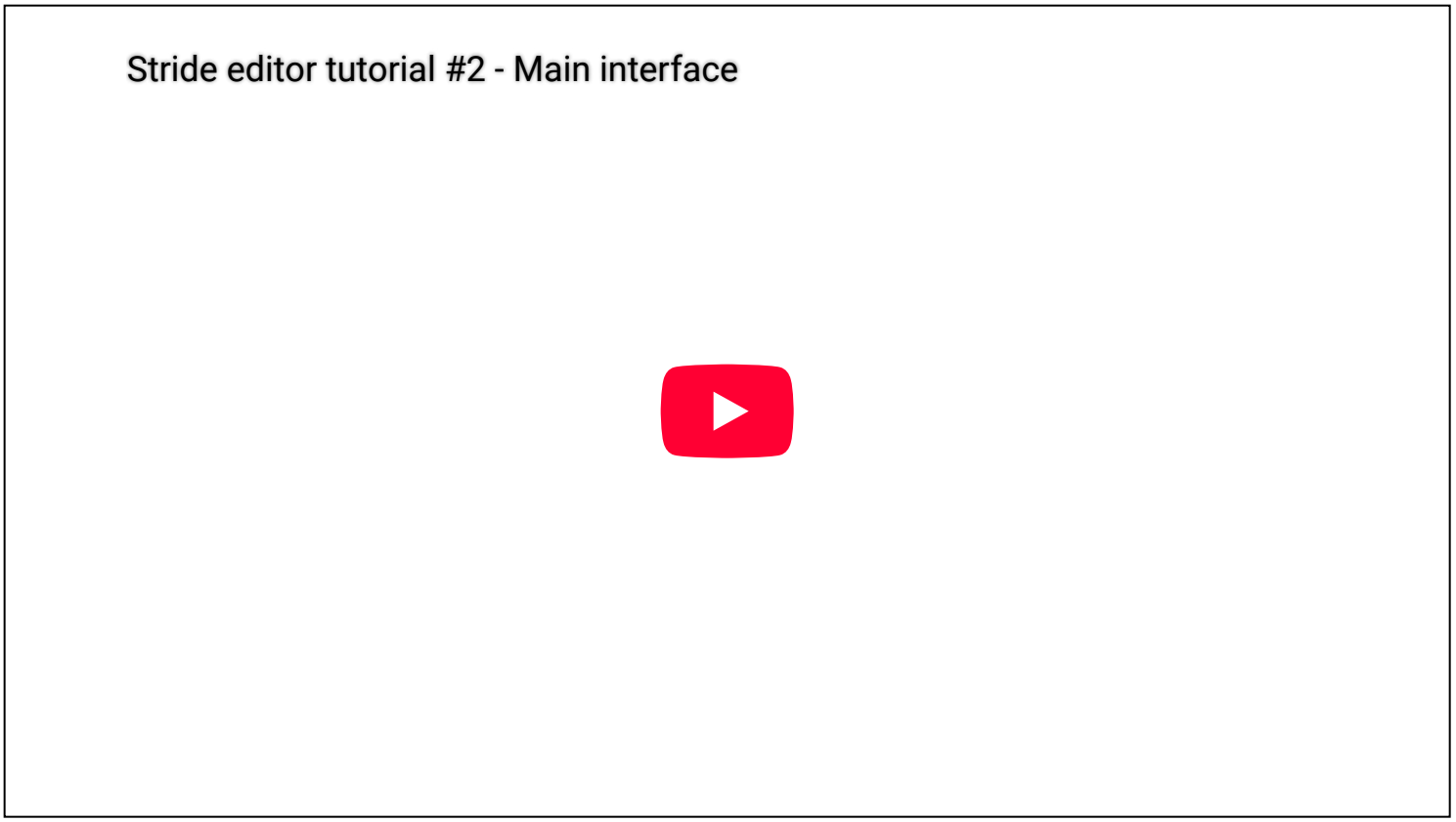
This tutorial explains the Stride Launcher.

> **NOTE**
>
> These videos were recorded when Stride was called 'Xenko'. Other than the name and logo change, the UI is almost 100% identical.

# Main interface

This tutorial explains the main interface of Game studio.

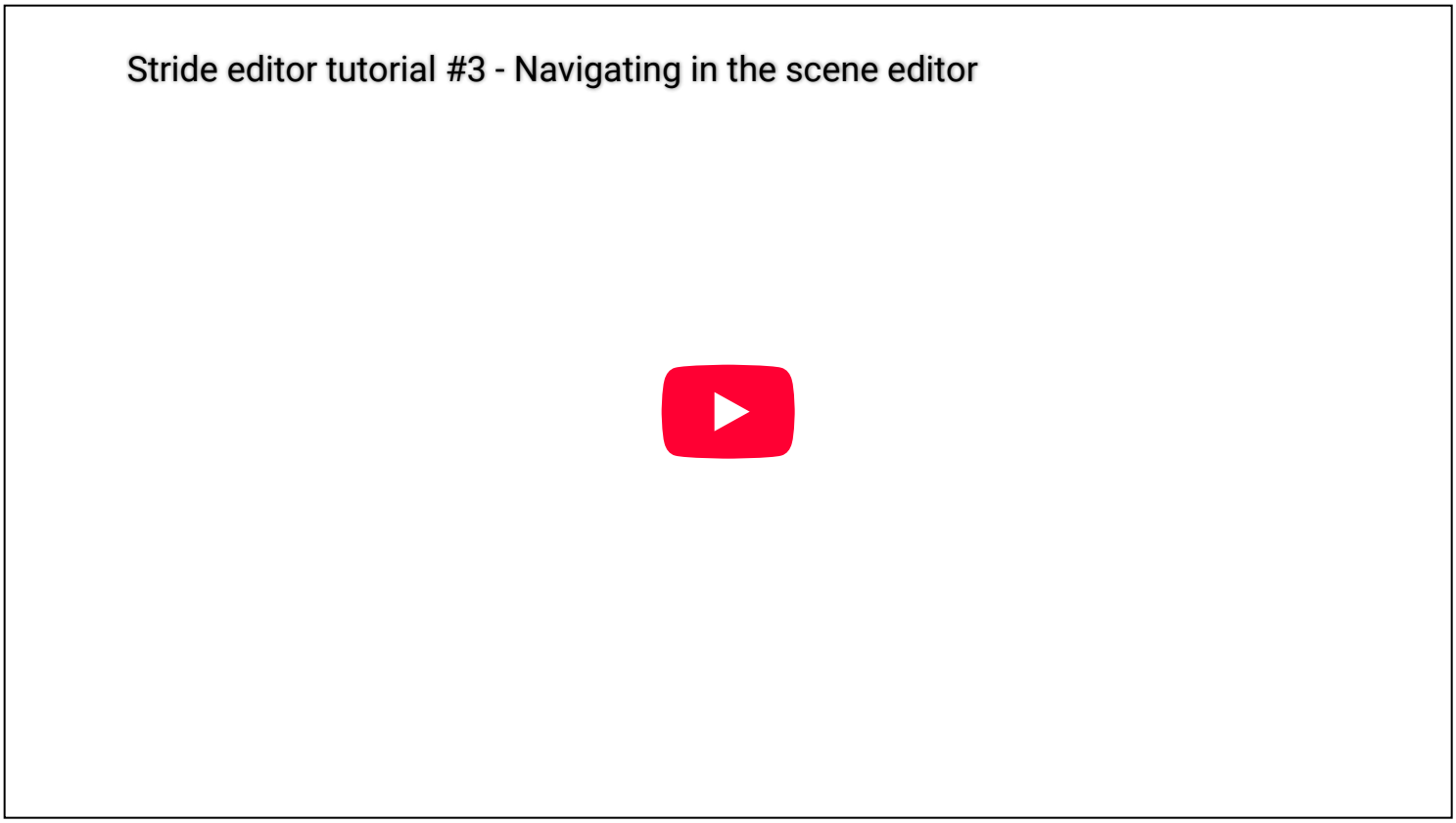
> **NOTE**
>
> These videos were recorded when Stride was called 'Xenko'. Other than the name and logo change, the UI is almost 100% identical.



# Navigating in the scene editor

In this tutorial we learn how to navigate around the scene editor. We also take a look at the various camera options.

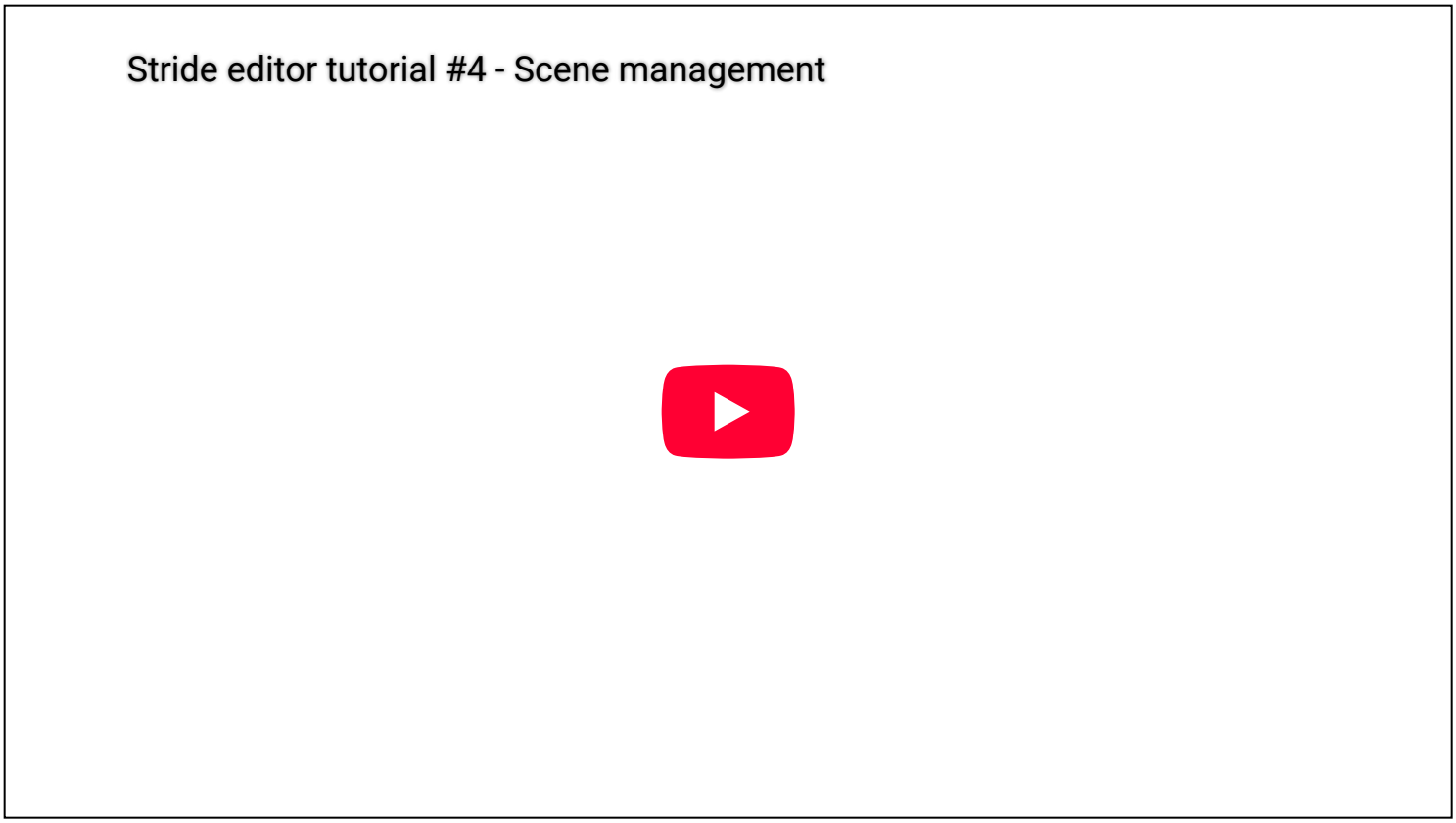
> **NOTE**
>
> These videos were recorded when Stride was called 'Xenko'. Other than the name and logo change, the UI is almost 100% identical.



# Scene management

In this tutorial we learn about scenes, child scenes, hiding and locking scenes and various other concepts that come with scene management in Stride game studio.

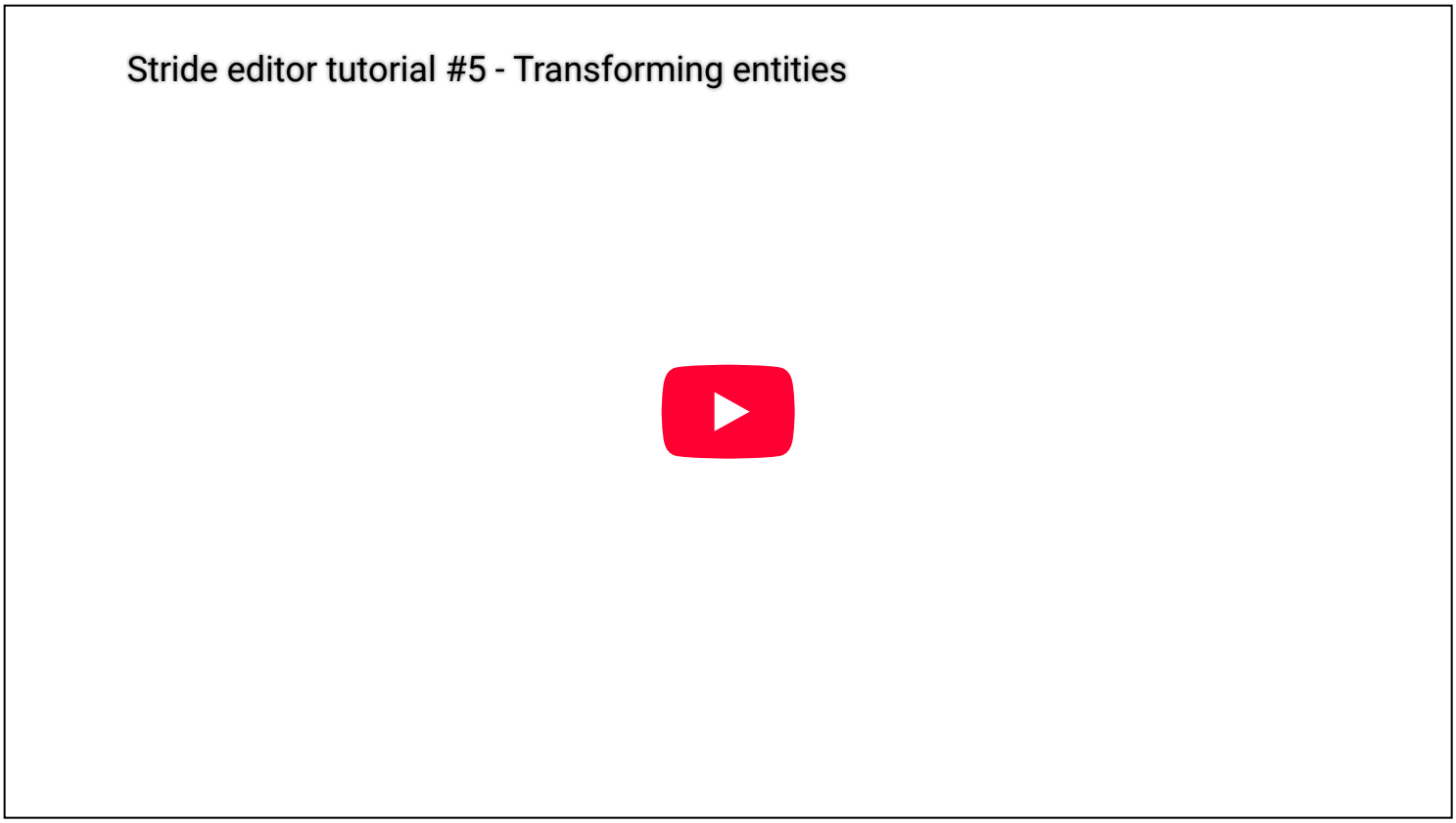
> **NOTE**
>
> These videos were recorded when Stride was called 'Xenko'. Other than the name and logo change, the UI is almost 100% identical.



# Transforming entities

In this tutorial we learn how to transform entities. There is translating, rotation and scaling. We also learn about the right handed coordinate system and how we can use snapping.

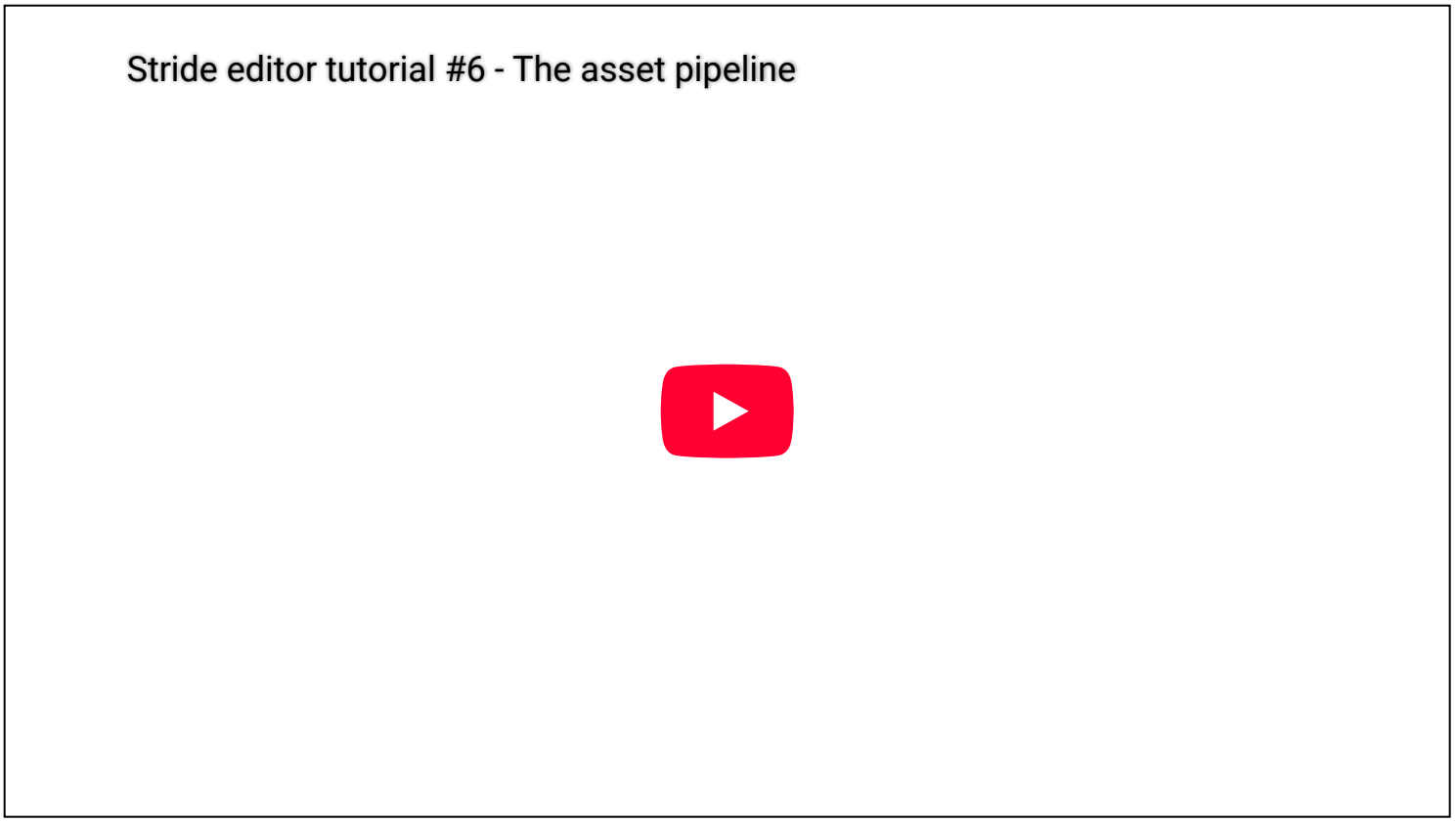
> **NOTE**
>
> These videos were recorded when Stride was called 'Xenko'. Other than the name and logo change, the UI is almost 100% identical.



# Asset pipeline

This more theoretical tutorial covers the general terminology of the Stride asset pipeline. We learn the differences between them and in what stage of the game making process they come up.

> **NOTE**
>
> These videos were recorded when Stride was called 'Xenko'. Other than the name and logo change, the UI is almost 100% identical.



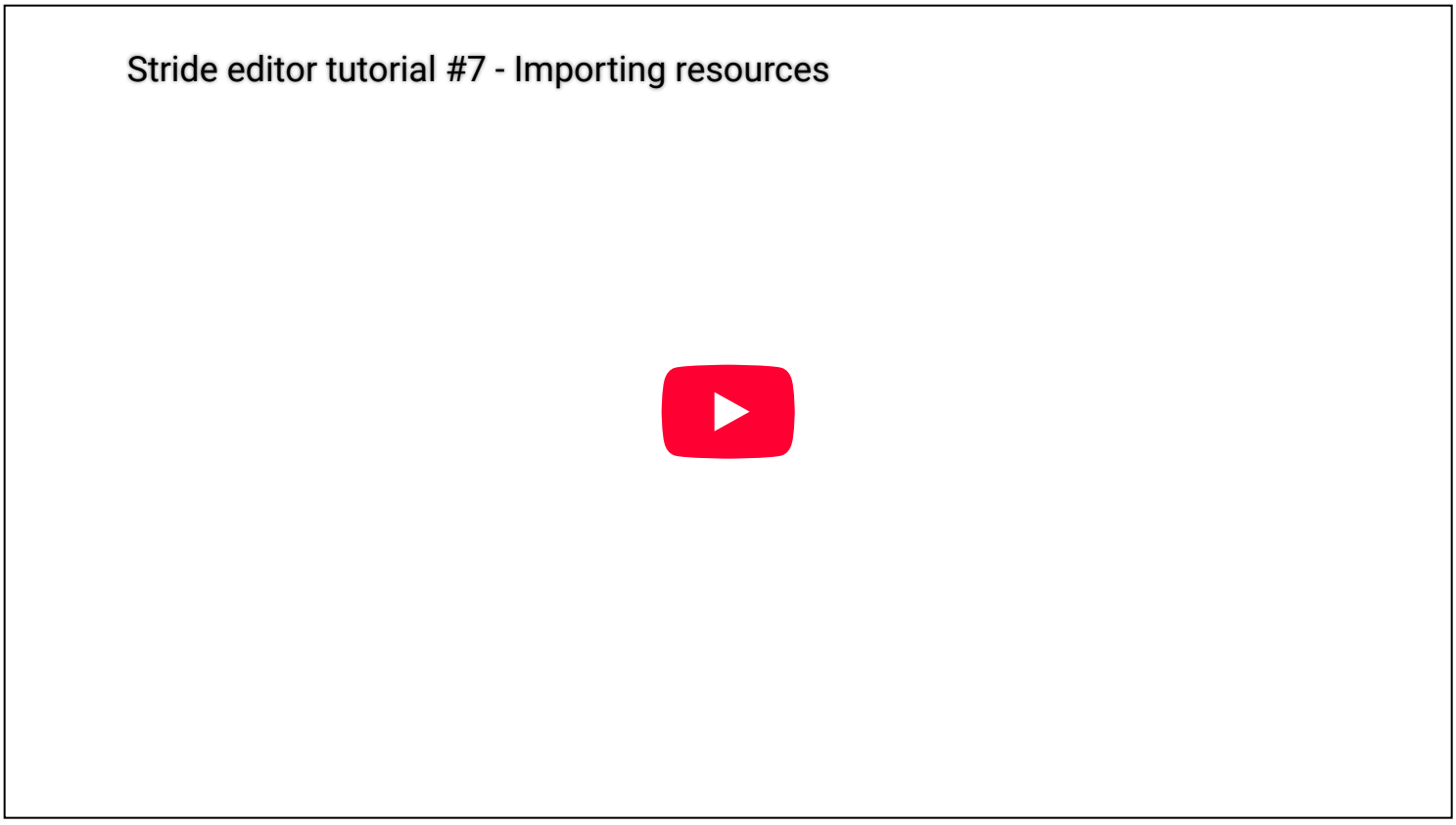
# Importing resources

In this tutorial, we will learn how to import resources into Stride game studio using two different methods:

- Creating an asset via the Asset View, or
- Dragging and dropping files directly from a folder

We also learn how important it is that we keep our resources in a projects resources folder, rather than having them scattered across our computer.

> **NOTE**
>
> These videos were recorded when Stride was called 'Xenko'. Other than the name and logo change, the UI is almost 100% identical.



# Textures

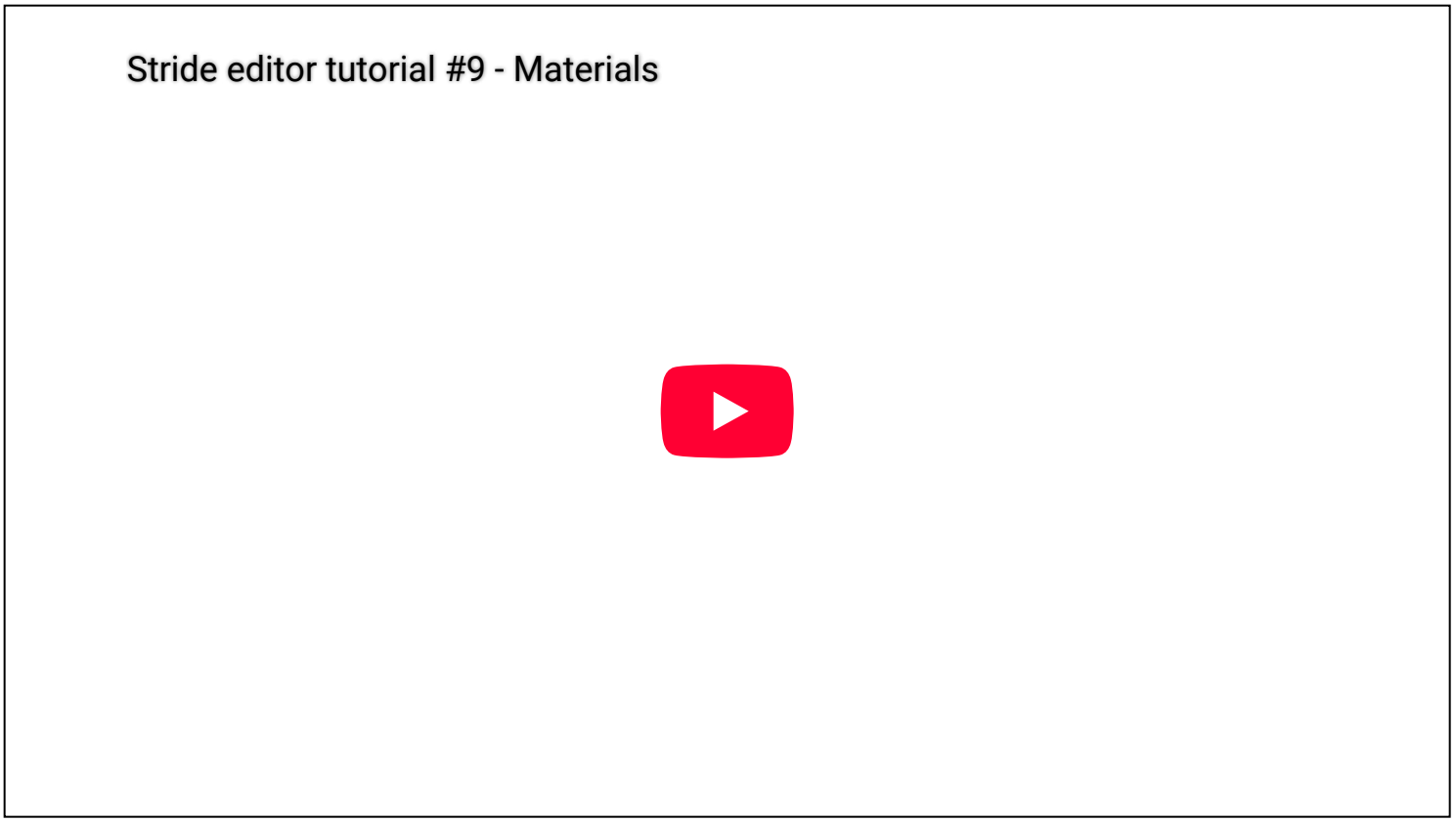
In this tutorial we learn about the different textures types, the options for every texture type and global texture settings.

> **NOTE**
>
> These videos were recorded when Stride was called 'Xenko'. Other than the name and logo change, the UI is almost 100% identical.

# Materials

In this tutorial we learn the basics of creating material assets in Stride game studio. We create a diffuse material and a material that used both diffuse and a normal texture. Since there are so many options for materials we learn about the different properties in a general sense rather than displaying all the possibilities.

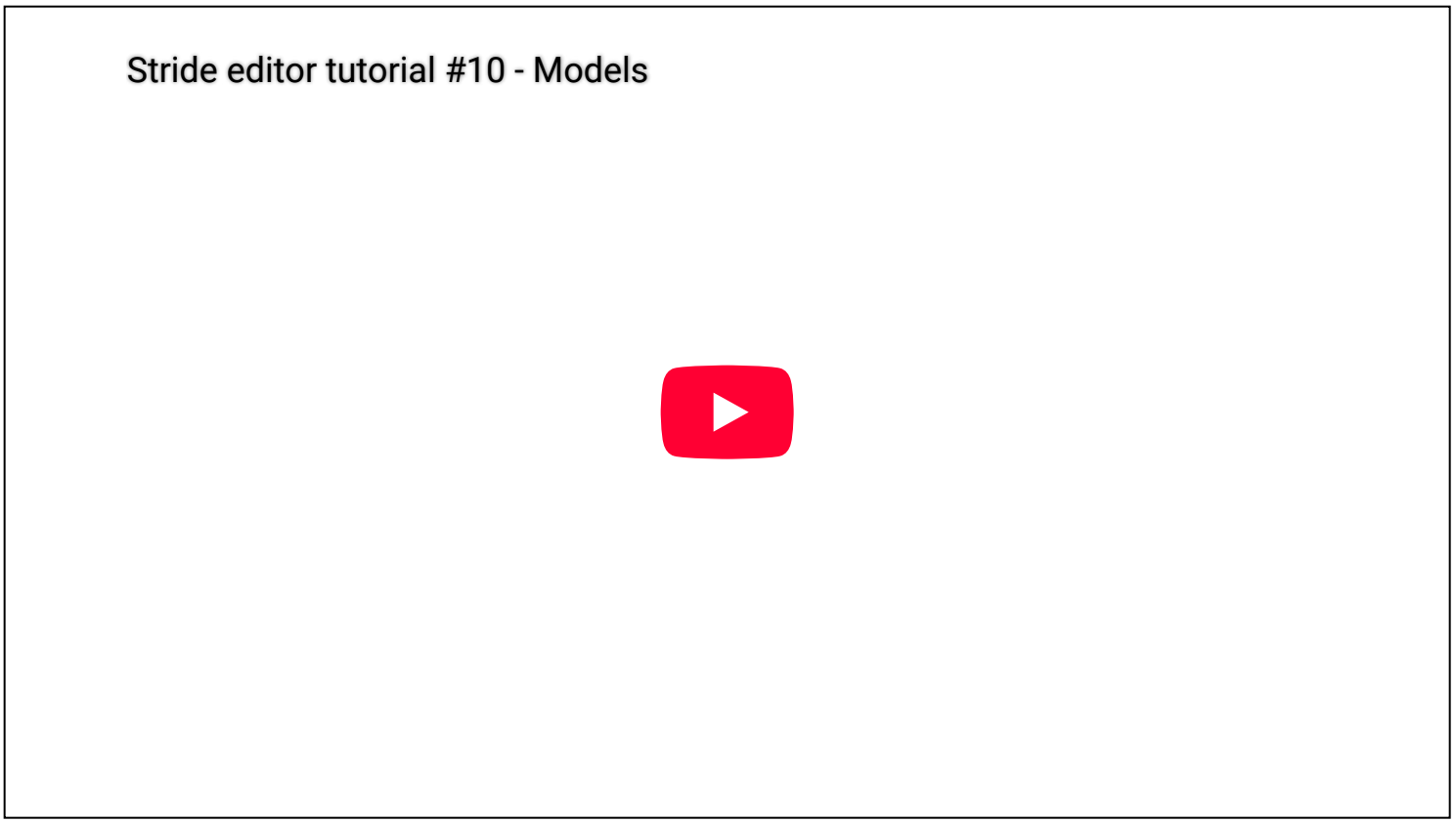
> **NOTE**
>
> These videos were recorded when Stride was called 'Xenko'. Other than the name and logo change, the UI is almost 100% identical.



# Models

In this tutorial we learn about models in Stride game studio. We learn about 5 different import scenarios. Some models have textures which we can automatically apply by importing the right material and textures files.

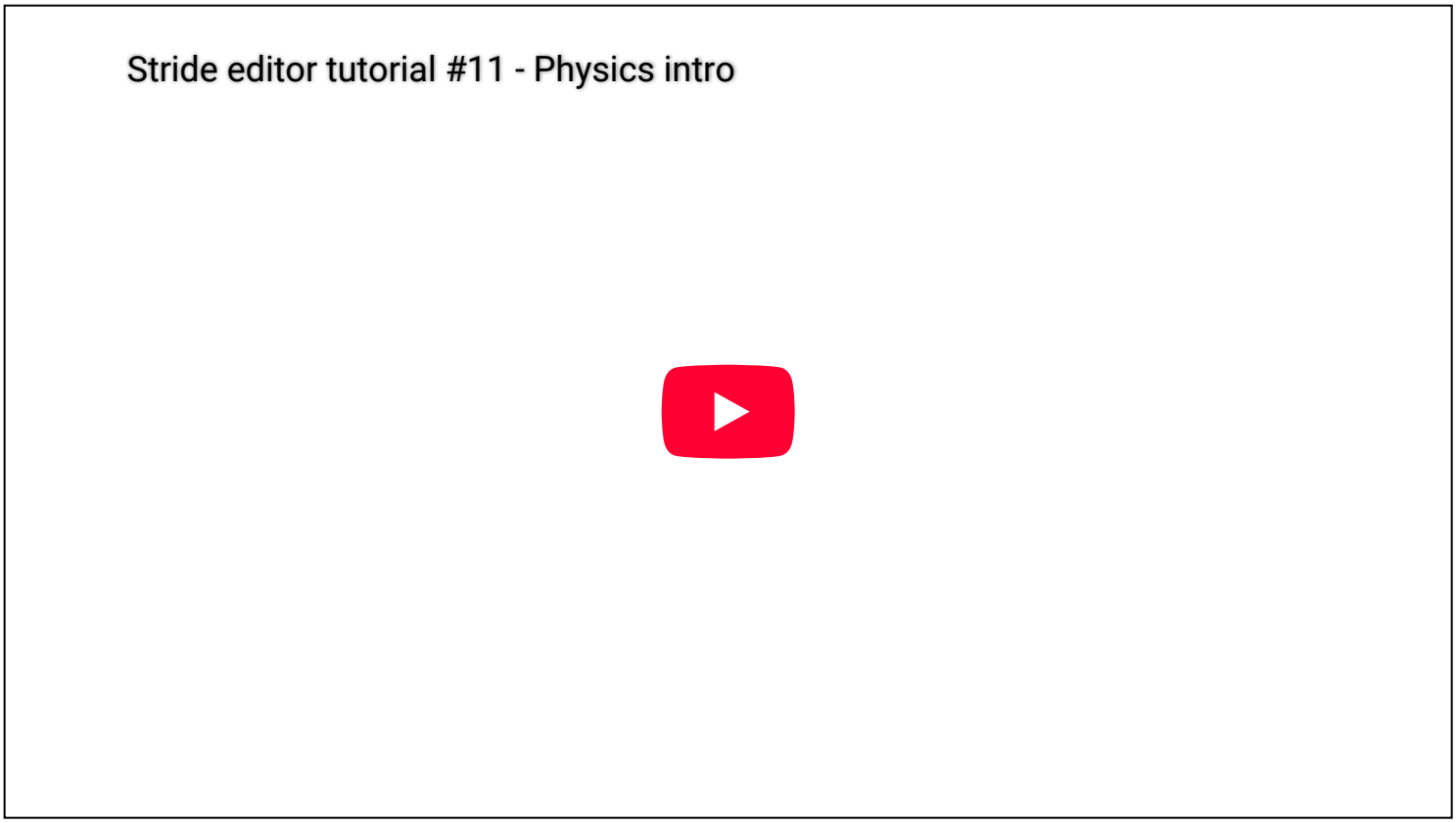
> **NOTE**
>
> These videos were recorded when Stride was called 'Xenko'. Other than the name and logo change, the UI is almost 100% identical.



# Physics intro

In this tutorial we learn about the basics of physics with Stride game studio. We learn that there are 3 different types of collider components and how to debug those in game studio as well as in game.

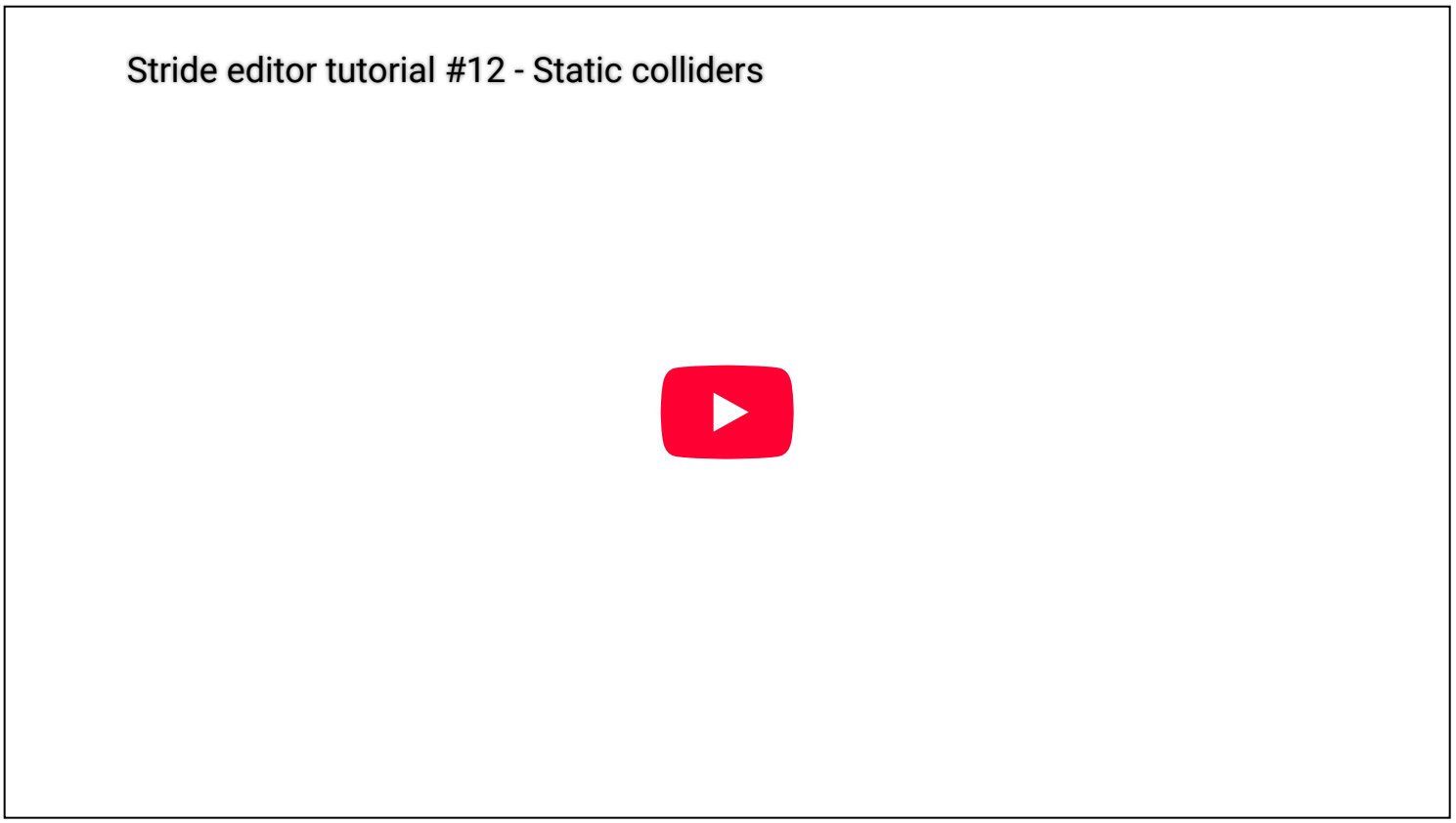
> **NOTE**
>
> These videos were recorded when Stride was called 'Xenko'. Other than the name and logo change, the UI is almost 100% identical.



# Static colliders

In this tutorial we learn about a specific collider type group in Stride: the static colliders. We take a quick tour of its common properties and we learn how to define different collider shapes for an entity. In particular we learn about generated collider hull which can be either convex or concave.

> **NOTE**
>
> These videos were recorded when Stride was called 'Xenko'. Other than the name and logo change, the UI is almost 100% identical.



# C# Beginner

**15 lessons** **2.5 hours**

These tutorials cover the beginner principles of using C# when working with the Stride game engine. Start here if you are new to Stride.

Note: These tutorials do not serve as an introduction to C# itself. While having some coding experience is helpful, it is not mandatory to get started with these tutorials.

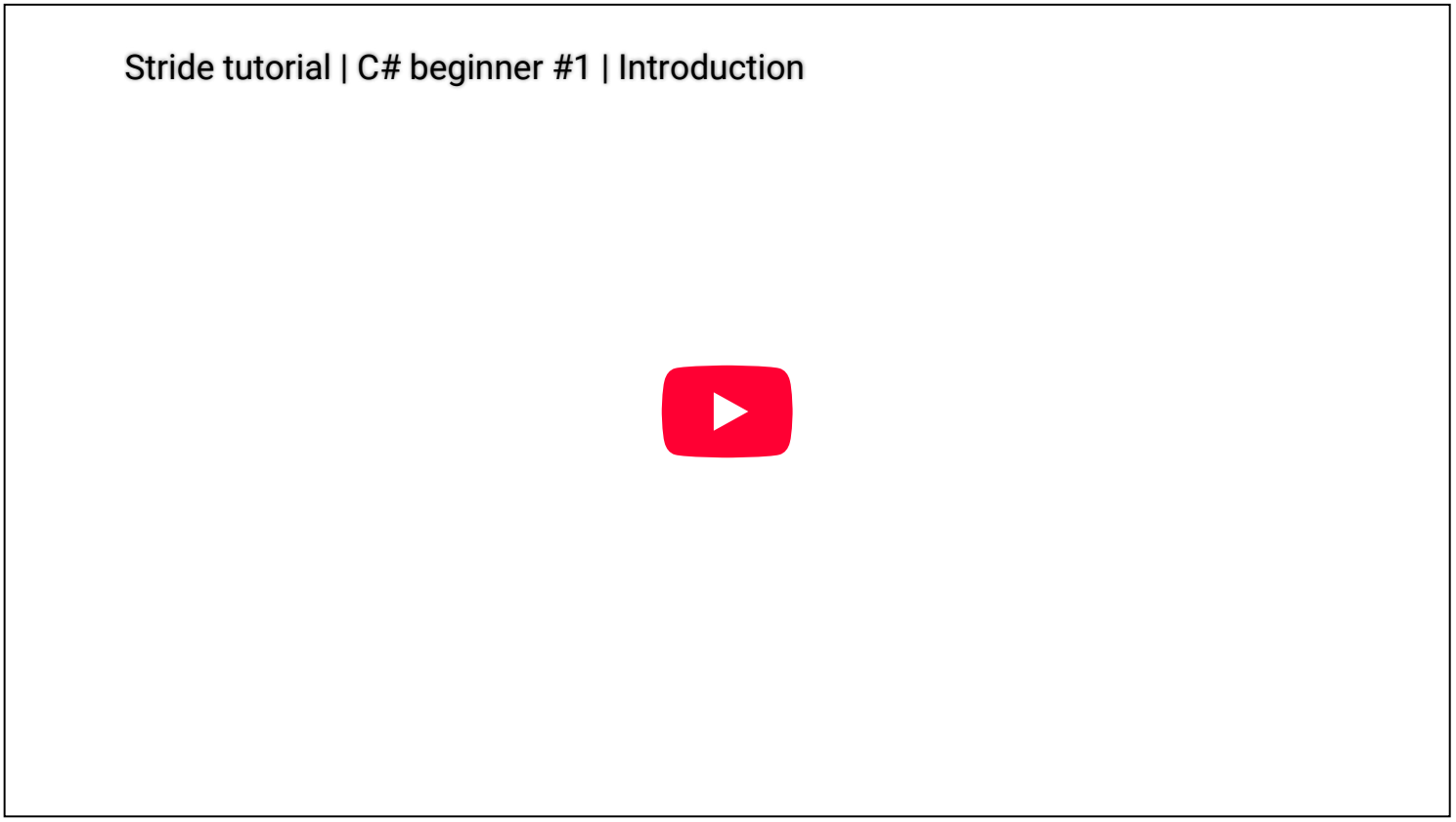
To create the C# beginner tutorial project:

1. Start the Stride launcher
2. Create a new project
3. Select the template: Tutorials -> C# beginner

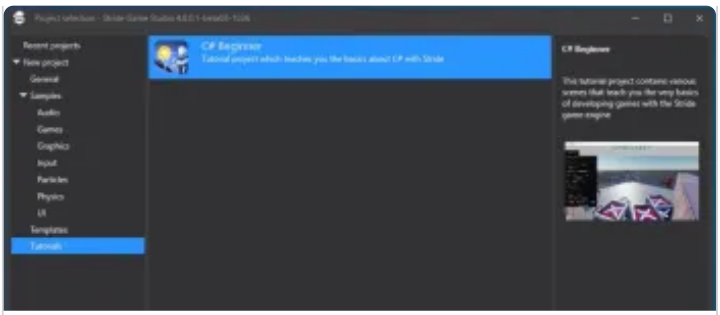
Each scene is loaded as a child scene and demonstrates a sample script.

## Stride C# beginner YouTube tutorial series

All tutorials are accompanied by a YouTube video. You can watch the entire C# Beginners playlist here.




## All tutorials



## Introduction

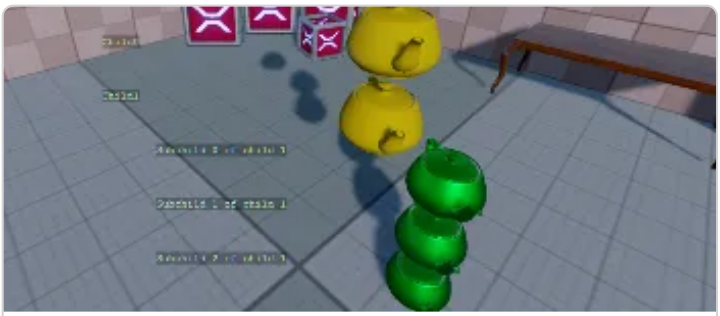
Learn about the C# beginners project template, how entities and components work, different types of scripts, and more.

📺 Watch the Introduction tutorial

## Getting the Entity

Learn how to retrieve the entity, retrieve the parent entity, print debug text, and more.

📺 Watch the Getting the Entity tutorial



## Child entities

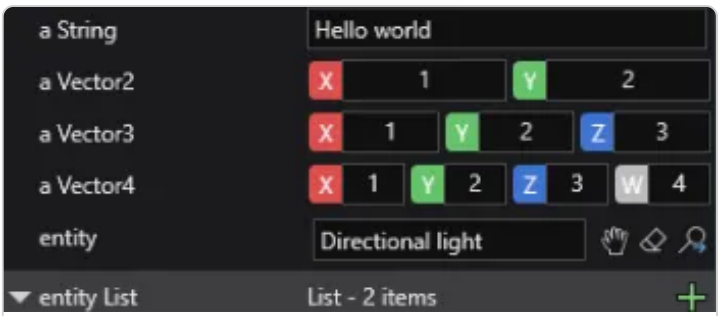
Learn how to get a specific child entity, retrieve children in a list, and access children of children.

📺 Watch the Child entities tutorial

## The transform

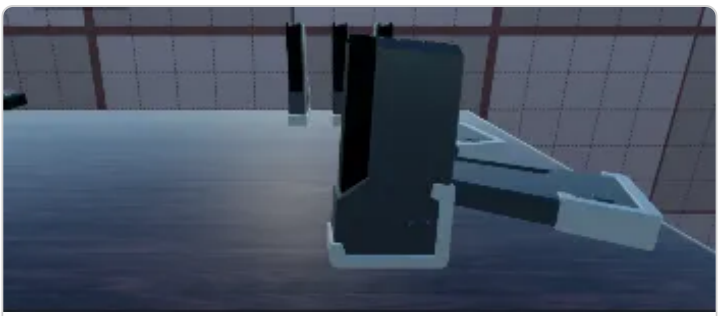
Learn how to access the Transform component, get the local position, and get the world position.

📺 Watch the transform tutorial




## Editor properties

Discover how to define various editor properties, create lists, and hide public properties.



## Getting a component

Understand how to get a component, remove a component, and access methods of other components.

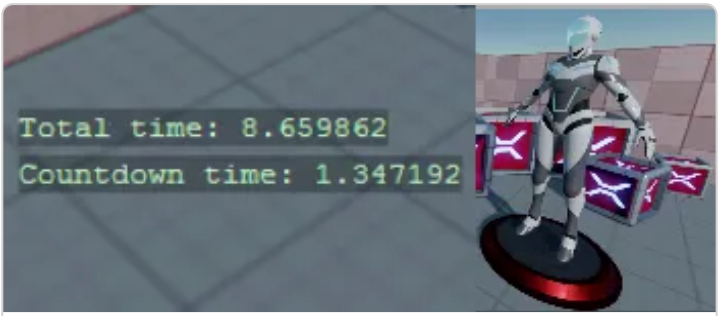
Watch the Editor properties tutorial

Watch the Getting a component tutorial

## Adding a component

Explore adding a component, removing all components of one type, and creating a component if it doesn't exist.

Watch the Adding a component tutorial




## Delta time

Learn how to retrieve delta time, create a simple timer, and make a simple countdown timer.

Watch the Delta time tutorial

## Cloning an entity

Learn how to clone an entity, add an entity to the current scene, and add an entity as a child to a parent entity.

Watch the Cloning an entity tutorial


## Removing an entity

Explore cloning new entities using a timer, removing entities using a timer, and removing an entity from the scene.

Watch the Removing an entity tutorial

## Mouse input


## Keyboard input

Understand how to manage holding down a mouse button, clicking or releasing a mouse button, and using the mouse wheel.

📺 Watch the Mouse input tutorial

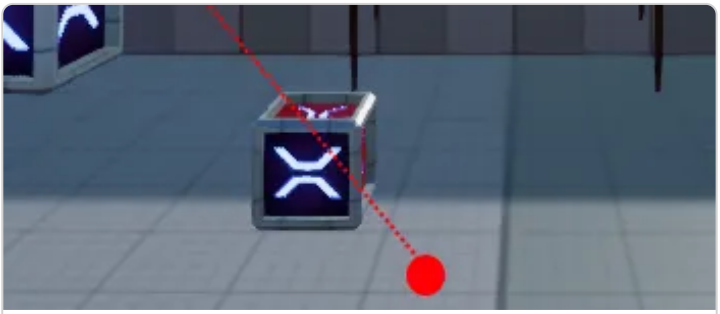
Discover how to handle holding down a key, clicking a key, and releasing a key.

📺 Watch the Keyboard input tutorial

## Virtual buttons

Learn how to define a virtual key configuration, bind input to the configuration, and use the virtual buttons.

📺 Watch the Virtual buttons tutorial



## Linear Interpolation

Explore calculating 'lerp' values, lerping between `Vector3` values, and using random values.

📺 Watch the Linear Interpolation tutorial

## Loading content

Discover how to load content from code, unload content, and attach models to entities.

📺 Watch the Loading content tutorial

## Instantiating prefabs

Learn how to instantiate a prefab, load a prefab from content, and parent a prefab entity.

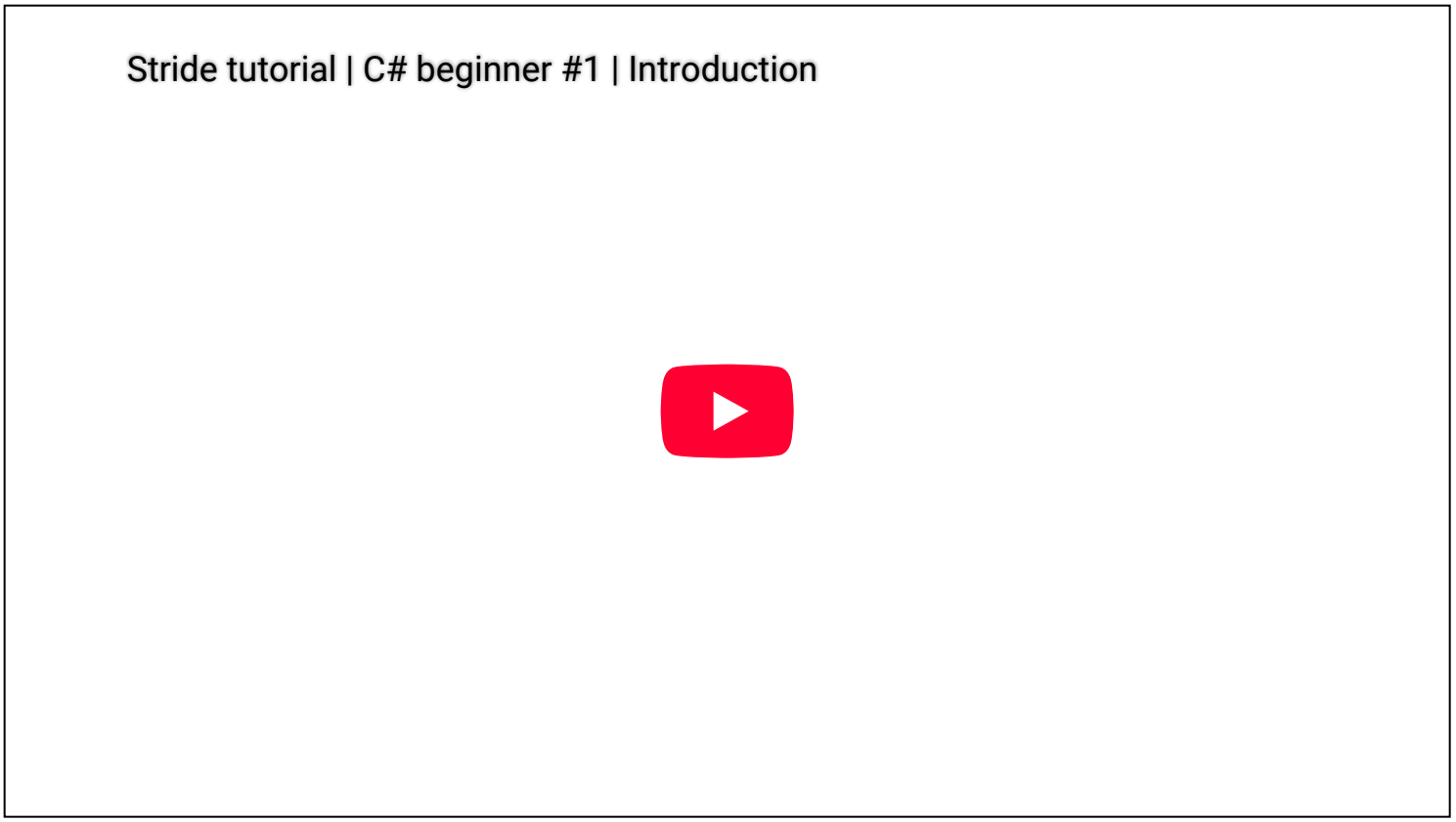
📺 Watch the Instantiating prefabs tutorial

# Introduction

In this tutorial we will learn about the basics of the Stride game engine. We will learn how to setup a new project, how to create entities and how to add components to them. We will also learn how to create scripts and how to debug them.

We will cover these topics:

- How to setup the C# beginners project template
- How entities and components work
- The types of script
- Logging
- Debugging



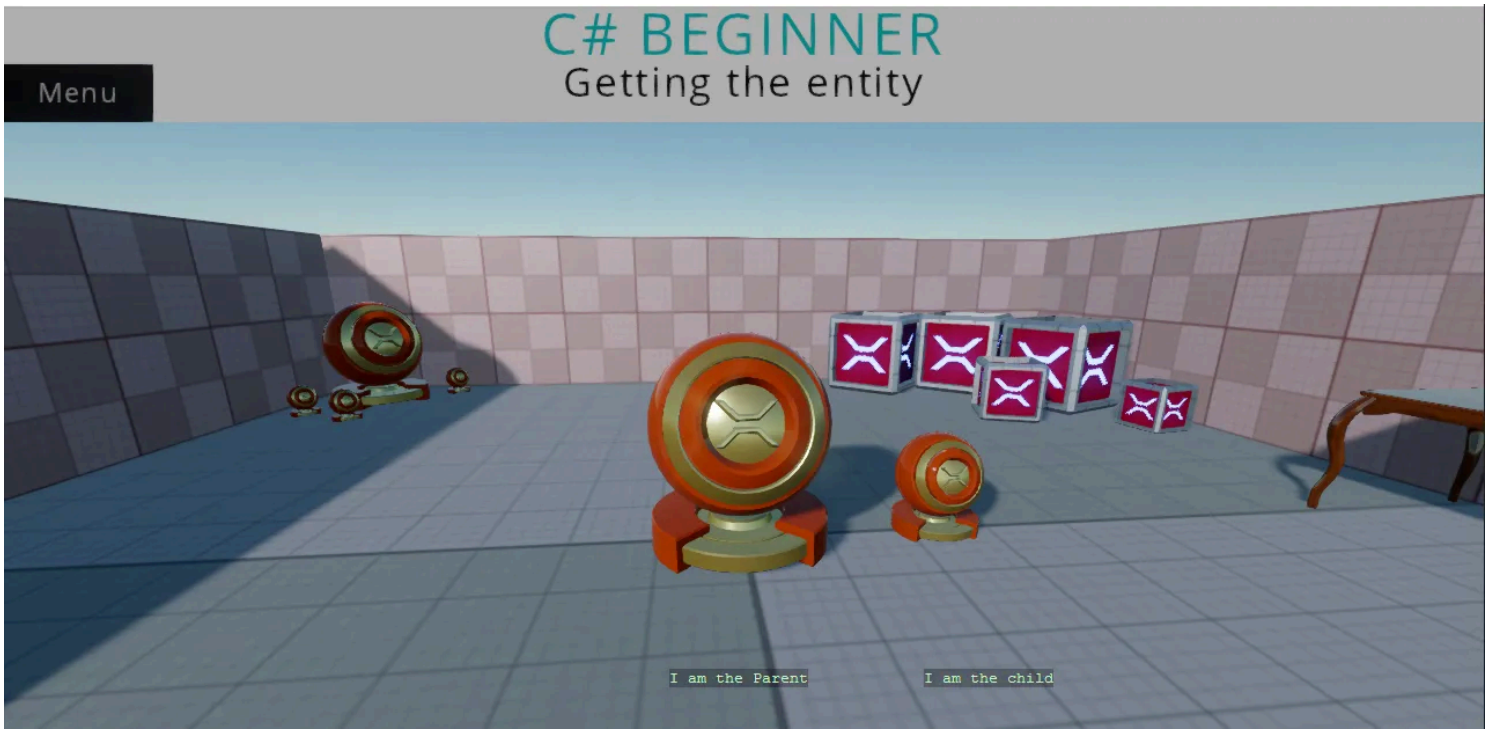
# Getting the entity

You can find this sample in the tutorial project: **Menu** -> **Getting an entity**

## Explanation

This C# Beginner tutorial covers how to get the entity object.

When a script is attached to an entity in the scene, we can access all properties of that Entity by using the `Entity` property. We can for instance get the entity's name or we can check if the entity has a parent in the scene.



Stride tutorial | C# beginner #2 | Getting the entity

# Code

```
using Stride.Core.Mathematics;
using Stride.Engine;

namespace CSharpBeginner.Code
{
    /// <summary>
    /// This script demonstrates how to access the entity where the script is attached to.
    /// We also learn how to access a parent of our entity and how to check if that
entity exists.
    /// <para>
    /// https://doc.stride3d.net/latest/en/tutorials/csharpbeginner/entity.html
    /// </para>
    /// </summary>
    public class GettingTheEntityDemo : SyncScript
    {
        private string name = string.Empty;
        private string parentName = string.Empty;

        // Executes only once, at the start of the game
        public override void Start()
        {
            // We store the name of the Entity that we are attached to
            name = Entity.Name;
```

```
            // We retrieve the parent entity by using the GetParent() command.
            var parentEntity = Entity.GetParent();

            // It is possible that our entity does not have a parent. We therefore check if
the parent is not null.
            if (parentEntity != null)
            {
                // We store the name of our Parent entity
                parentName = parentEntity.Name;
            }

            // The above code can be shortened to 1 line by using the '?' operator
            parentName = Entity.GetParent()?.Name ?? string.Empty;
        }

        // Updates every frame
        public override void Update()
        {
            // Using the 'DebugText.Print' command, we can quickly print information to
the screen
            // NOTE: DebugText only works when debugging the game. During release it is
automatically disabled
            DebugText.Print(parentName, new Int2(580, 580));
            DebugText.Print(name, new Int2(800, 580));
        }
    }
}
```

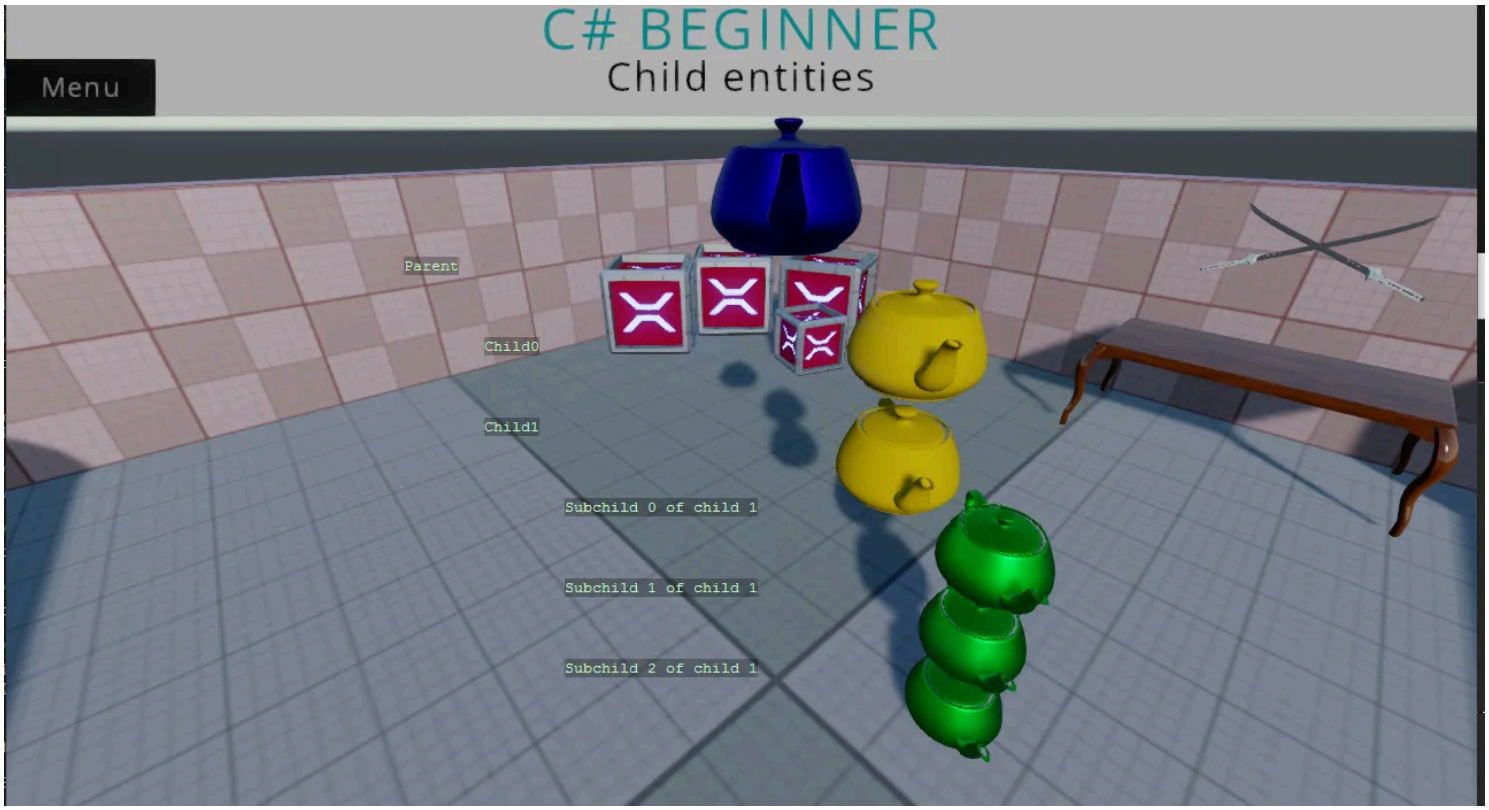
# Children of entities

You can find this sample in the tutorial project: **Menu → Child entities**

## Explanation

This C# basics tutorial covers how to get an entities children.

Since those children are also entities, we can retrieve their children too.



Stride tutorial | C# beginner #3 | Child entities

# Code

```
using Stride.Core.Mathematics;
using Stride.Engine;

namespace CSharpBeginner.Code
{
    /// <summary>
    /// This script is used to demonstrate how we can get child entities of an entity
    /// <para>
    /// https://doc.stride3d.net/latest/en/tutorials/csharpbeginner/child-entities.html
    /// </para>
    /// </summary>
    public class ChildEntitiesDemo : SyncScript
    {
        private Entity child0;
        private Entity child1;

        public override void Start()
        {
            // We can get a child by using GetChild(). This takes an index number starting
at 0
            child0 = Entity.GetChild(0);
            child1 = Entity.GetChild(1);
```

```
            // If we would try to get Child 3 (which doesn't exist), we would get
an exception
            // var nonExistingChild = Entity.GetChild(2);
        }

        public override void Update()
        {
            // We store some drawing positions
            int drawX = 350, drawY = 230, increment = 70;

            // We print the name of the our entity
            DebugText.Print(Entity.Name, new Int2(drawX, drawY));

            // We loop over all the children of our entity using GetChildren()
            // NOTE: This does not include any subchildren of those children
            foreach (var child in Entity.GetChildren())
            {
                // We print the name of the child
                drawY += increment;
                DebugText.Print(child.Name, new Int2(drawX + increment, drawY));

                // It is possible that this child, also has children. We now loop over these
'subchildren' and print their name too
                foreach (var subChild in child.GetChildren())
                {
                    drawY += increment;
                    DebugText.Print(subChild.Name, new Int2(drawX + (increment *
2), drawY));
                }
            }
        }
    }
}
```

# Transform Position

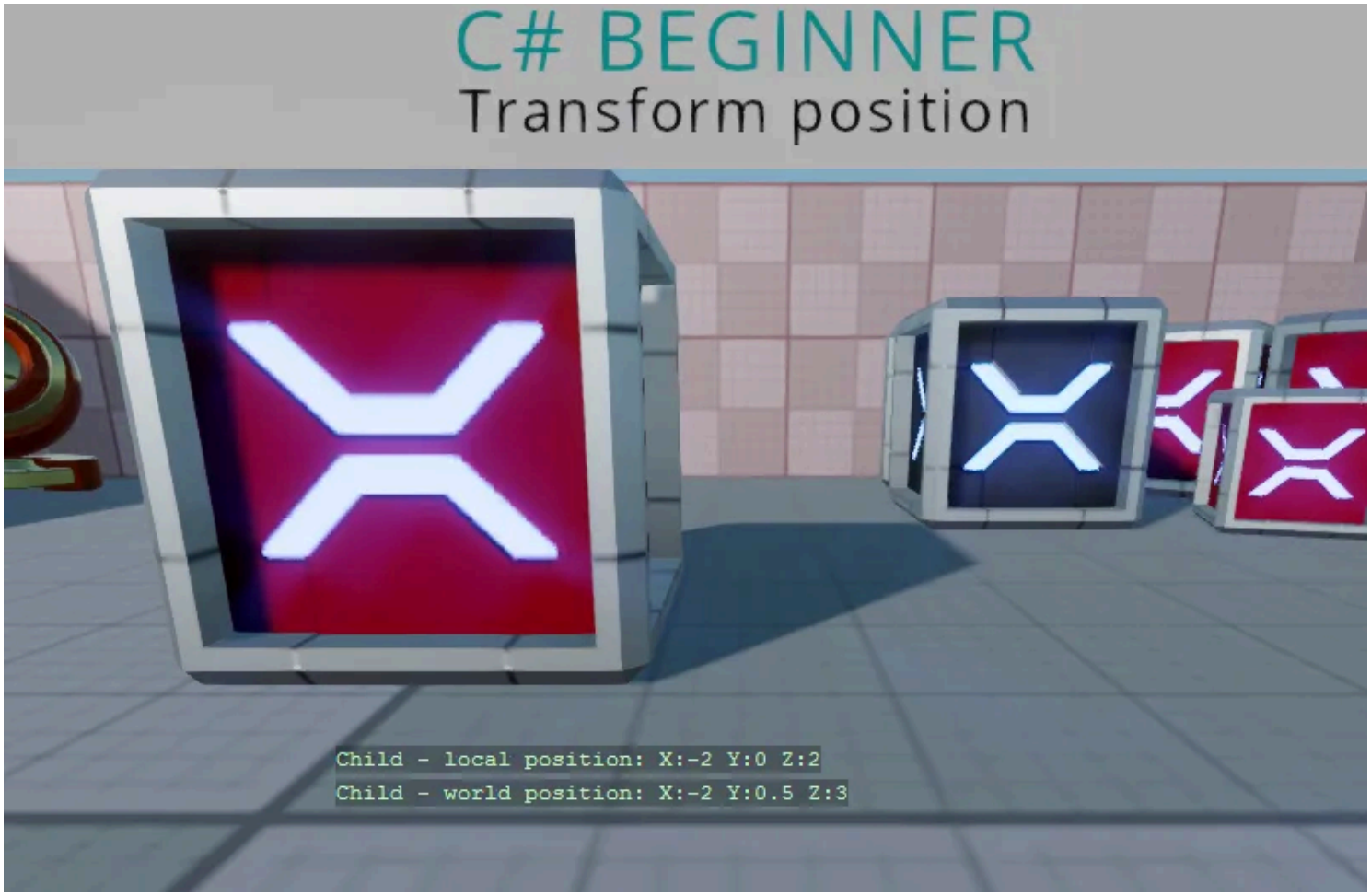
You can find this sample in the tutorial project: **Menu → Transform Position**

## Explanation

This C# Beginner tutorial covers the Transform component of an entity.

The Transform component is such a commonly used component, that you can quick access it via `Entity.Transform`.

The transform contains all kinds of properties and methods for `Position`, `Rotation` and `Scale`. In this example we learn the difference between local and world position.



Stride tutorial | C# beginner #4 | Transform position

# Code

```
using Stride.Core.Mathematics;
using Stride.Engine;

namespace CSharpBeginner.Code
{
    /// <summary>
    /// This script demonstrates how to access the entity's local and world position and
displays them on screen.
    /// <para>
    /// https://doc.stride3d.net/latest/en/tutorials/csharpbeginner/transform-position.html
    /// </para>
    /// </summary>
    public class TransformPositionDemo : SyncScript
    {
        public override void Start() { }

        public override void Update()
        {
            // We store the local and world position of our entity's tranform in a
Vector3 variable
            Vector3 localPosition = Entity.Transform.Position;
            Vector3 worldPosition = Entity.Transform.WorldMatrix.TranslationVector;
```

```
            // We display the entity's name and its local and world position on screen
            DebugText.Print(Entity.Name + " - local position: " + localPosition, new
Int2(400, 450));
            DebugText.Print(Entity.Name + " - world position: " + worldPosition, new
Int2(400, 470));
        }
    }
}
```

# Editor properties

You can find this sample in the tutorial project: **Menu → Editor properties**

## Explanation

This C# Beginner tutorial covers how to expose editor properties for Stride Game Studio.

By creating a public variable at the top of our script, we can create editor properties. Some of the most common properties are demonstrated. We can also create public variables that are not shown in the editor.


C# BEGINNER
Properties
Integer: 10
Float: 5.6
Boolean: True
String: Hello world
Vector2: X:1 Y:2
Vector3: X:1 Y:2 Z:3
Vector4: X:1 Y:2 Z:3 W:4
Color: #FFFF0000
Entity: Ladder
String list count: 3
Entity list count: 2

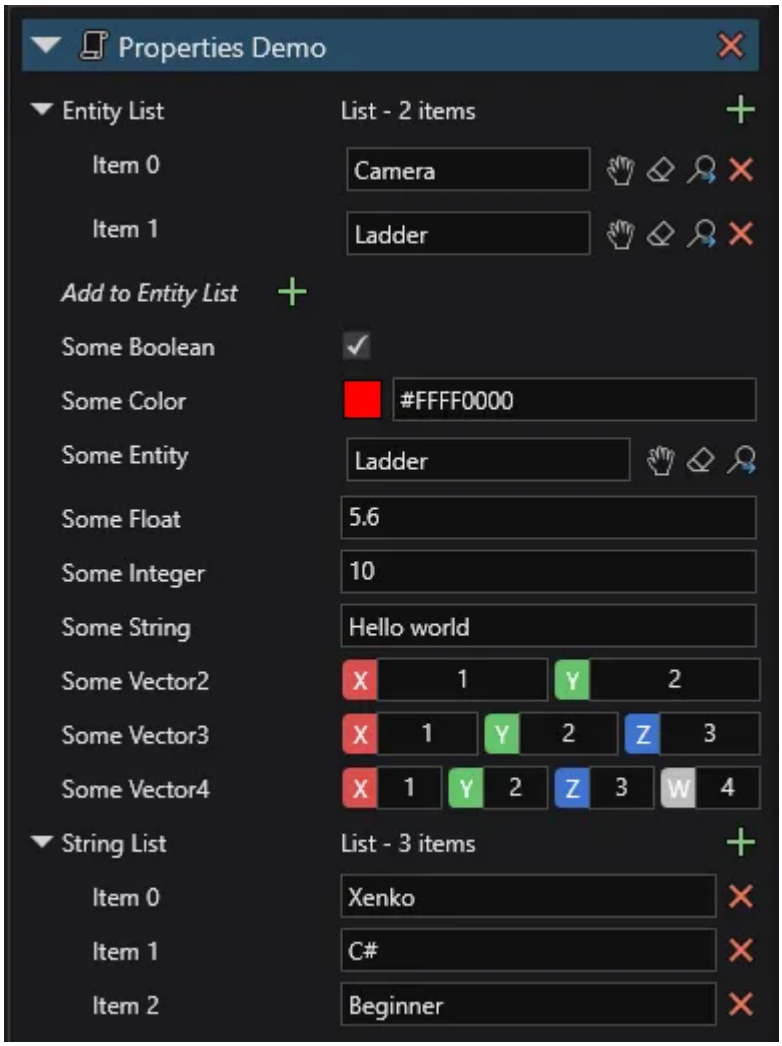

# Code

```
using System.Collections.Generic;
using Stride.Core;
using Stride.Core.Annotations;
using Stride.Core.Mathematics;
using Stride.Engine;

namespace CSharpBeginner.Code
{
    /// <summary>
    /// This script demonstrates the most common properties you can expose to the editor.
    /// When we add the public keyword to the variables, they show up as properties in
the editor.
    /// <para>
    /// https://doc.stride3d.net/latest/en/tutorials/csharpbeginner/editor-properties.html
    /// </para>
    /// </summary>
    public class PropertiesDemo : SyncScript
    {
        // By default public properties are ordered alphabetically
        public string SomeString = "Hello Stride";
        public int SomeInteger = 10;

        // Use DataMember followed by a number to order properties by number
```

```
        [DataMember(5)]
        public bool SomeBoolean = true;

        // Instead of using the name of the variable, you can also define your own text
        [DataMember(4, "My custom text")]
        public float SomeFloat = 5.6f;

        // Vectors
        public Vector2 SomeVector2; // Is the same as = new Vector2(0, 0);
        public Vector3 SomeVector3 = new Vector3(1, 2, 3);
        public Vector4 SomeVector4 = new Vector4(5); // All 4 float value get the value of 5
        public Color SomeColor = Color.Red;

        // Turns a float in to a range slider
        // Dragging the slider uses the smallstep value
        // Clicking the slider uses the bigstep value
        [DataMemberRange(1, 100, 0.1, 1, 3)]
        public float RangedFloat = 10.0f;

        // Entities and components
        public Entity ASingleEntity;
        public CameraComponent ASingleCameraComponent;

        // If we want a list of objects like strings, entities or specific components,
        // we have to initialize the new List<> right away
        public List<string> StringList = new List<string>();
        public List<Entity> EntityList = new List<Entity>();
        public List<CameraComponent> CameraList = new List<CameraComponent>();

        // Dictionaries also need to be initialized. The first value needs to be a primitive
type like string
        public Dictionary<string, int> aSimpleDictionary = new Dictionary<string, int>();

        // If we dont want a public property to be visible in the editor we can
use '[DataMemberIgnore]'
        [DataMemberIgnore]
        public string HideMe;

        // Enums can be used for dropdowns
        public CharacterType Character;
        public enum CharacterType
        {
            Warrior,
            Archer,
            Mage
        }
```

```
        /// <userdoc>This is a super long description. Use it to help the people that use
your components, understand what a propery does.</userdoc>
        /// The text above is displayed in game studio when a property is selected and also
shows up when you hover over the property
        public string Explanation;

        public override void Update()
        {
            var x = 400;
            DebugText.Print("Integer: " + SomeInteger,  new Int2(x, 200));
            DebugText.Print("Float: " + SomeFloat,      new Int2(x, 220));
            DebugText.Print("Boolean: " + SomeBoolean,  new Int2(x, 240));
            DebugText.Print("String: " + SomeString,    new Int2(x, 260));
            DebugText.Print("Vector2: " + SomeVector2,  new Int2(x, 280));
            DebugText.Print("Vector3: " + SomeVector3,  new Int2(x, 300));
            DebugText.Print("Vector4: " + SomeVector4,  new Int2(x, 320));
            DebugText.Print("Color: " + SomeColor,      new Int2(x, 340));
            DebugText.Print("String list count: " + StringList.Count, new Int2(x, 360));
            DebugText.Print("Entity list count: " + EntityList.Count, new Int2(x, 380));
            DebugText.Print("Camera list count: " + CameraList.Count, new Int2(x, 400));
        }
    }
}
```

The code above will result in the following properties inside Stride game studio.


Properties Demo
Entity List
List - 2 items
Item 0
Camera
Item 1
Ladder
Add to Entity List
Some Boolean
Some Color
#FFFF0000
Some Entity
Ladder
Some Float
5.6
Some Integer
10
Some String
Hello world
Some Vector2
X 1 Y 2
Some Vector3
X 1 Y 2 Z 3
Some Vector4
X 1 Y 2 Z 3 W 4
String List
List - 3 items
Item 0
Xenko
Item 1
C#
Item 2
Beginner

# Getting a component

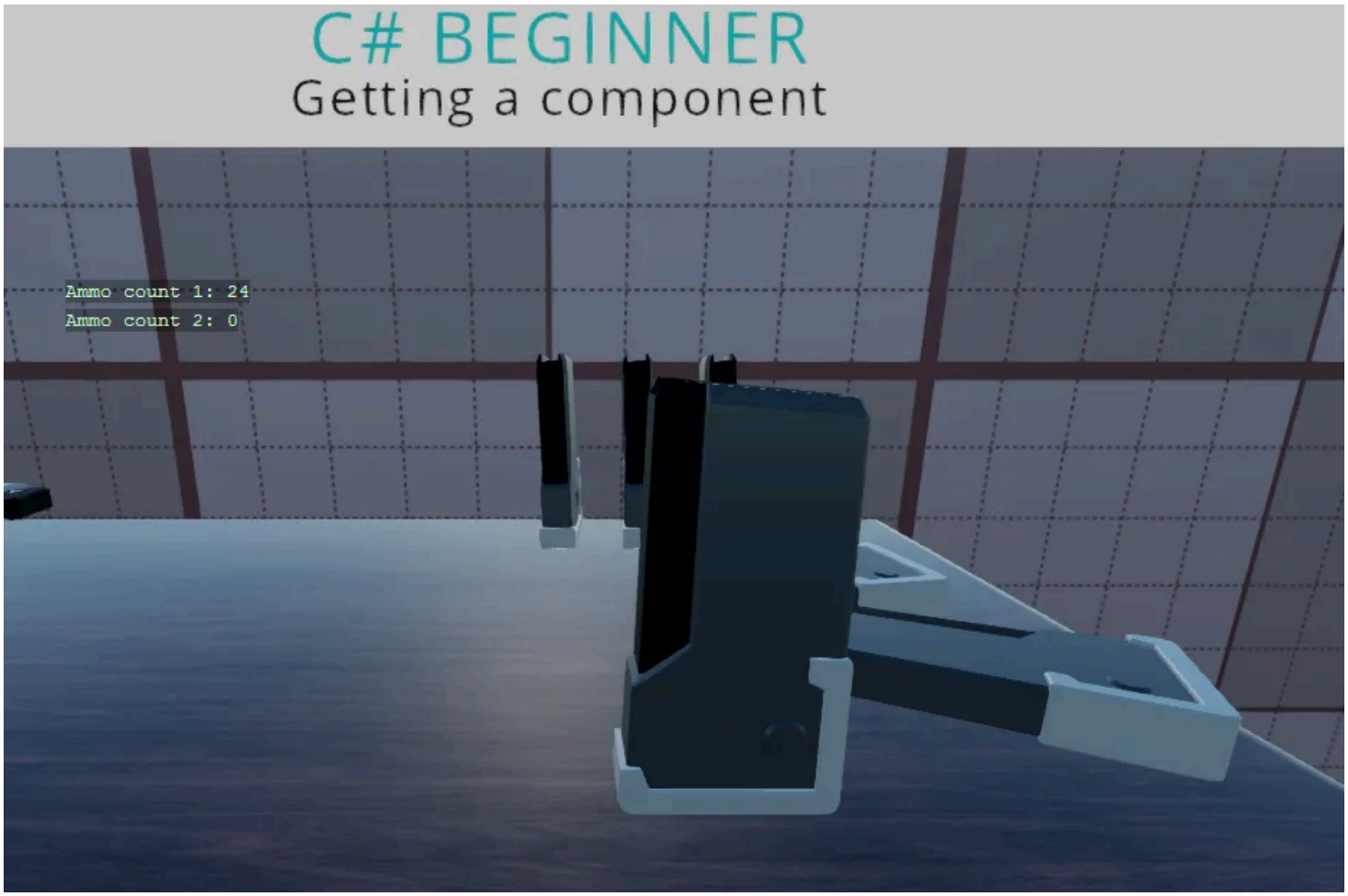
You can find this sample in the tutorial project: **Menu → Getting a component**

## Explanation

This C# beginner tutorial covers how to get and remove components.

Components are one of the most important concepts in Stride. Every entity in the scene has a list of components. The transform for instance is also a component.

When we make custom scripts that inherit from `SyncScript` or `AsyncScript`, they turn into Components that we can attach to entities. We can attach these components to entities by using the editor or we can attach them by code.



Stride tutorial | C# beginner #6 | Getting components

# Code

## AmmoComponent

This is the first component that we attach to an entity. In the second script, we will try to get this AmmoComponent.

```
using Stride.Engine;

namespace CSharpBeginner.Code
{
    /// <summary>
    /// This script is used in combination with the GettingAComponent.cs script
    /// </summary>
    public class AmmoComponent : StartupScript
    {
        private readonly int maxBullets = 30;
        private readonly int currentBullets = 12;

        public override void Start() { }

        public int GetRemainingAmmo()
        {
            return maxBullets - currentBullets;
        }
```

```
    }
}
```

## Getting A Component

This component script, will retrieve the AmmoComponent script above and use its public method.

```
using Stride.Core.Mathematics;
using Stride.Engine;

namespace CSharpBeginner.Code
{
    /// <summary>
    /// This script demonstrates how to get and remove components that are attached to
an entity.
    /// Try not to Get a component every frame as this will have negative
performance impact.
    /// Instead try to cache a component in the start method or when an object
is initialized/triggered
    /// <para>
    /// https://doc.stride3d.net/latest/en/tutorials/csharpbeginner/get-component.html
    /// </para>
    /// </summary>
    public class GettingAComponentDemo : SyncScript
    {
        private int ammoCount1 = 0;
        private int ammoCount2 = 0;

        public override void Start()
        {
            // We retrieve the Ammo component that is also attached to the current entity
            var ammoComponent1 = Entity.Get<AmmoComponent>();

            // We can now access public methods and properties of the retrieve component
            ammoCount1 = ammoComponent1.GetRemainingAmmo();

            // We now remove the AmmoComponent from our entity. If we try to retrieve it
again, null will be returned
            Entity.Remove<AmmoComponent>();
            var ammoComponent2 = Entity.Get<AmmoComponent>();

            // Now that 'ammoComponent' is null, we will never be able to retrieve the
total ammo
            if (ammoComponent2 != null)
            {
```

```
            // This line will never happen
            ammoCount2 = ammoComponent2.GetRemainingAmmo();
        }

        // Add the component again so that it doesn't crash next run
        Entity.Add(ammoComponent1);
    }

    public override void Update()
    {
        // We display the stored ammo count on screen
        DebugText.Print("Ammo count 1: " + ammoCount1.ToString(), new Int2(300, 200));
        DebugText.Print("Ammo count 2: " + ammoCount2.ToString(), new Int2(300, 220));
    }
  }
}
```

# Adding a component

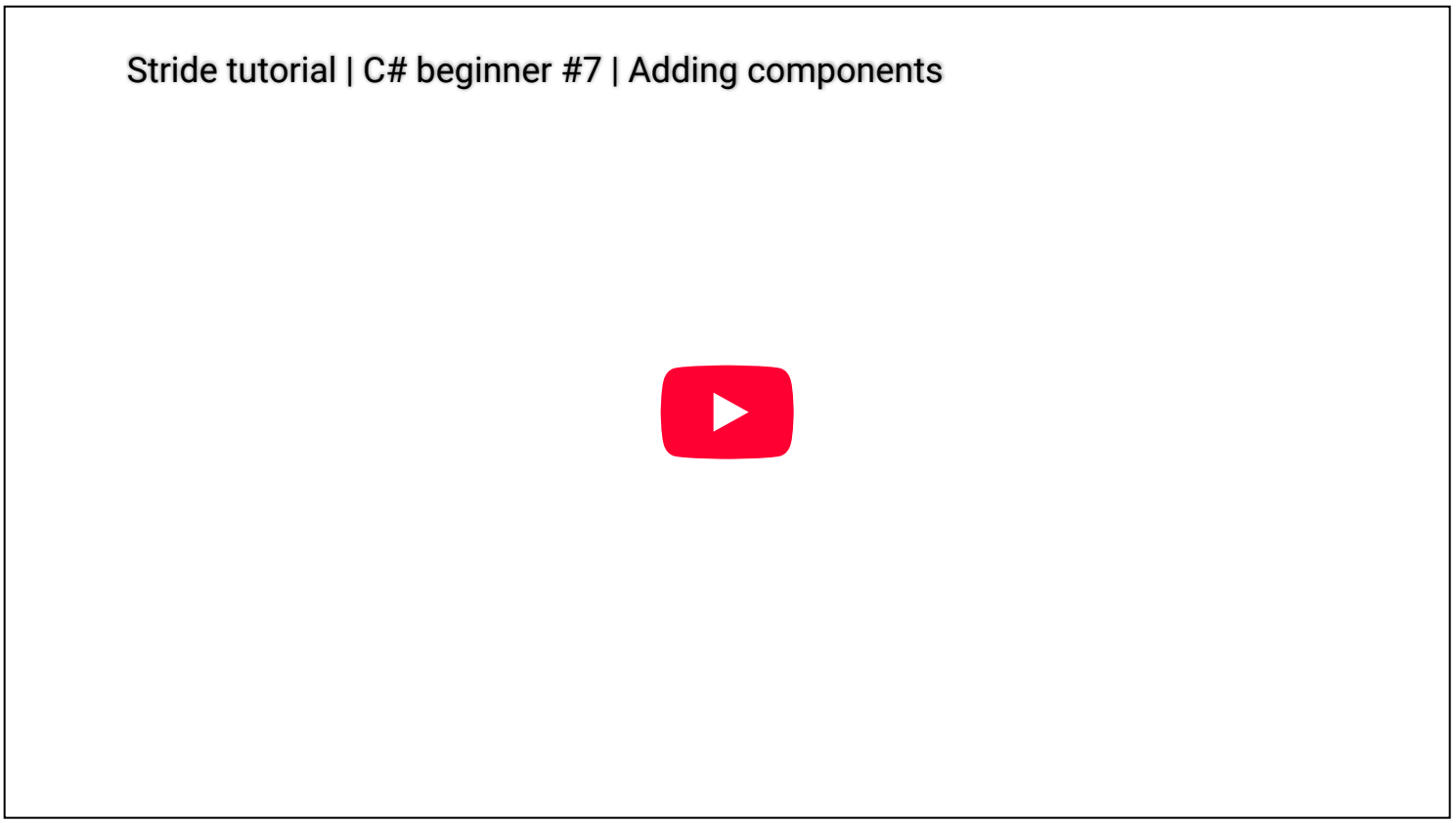
You can find this sample in the tutorial project: **Menu → Adding a component**

## Explanation

This C# Beginner tutorial covers how to add and remove components.

In the previous tutorial we learned how we can retrieve components that are already attached to an entity through the editor. This tutorial shows that we can accomplish the same thing by code.

We can add the same component several times to the same entity. We also learn how to remove all of components of the same type again.




# Code

## AmmoComponent

This is the AmmoComponent. We will not attach it to the entity in the editor. Instead we will add it ourselves in the AddingAComponent script.

```
using Stride.Engine;

namespace CSharpBeginner.Code
{
    /// <summary>
    /// This script is used in combination with the GettingAComponent.cs script
    /// </summary>
    public class AmmoComponent : StartupScript
    {
        private readonly int maxBullets = 30;
        private readonly int currentBullets = 12;

        public override void Start() { }

        public int GetRemainingAmmo()
        {
            return maxBullets - currentBullets;
        }
```

```
    }
}
```

## Adding A Component

This component script, will add the AmmoComponent script to the entity. We then add another component (of the same type) before we remove all components of that type.

Finally we learn how to automatically create a component, attach it to the entity and get a reference all in 1 line of code. This only works if the entity doesn't have any components of the given attached yet.

```
using Stride.Core.Mathematics;
using Stride.Engine;

namespace CSharpBeginner.Code
{
    /// <summary>
    /// This script demonstrates how to add a component to an entiy.
    /// We also learn a way to automically create and attach a component to our entity.
    /// <para>
    /// https://doc.stride3d.net/latest/en/tutorials/csharpbeginner/add-component.html
    /// </para>
    /// </summary>
    public class AddingAComponentDemo : SyncScript
    {
        private AmmoComponent ammoComponent1;
        private AmmoComponent ammoComponent2;
        private AmmoComponent ammoComponent3;

        public override void Start()
        {
            // We can add a new component to an entity using the 'Add' method.
            ammoComponent1 = new AmmoComponent();
            Entity.Add(ammoComponent1);

            // We can even add the component a second time
            ammoComponent2 = new AmmoComponent();
            Entity.Add(ammoComponent2);

            // Lets remove all components of type AmmoComponent
            Entity.RemoveAll<AmmoComponent>();


            // When there is no AmmoComponent of attached, but we like there to be one, we
can create it automatically
```

```
            // NOTE: when a component is created this way,
            // the 'Start' method of the AmmoComponent will be called after this script's
Update method has executed
            ammoComponent3 = Entity.GetOrCreate<AmmoComponent>();
        }

        public override void Update()
        {
            DebugText.Print("Remaining ammo: " +
ammoComponent3.GetRemainingAmmo().ToString(), new Int2(440, 200));
        }
    }
}
```

# Delta Time

You can find this sample in the tutorial project: **Menu → Delta Time**

## Explanation

This C# Beginner tutorial covers the retrieval and usage of delta time.

A game tries to update itself as often as possible. The amount of times it updates in a single second is called 'Frames Per Second' or shortened to 'FPS'.

If we wanted to update a timer value, we would need a value that takes into account what the current amount of frames per second is. That is what delta time is used for. So whether your game runs 30 FPS or 120 FPS: you always want to have the same time scale.

Stride tutorial | C# beginner #8 | Delta time

# Code

```
using Stride.Core.Mathematics;
using Stride.Engine;

namespace CSharpBeginner.Code
{
    /// <summary>
    /// DeltaTime is used to calculate frame independent values.
    /// DeltaTime can also be used for creating Timers.
    /// <para>
    /// https://doc.stride3d.net/latest/en/tutorials/csharpbeginner/delta-time.html
    /// </para>
    /// </summary>
    public class DeltaTimeDemo : SyncScript
    {
        private float rotationSpeed = 0.6f;

        // In this variable we keep track of the total time the game runs
        private float totalTime = 0;

        // We use these variable for creating a simple countdown timer
        private float countdownStartTime = 5.0f;
        private float countdownTime = 0;
```

```
        public override void Start()
        {
            // We start the countdown timer at the initial countdown time of 5 seconds
            countdownTime = countdownStartTime;
        }

        public override void Update()
        {
            /// We can access Delta time through the static 'Game' object.
            var deltaTime = (float)Game.UpdateTime.Elapsed.TotalSeconds;

            // We update the total time
            totalTime += deltaTime;

            // Since we have a countdown timer, we subtract the delta time from the count
down time
            countdownTime -= deltaTime;

            // If the repeatTimer, reaches 0, we reset the countdownTime back to the count
down start time
            if (countdownTime < 0)
            {
                countdownTime = countdownStartTime;
                rotationSpeed *= -1;
            }

            Entity.Transform.Rotation *= Quaternion.RotationY(deltaTime * rotationSpeed);

            // We display the total time and the countdown time on screen
            DebugText.Print("Total time: " + totalTime, new Int2(480, 540));
            DebugText.Print("Countdown time: " + countdownTime, new Int2(480, 560));
        }
    }
}
```

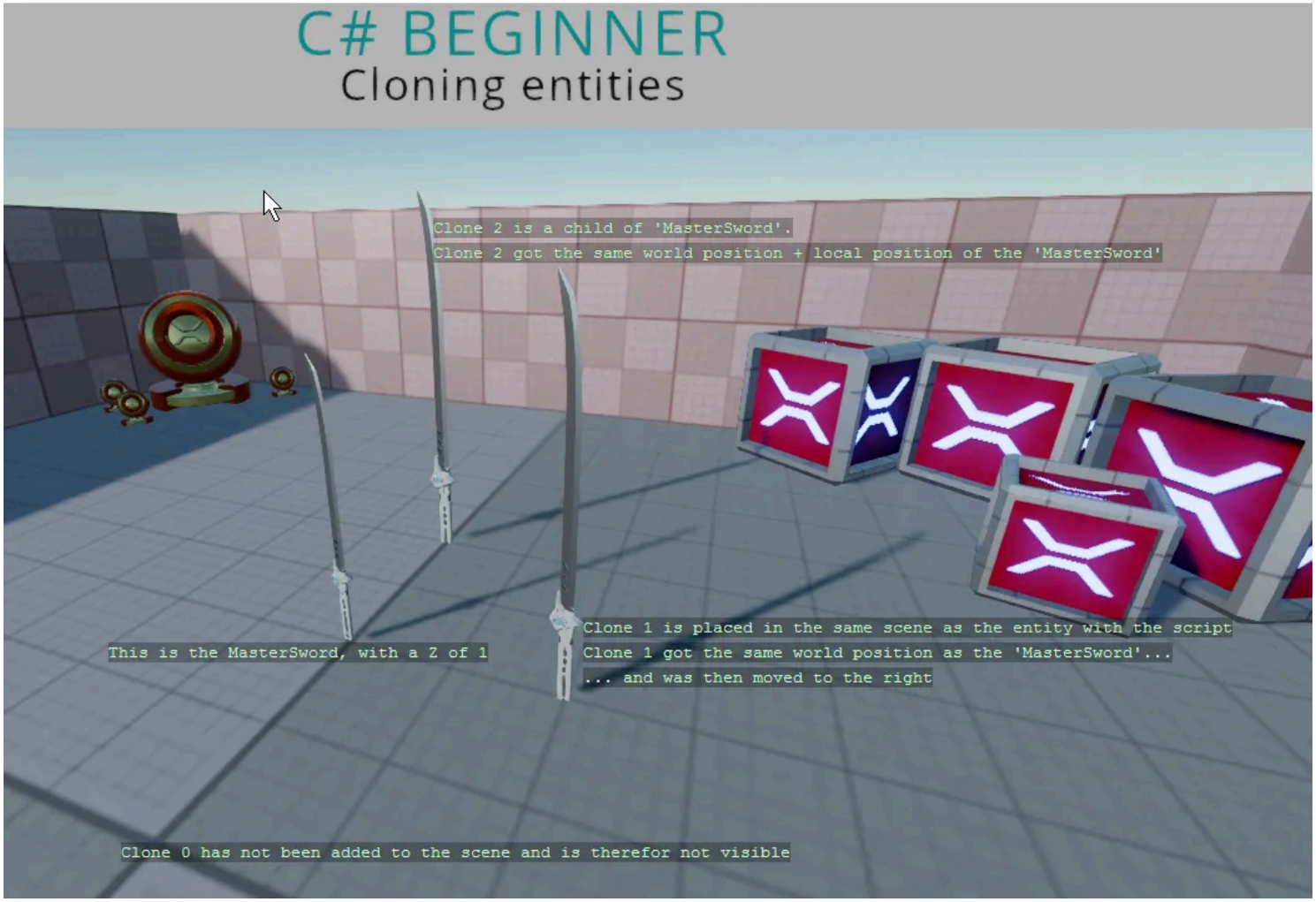
# Cloning an entity

You can find this sample in the tutorial project: **Menu → Cloning entities**

## Explanation

This C# Beginner tutorial covers how to clone an existing entity and how to add that clone to the scene.

A cloned entity is an exact copy of an entity, which means that the Transform and all other components with their set values are copied too.



Stride tutorial | C# beginner #9 | Cloning entities

# Code

```
using Stride.Core.Mathematics;
using Stride.Engine;

namespace CSharpBeginner.Code
{
    /// <summary>
    /// This script demonstrates how to clone an existing entity.
    /// Cloned entities can be added to the scene hierarchy.
    /// <para>
    /// https://doc.stride3d.net/latest/en/tutorials/csharpbeginner/cloning-entities.html
    /// </para>
    /// </summary>
    public class CloneEntityDemo : SyncScript
    {
        public Entity MasterSword;

        private Entity clone1;
        private Entity clone2;
        private Entity clone3;

        public override void Start()
        {
            // Clone 0
```

```
            // The Clone method clones an existing entity, including all of its components
            // However, if we don't add it to the scene, we will never get to see it.
            var clone0 = MasterSword.Clone();
            clone0.Transform.Position += new Vector3(0, 1, 0);


            // Clone 1
            // We can add Clone1 to the same scene that the current entity is part of
            clone1 = MasterSword.Clone();
            Entity.Scene.Entities.Add(clone1);
            // The cloned entity will be at the same worldposition as the original
Sword entity
            // Move it to the right so that we can see it
            clone1.Transform.Position += new Vector3(-1, 0, 0);
            clone1.Transform.Scale *= new Vector3(0.8f);


            // Clone 2
            // We can also add a cloned entity as a child of an existing entity.
            clone2 = MasterSword.Clone();
            Entity.AddChild(clone2);
            clone2.Transform.Position += new Vector3(1, 0, 0);
            clone2.Transform.Scale = new Vector3(0.6f);


            // Clone 3
            // We can also add a cloned entity as a child of an existing entity by setting
the parent
            // That means it will use the parent's world position + parent's local position
            clone3 = MasterSword.Clone();
            clone3.Transform.Parent = Entity.Transform; // Or
Entity.SetParent(Entity.Transform)
            clone3.Transform.Position += new Vector3(0, 0, -0.5f);
            clone3.Transform.Scale = new Vector3(0.4f);
        }

        public override void Update()
        {
            DebugText.Print("This is the MasterSword, with a Z of 1", new Int2(500, 320));
            DebugText.Print("Clone 0 has not been added to the scene and is therefore not
visible", new Int2(600, 250));

            DebugText.Print("Clone 1 is placed in the same scene as the entity with the
script", new Int2(700, 600));
            DebugText.Print("Clone 1 got the same world position as the 'MasterSword'...",
new Int2(700, 620));
```

```
            DebugText.Print("... and was then moved to the right", new Int2(700, 640));

            DebugText.Print("Clone 2 and 3 are a child of 'MasterSword'.", new
Int2(330, 600));
        }
    }
}
```

# Removing entities

You can find this sample in the tutorial project: **Menu → Removing entities**

## Explanation

This C# Beginner tutorial covers how to remove existing entities from the scene and how to remove an entity that is a child of another entity.

When we add entities to the Scene root we can remove that entity again by accessing the scene.Entities property. Entities that are added as children of other entities can be removed by accessing the children of an entity.

Stride tutorial | C# beginner #10 | Removing entities

# Code

```
using Stride.Core.Mathematics;
using Stride.Engine;

namespace CSharpBeginner.Code
{
    /// <summary>
    /// This script demonstrates how to remove an existing entity from the scene hierarchy.
    /// <para>
    /// https://doc.stride3d.net/latest/en/tutorials/csharpbeginner/removing-entities.html
    /// </para>
    /// </summary>
    public class RemoveEntitiesDemo : SyncScript
    {
        public Entity EntityToClone;

        private Entity clonedEntity1;
        private Entity clonedEntity2;
        private float timer = 0;

        private float currentTimer = 0;
        private float existTime = 4;
        private float goneTime = 2;
```

```
        private bool entitiesExist = false;

        public override void Start()
        {
            CloneEntityAndAddToScene();
            CloneEntityAndAddAsChild();
            entitiesExist = true;
        }

        /// This method clones an entity, adds it as a child of the current entity
        private void CloneEntityAndAddAsChild()
        {
            clonedEntity1 = EntityToClone.Clone();
            clonedEntity1.Transform.Position = new Vector3(0);
            Entity.AddChild(clonedEntity1);
        }

        /// This method clones an entity, adds it to the scene root
        private void CloneEntityAndAddToScene()
        {
            clonedEntity2 = EntityToClone.Clone();
            clonedEntity2.Transform.Position += new Vector3(0, 0, -0.5f);
            Entity.Scene.Entities.Add(clonedEntity2);
        }

        public override void Update()
        {
            // We use a simple timer
            timer += (float)Game.UpdateTime.Elapsed.TotalSeconds;
            if (timer > currentTimer)
            {
                // If the entities exist, we remove them from the scene
                if (entitiesExist)
                {
                    // We remove the cloned entity that is a child of the current entity
                    Entity.RemoveChild(clonedEntity1); // Alternative:
clonedEntity1.Transform.Parent = null;

                    // We remove the cloned entity from the scene root
                    Entity.Scene.Entities.Remove(clonedEntity2);

                    // We also need to set the clones to null, otherwise the clones
still exist
                    clonedEntity1 = null;
                    clonedEntity2 = null;
```

```
                    entitiesExist = false;
                    currentTimer = goneTime;
                }
                else // If the entities don't exist, we create new clones
                {
                    CloneEntityAndAddToScene();
                    CloneEntityAndAddAsChild();
                    entitiesExist = true;

                    currentTimer = existTime;
                }

                // Reset timer
                timer = 0;
            }

            DebugText.Print("For " + existTime.ToString() + " seconds: ", new
Int2(860, 240));
            DebugText.Print("- Clone 1 is a child of the script entity", new
Int2(860, 260));
            DebugText.Print("- Clone 2 is a child of the scene root", new Int2(860, 280));
            DebugText.Print("For " + goneTime.ToString() + " seconds, the cloned entities
are gone", new Int2(860, 300));

            if (entitiesExist)
            {
                DebugText.Print("Cloned entity 1 is a child of the Script entity", new
Int2(450, 350));
                DebugText.Print("Cloned entity 2 is in the scene root", new Int2(450, 600));
            }
            else
            {
                DebugText.Print("Cloned entity 1 and 2 have been removed", new
Int2(450, 600));
            }
        }
    }
}
```

# Mouse input

You can find this sample in the tutorial project: **Menu → Mouse input**

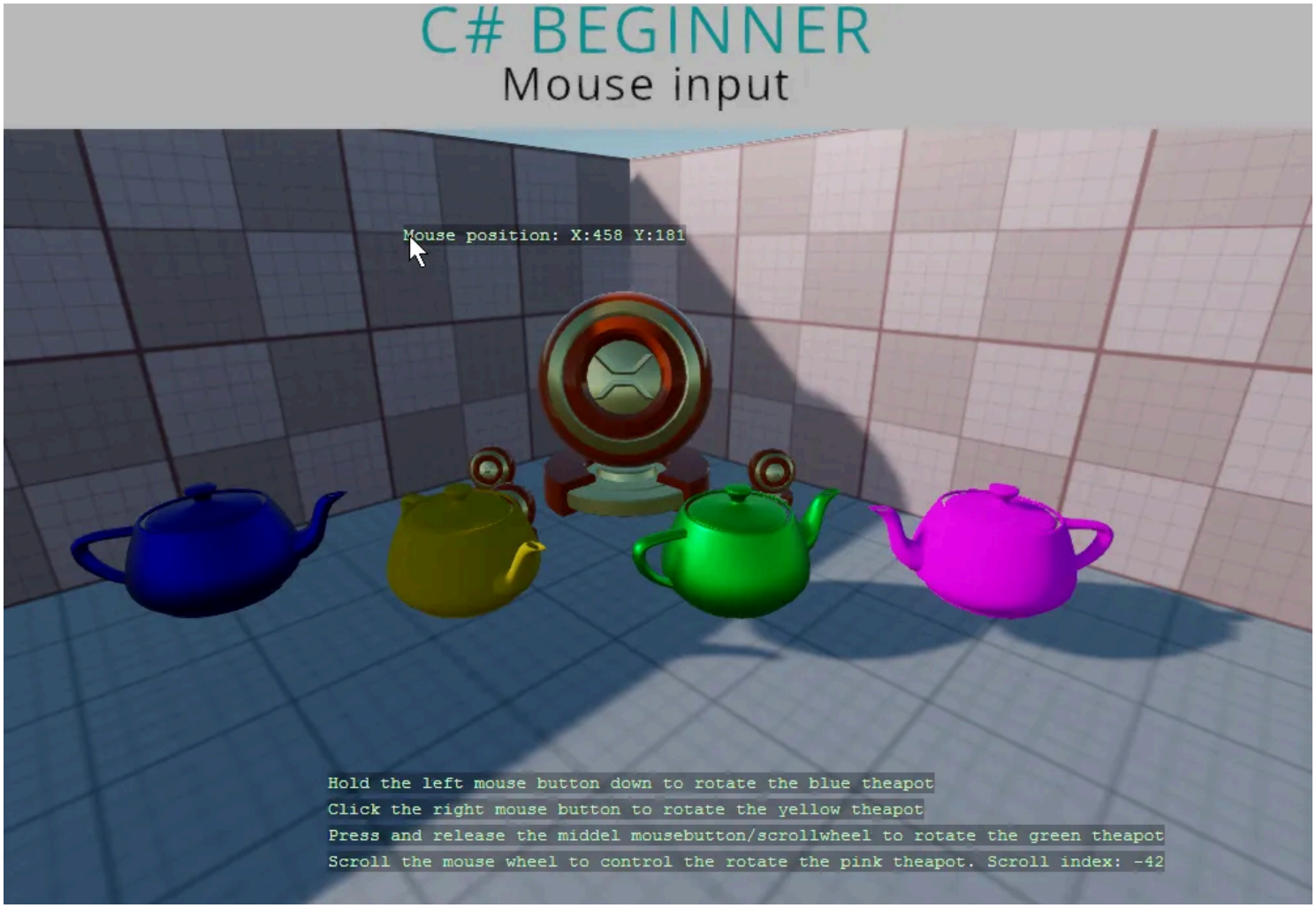
## Explanation

This C# Beginner tutorial covers how to handle mouse input.

We can check for the existence of a mouse and then we can use various methods to check if a mouse buttons are clicked, held down or released.

We can also check for the mouse wheel (middle mouse) being clicked. We can use the mouse wheel delta to determine if the mouse wheel has been scrolled in a frame.

And finally we learn how to use the absolute mouse position to draw text at the position of the mouse on the screen.



Stride tutorial | C# beginner #11 | Mouse input

# Code

```
using Stride.Core.Mathematics;
using Stride.Engine;
using Stride.Input;

namespace CSharpBeginner.Code
{
    /// <summary>
    /// This script demonstrates how to check for any mouse input.
    /// <para>
    /// https://doc.stride3d.net/latest/en/tutorials/csharpbeginner/mouse-input.html
    /// </para>
    /// </summary>
    public class MouseInputDemo : SyncScript
    {
        public Entity BlueTeapot;
        public Entity YellowTeapot;
        public Entity GreenTeapot;
        public Entity PinkTeapot;

        private float currentScrollIndex = 0;

        public override void Start() { }
```

```
        public override void Update()
        {
            // First lets check if we have a mouse.
            if (Input.HasMouse)
            {
                // Key down is used for when a key is being held down.
                DebugText.Print("Hold the left mouse button down to rotate the blue teapot",
new Int2(400, 600));
                if (Input.IsMouseButtonDown(MouseButton.Left))
                {
                    var deltaTime = (float)Game.UpdateTime.Elapsed.TotalSeconds;
                    BlueTeapot.Transform.Rotation *= Quaternion.RotationY(0.4f * deltaTime);
                }

                // Use 'IsMouseButtonPressed' for a single mouse click event.
                DebugText.Print("Click the right mouse button to rotate the yellow teapot",
new Int2(400, 620));
                if (Input.IsMouseButtonPressed(MouseButton.Right))
                {
                    YellowTeapot.Transform.Rotation *= Quaternion.RotationY(-0.4f);
                }

                // 'IsMouseButtonReleased' is used for when you want to know when a mouse
button is released after being either held down or pressed.
                DebugText.Print("Press and release the scrollwheel to rotate the green
teapot", new Int2(400, 640));
                if (Input.IsMouseButtonReleased(MouseButton.Middle))
                {
                    GreenTeapot.Transform.Rotation *= Quaternion.RotationY(0.4f);
                }

                // We can use the mousewheel delta do determine if a mousewheel has rotated.
                // Scrolling forward gives a mousewheel delta of 1, and scrolling backwards
gives a mousewheel delta of -1.
                // If in the next frame the mousewheel is not scrolled, the mouse wheel
delta is 0 again.
                currentScrollIndex += Input.MouseWheelDelta;
                DebugText.Print("Scroll the mouse wheel to rotate the pink teapot. Scroll
index: " + currentScrollIndex, new Int2(400, 660));
                PinkTeapot.Transform.Rotation = Quaternion.RotationY(0.02f
* currentScrollIndex);

                // We can draw some text at the position of our mouse by getting the
absolute mouse position
                var mousePos = Input.AbsoluteMousePosition;
                DebugText.Print("Mouse position: " + mousePos, new Int2(mousePos));
```

```
            }
        }
    }
}
```

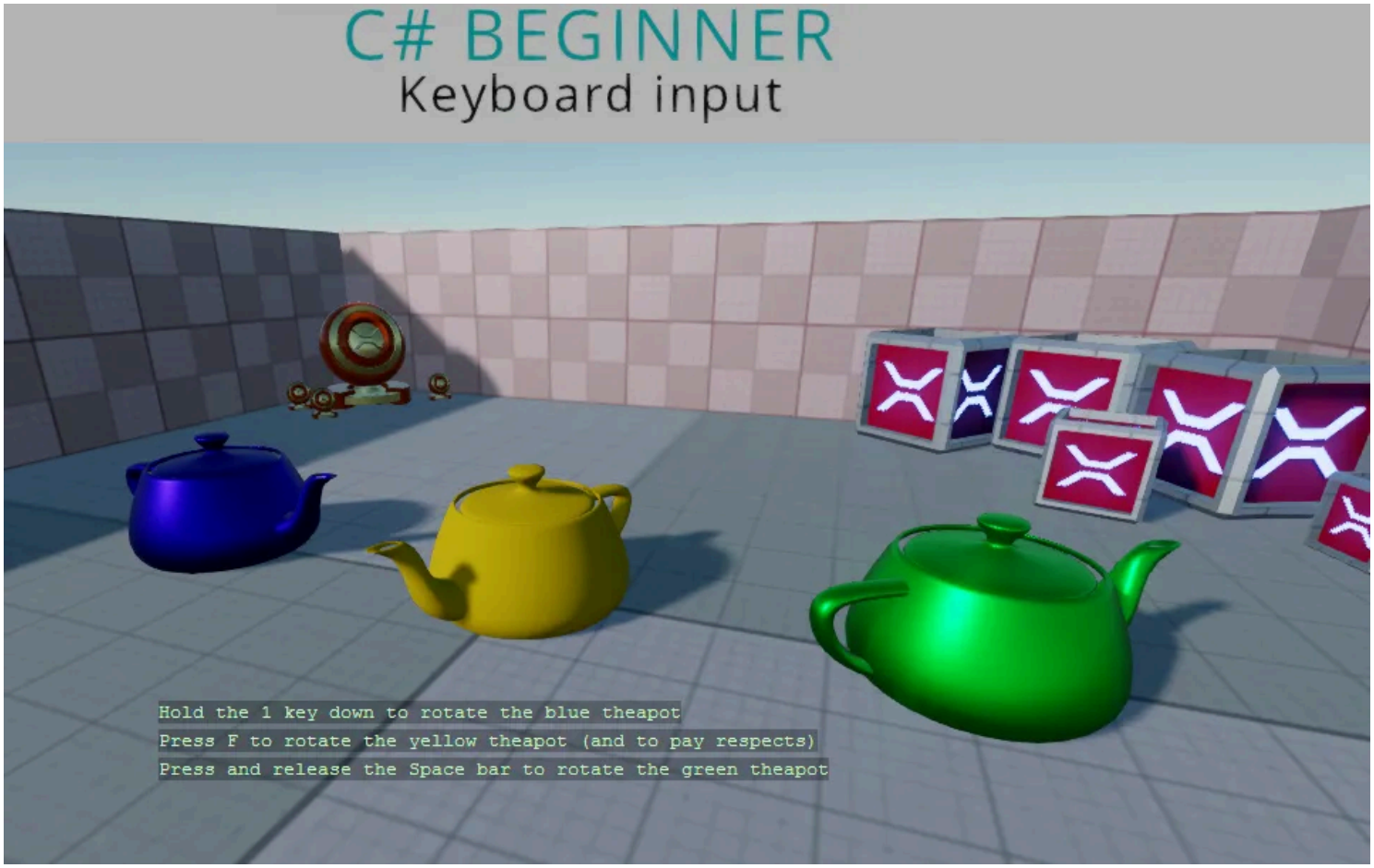
# Keyboard input

You can find this sample in the tutorial project: **Menu → Keyboard input**

## Explanation

This C# Beginner tutorial covers how to handle keyboard input.

We can check for the existence of a keyboard and then we can use various methods to check if a key is pressed, held down or released.



Stride tutorial | C# beginner #12 | Keyboard input

# Code

```
using Stride.Core.Mathematics;
using Stride.Engine;
using Stride.Input;

namespace CSharpBeginner.Code
{
    /// <summary>
    /// This script demonstrates how to check for keyboard input.
    /// <para>
    /// https://doc.stride3d.net/latest/en/tutorials/csharpbeginner/keyboard-input.html
    /// </para>
    /// </summary>
    public class KeyboardInputDemo : SyncScript
    {
        public Entity BlueTeapot;
        public Entity YellowTeapot;
        public Entity GreenTeapot;

        public override void Start() { }

        public override void Update()
        {
            // First lets check if we have a keyboard.
```

```
            if (Input.HasKeyboard)
            {
                // Key down is used for when a key is being held down.
                DebugText.Print("Hold the 1 key down to rotate the blue teapot", new
Int2(340, 500));
                if (Input.IsKeyDown(Keys.D1))
                {
                    var deltaTime = (float)Game.UpdateTime.Elapsed.TotalSeconds;
                    BlueTeapot.Transform.Rotation *= Quaternion.RotationY(0.3f * deltaTime);
                }

                // Use 'IsKeyPressed' for a single key press event.
                DebugText.Print("Press F to rotate the yellow teapot (and to pay respects)",
new Int2(340, 520));
                if (Input.IsKeyPressed(Keys.F))
                {
                    YellowTeapot.Transform.Rotation *= Quaternion.RotationY(-0.4f);
                }

                // 'IsKeyReleased' is used for when you want to know when a key is released
after being either held down or pressed.
                DebugText.Print("Press and release the Space bar to rotate the green
teapot", new Int2(340, 540));
                if (Input.IsKeyReleased(Keys.Space))
                {
                    GreenTeapot.Transform.Rotation *= Quaternion.RotationY(0.6f);
                }
            }
        }
    }
}
```

# Virtual buttons

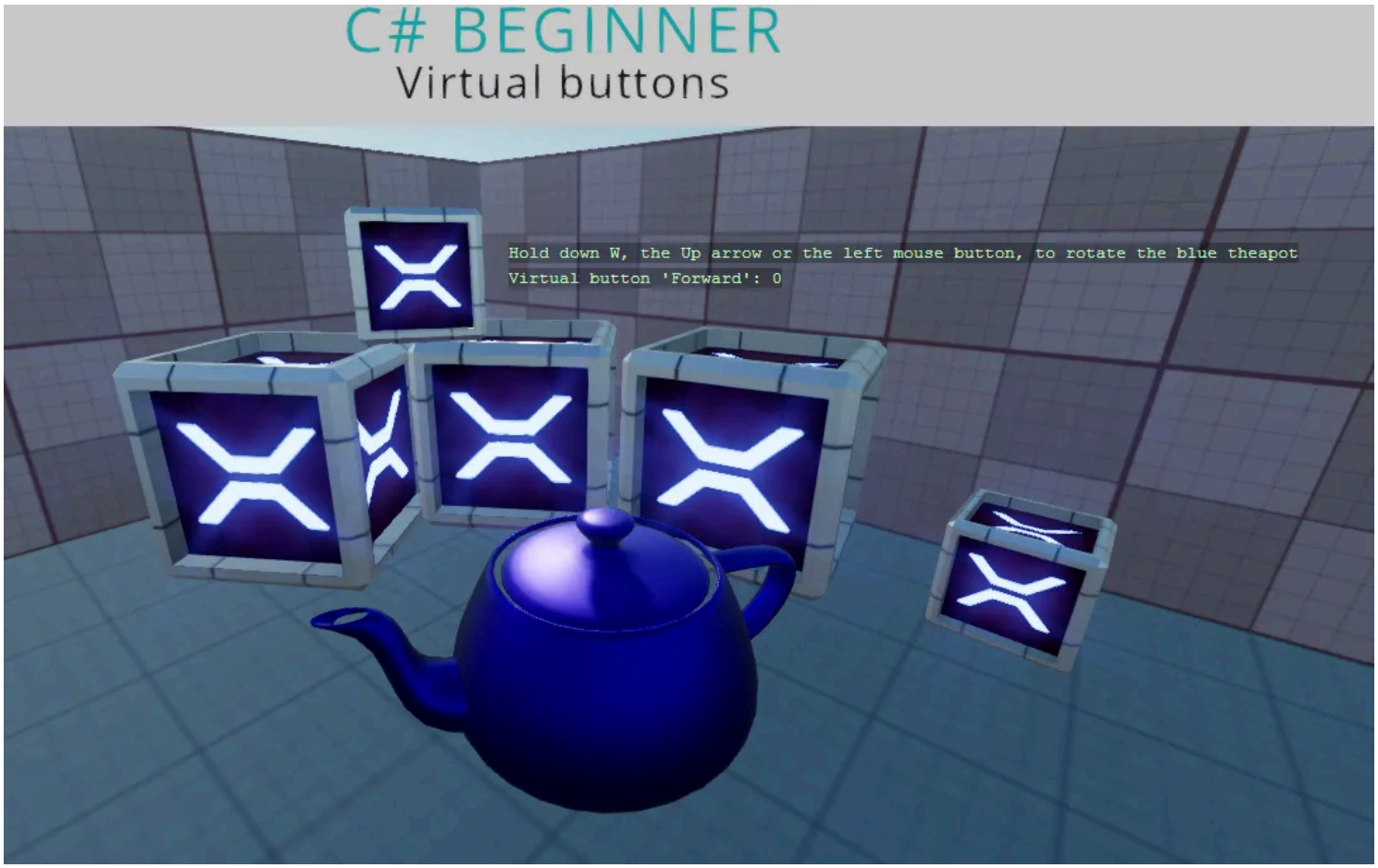
You can find this sample in the tutorial project: **Menu → Virtual buttons**

## Explanation

This C# Beginner tutorial covers how to create virtual buttons.

Lets say that you want a player to jump when a key is pressed. The space bar is a common option, but what if a gamer wants to have a different key bind to this 'Jump' action?

The answer here is the 'Virtual button'. Virtual buttons allow the mapping of one or more keyboard keys, mouse buttons or joystick buttons to a single 'Virtual button'. We can check for the name of that virtual button to see if any of the virtual buttons are triggered.



Stride tutorial | C# beginner #13 | Virtual buttons

# Code

```
using Stride.Core.Mathematics;
using Stride.Engine;
using Stride.Input;

namespace CSharpBeginner.Code
{
    /// <summary>
    /// This script demonstrates how to create virtual buttons and how to use them.
    /// <para>
    /// https://doc.stride3d.net/latest/en/tutorials/csharpbeginner/virtual-buttons.html
    /// </para>
    /// </summary>
    public class VirtualButtonsDemo : SyncScript
    {
        public Entity BlueTeapot;

        public override void Start()
        {
            // Create a new VirtualButtonConfigSet if none exists.
            Input.VirtualButtonConfigSet = Input.VirtualButtonConfigSet ??
new VirtualButtonConfigSet();

            // Bind the "W" key and "Up arrow" to a virtual button called "Forward".
```

```
            var forwardW = new VirtualButtonBinding("Forward", VirtualButton.Keyboard.W);
            var forwardUpArrow = new VirtualButtonBinding("Forward",
VirtualButton.Keyboard.Up);
            var forwardLeftMouse = new VirtualButtonBinding("Forward",
VirtualButton.Mouse.Left);
            var forwardLeftTrigger = new VirtualButtonBinding("Forward",
VirtualButton.GamePad.LeftTrigger);

            // Create a new virtual button configuration and add the virtual button bindings
            var virtualButtonForward = new VirtualButtonConfig
            {
                forwardW,
                forwardUpArrow,
                forwardLeftMouse,
                forwardLeftTrigger
            };

            // Add the virtual button binding to the virtual button configuration
            Input.VirtualButtonConfigSet.Add(virtualButtonForward);
        }

        public override void Update()
        {
            // We retrieve a float value from the virtual button.
            // When the value is higher than 0, we know that we have at least one of the
keys or mouse pressed
            // Keyboard and mouse return a value of 1 if they are being pressed.
            // Gamepads can have a more accurate value between 0 and 1 depending on how far
a trigger is being pressed
            var forward = Input.GetVirtualButton(0, "Forward");

            // Note: Gamepad sticks can be a negative value. For this example we only check
if the value is higher than 0
            if (forward > 0)
            {
                var deltaTime = (float)Game.UpdateTime.Elapsed.TotalSeconds;
                BlueTeapot.Transform.Rotation *= Quaternion.RotationY(0.6f * forward
* deltaTime);
            }

            DebugText.Print("Hold down W, the Up arrow the left mouse button or the Left
trigger on a gamepad", new Int2(600, 200));
            DebugText.Print("Virtual button 'Forward': " + forward, new Int2(600, 220));
        }
    }
}
```

# Linear Interpolation

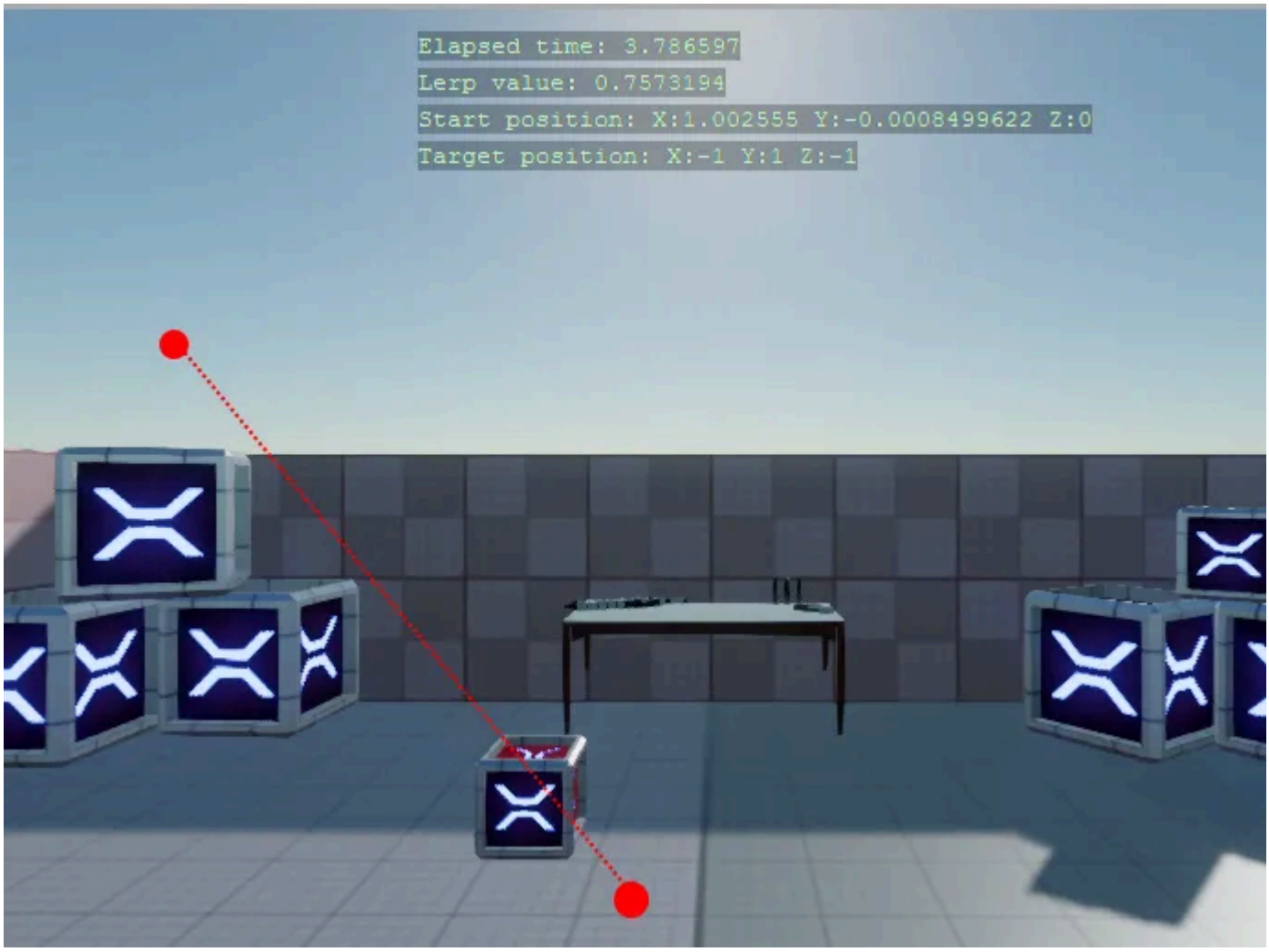
You can find this sample in the tutorial project: **Menu → Linear Iterpolation**

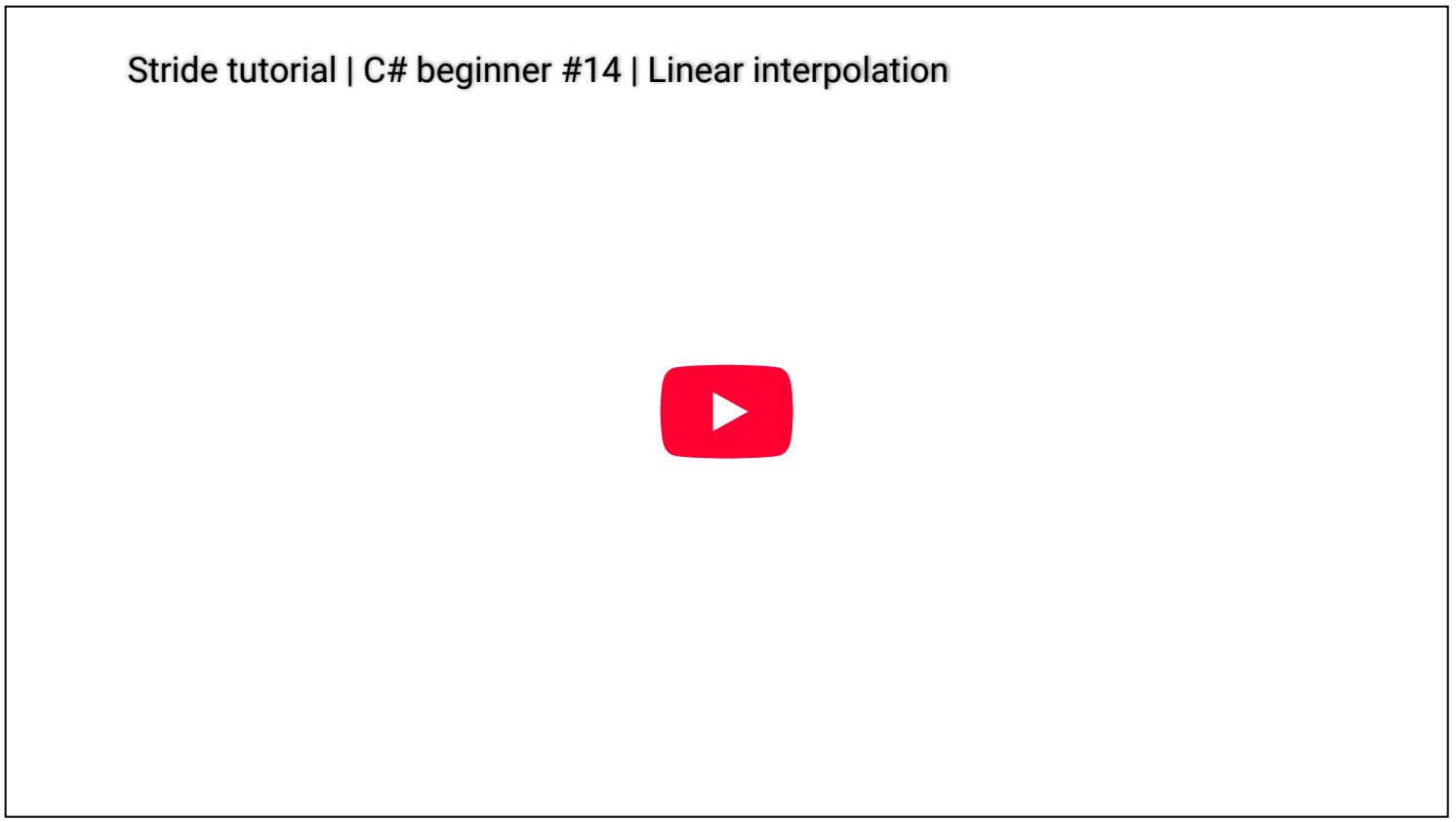
## Explanation

This C# Beginner tutorial covers linear interpolation which is often shortened to 'Lerp'.

Sometimes you want to gradually change a value from a start value to a target value. This process is called linear interpolation.

Stride exposes several Lerp functions for various types. Among them are `Vector2`, `Vector3` and `Vector4`.






# Code

The example consists of a simple timer that resets after a couple seconds. When the timer starts, a start position and a randomly generated target position are stored. A box will move between these two positions.

Every frame a 'Lerp value' is calculated. The lerp value is used to determined what the current position of a moving box should be. Once the timer is done, the current position will become the start position and a new target position is again randomly generated.

```
using System;
using Stride.Core.Mathematics;
using Stride.Engine;

namespace CSharpBeginner.Code
{
    /// <summary>
    /// Liniear interpolation or in short 'Lerp' can be used to graduatly change a value
from a start value to a target value
    /// This is used during animation of models, ui elements, camera movements and many
other scenarios
    /// This example uses Lerp to graduatly move from a start vector3 coordinate to target
Vector3 coordinate
    /// The same thing can be done with Vector2 and Vector4
    /// <para>
```

```
    /// https://doc.stride3d.net/latest/en/tutorials/csharpbeginner/linear-
interpolation.html
    /// </para>
    /// </summary>
    public class LerpDemo : SyncScript
    {
        public float AnimationTimer = 5.0f;

        private float elapsedTime = 0;
        private Random random = new Random();
        private Vector3 startPosition;
        private Vector3 targetPosition;

        public override void Start()
        {
            SetNewLerpTargetAndResetTimer();
        }

        public override void Update()
        {
            // Keep track of elapsed time
            var deltaTime = (float)Game.UpdateTime.Elapsed.TotalSeconds;
            elapsedTime += deltaTime;

            // In order to make use of the lerp method, we need to calculate the
'interpolation value': a value going from 0 to 1.
            var lerpValue = elapsedTime / AnimationTimer;

            // The Vector3 class exposes a 'Lerp' method that requires a start and target
position. The third argument is the lerp value.
            Entity.Transform.Position = Vector3.Lerp(startPosition,
targetPosition, lerpValue);

            // If the elapsedTime passes the animation timer we reset the timer and set a
new target
            if (elapsedTime > AnimationTimer)
            {
                SetNewLerpTargetAndResetTimer();
            }

            DebugText.Print("Elapsed time: " + elapsedTime, new Int2(480, 120));
            DebugText.Print("Lerp value: " + lerpValue, new Int2(480, 140));
            DebugText.Print("Start position: " + startPosition, new Int2(480, 160));
            DebugText.Print("Target position: " + targetPosition, new Int2(480, 180));
        }
```

```
        /// <summary>
        /// Resets timer, stores the current position and randomly sets a new
target position
        /// </summary>
        private void SetNewLerpTargetAndResetTimer()
        {
            elapsedTime = 0;
            startPosition = Entity.Transform.Position;
            targetPosition = new Vector3(random.Next(-2, 2), random.Next(0, 3),
random.Next(-1, 1));
        }
    }
}
```

# Loading content

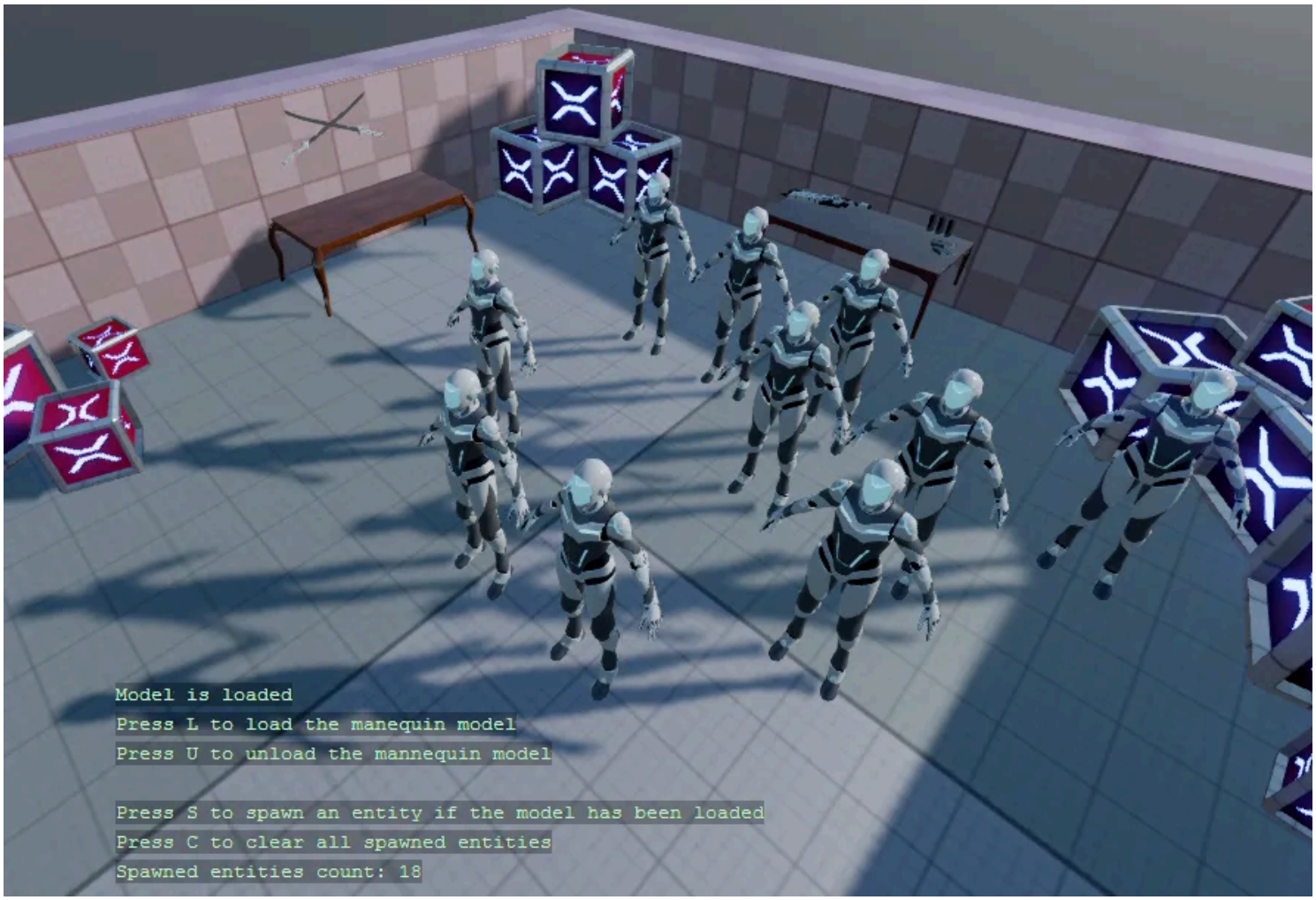
You can find this sample in the tutorial project: **Menu → Loading content from code**

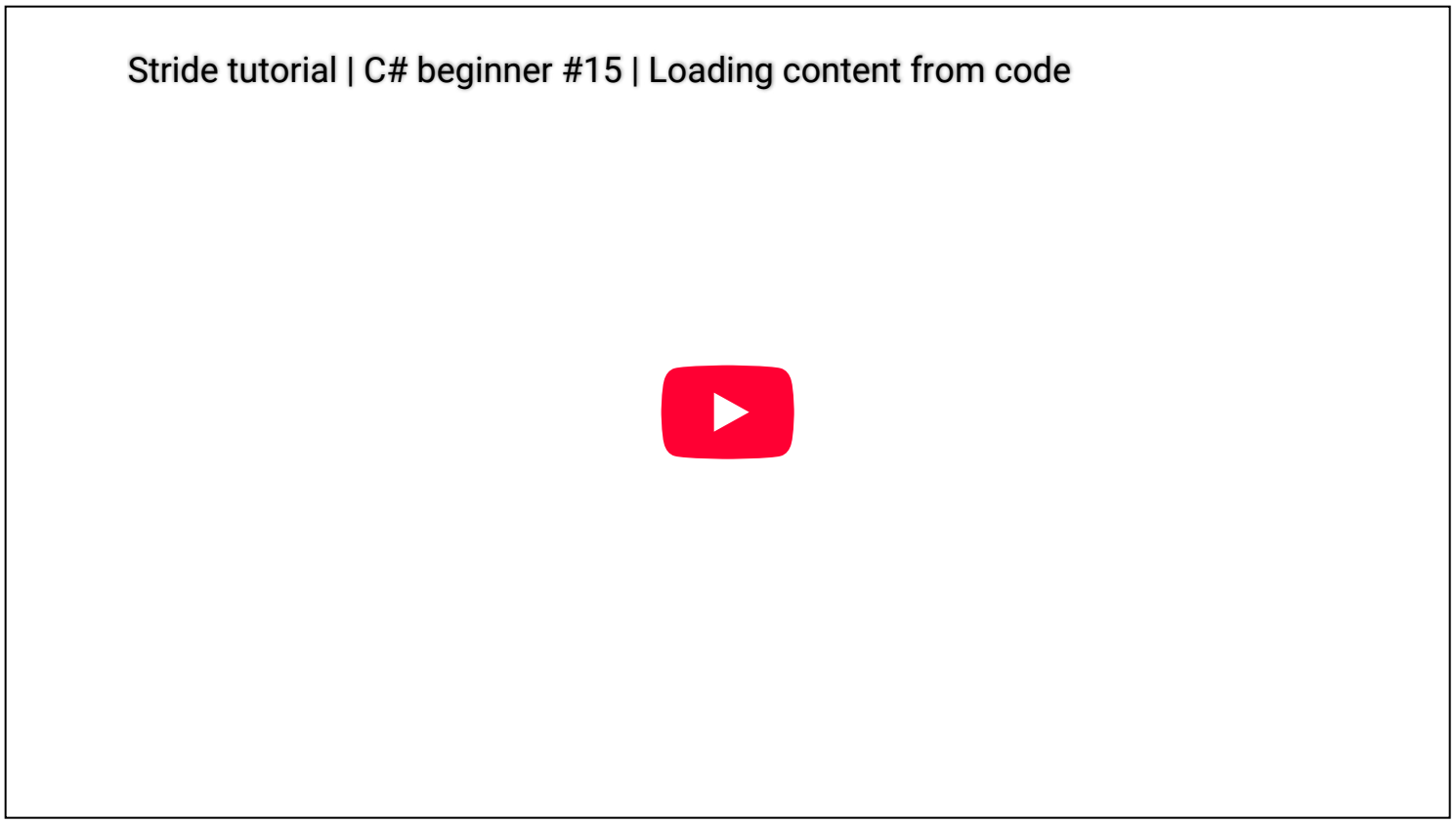
## Explanation

This C# Beginner tutorial covers how to load content from code.

Assets like models, textures, sound etc can be loaded from during runtime. At that point we no longer speak of assets but of 'content'.

This tutorial specifically loads content of the `Model` type. Loaded content that is no longer required in your scene, should be unloaded again so save up memory. For more information on assets see Manage assets.






# Code

With the **L** and **U** key you can either Load or Unload the model of a mannequin. If there is a model loaded, you can use the **S** key to spawn a new entity with the loaded mannequin model.

The **C** clears all of the spawned entities in the scene. This demo demonstrates that when models are unloaded, any entities that reference the model are still existing in the scene.

```
using System;
using System.Collections.Generic;
using Stride.Core.Mathematics;
using Stride.Engine;
using Stride.Input;
using Stride.Rendering;

namespace CSharpBeginner.Code
{
    /// <summary>
    /// This script demonstrates how we can load contect from code, and attach it to
an entity
    /// <para>
    /// https://doc.stride3d.net/latest/en/tutorials/csharpbeginner/loading-content.html
    /// </para>
    /// </summary>
    public class LoadingContentDemo : SyncScript
```

```
    {
        private Model loadedMannequinModel = null;
        private Stack<Entity> spawnedEntities = new Stack<Entity>();
        private Random random = new Random();

        public override void Start() { }

        public override void Update()
        {
            // To load any content we use the Load method. First we need to specify the type
between the '< >'. The we provide the URL
            if (Input.IsKeyPressed(Keys.L))
            {
                loadedMannequinModel = Content.Load<Model>("Models/mannequinModel");
            }

            // To remove loaded content we use the unload method to remove all existing
models from the scene.
            // Note: when we remove content, we can no longer see the model, but the entity
still exists in the scene
            if (Input.IsKeyPressed(Keys.U))
            {
                Content.Unload(loadedMannequinModel);
                loadedMannequinModel = null;
            }

            // If the model has been loaded, create a new entity and randomly place it in
the scene
            if (Input.IsKeyPressed(Keys.S))
            {
                CreateEntityWithModelAndRandomlyPositionInScene();
            }

            // Clear all entities from the tutorial scene. This does not unload the model
            if (Input.IsKeyPressed(Keys.C))
            {
                while (spawnedEntities.Count > 0)
                {
                    Entity.Scene.Entities.Remove(spawnedEntities.Pop());
                }
            }

            DebugText.Print("Model is " + (loadedMannequinModel == null ? "not loaded" :
"loaded"), new Int2(340, 580));
            DebugText.Print("Press L to load the manequin model", new Int2(340, 600));
            DebugText.Print("Press U to unload the mannequin model", new Int2(340, 620));
```

```
            DebugText.Print("Press S to spawn an entity if the model has been loaded", new
Int2(340, 660));
            DebugText.Print("Press C to clear all spawned entities", new Int2(340, 680));
            DebugText.Print("Spawned entities count: " + spawnedEntities.Count, new
Int2(340, 700));
        }

        private void CreateEntityWithModelAndRandomlyPositionInScene()
        {
            if (loadedMannequinModel != null)
            {
                // Create a new model component that references the loaded mannequin model
                var modelComponent = new ModelComponent(loadedMannequinModel);

                // Get a random position near the center of the scene
                var randomPosition = new Vector3(random.Next(-2, 4), 0, random.Next(-2, 2));

                // Create a new entity and attach a model component
                var entity = new Entity("My new entity with a model
component", randomPosition);
                entity.Add(modelComponent);

                // Add the new entity to the current tutorial scene
                Entity.Scene.Entities.Add(entity);

                // We add the spawned entities to a stack to keep track of them
                spawnedEntities.Push(entity);
            }
        }
    }
}
```

# Instantiating Prefabs

You can find this sample in the tutorial project: **Menu → Instantiating prefabs**

## Explanation

This C# Beginner tutorial covers how to instantiate prefabs.

A prefab is a "master" version of an object that you can reuse wherever you need. When you change the prefab, every instance of the prefab changes too.

A prefab that is instantiated by code does not give you a new prefab object, but instead gives you a list of entities. As long as these entities are not added to the scene, they wont be visible and attached scripts will not be executed.

Stride tutorial | C# beginner #16 | Instantiating prefabs

# Code

```
using Stride.Core.Mathematics;
using Stride.Engine;

namespace CSharpBeginner.Code
{
    /// <summary>
    /// This script demonstrates how we can instantiate prefabs
    /// <para>
    /// https://doc.stride3d.net/latest/en/tutorials/csharpbeginner/instantiating-
prefabs.html
    /// </para>
    /// </summary>
    public class InstantiatingPrefabsDemo : SyncScript
    {
        public Prefab PileOfBoxesPrefab;
        public override void Start()
        {
            // A prefab can be instantiated. Is does not give you a new prefab, but instead
gives you a list of entities
            var pileOfBoxesInstance = PileOfBoxesPrefab.Instantiate();

            // An instantiated prefab does nothing and isn't visible untill we add it to
the scene
```

```
            Entity.Scene.Entities.AddRange(pileOfBoxesInstance);


            // We can also load a prefab by using the Content.Load method
            var pileOfBoxesPrefabFromContent = Content.Load<Prefab>("Prefabs/Pile
of boxes");
            var pileOfBoxesInstance2 = pileOfBoxesPrefabFromContent.Instantiate();

            // We add the entities to a new entity that we can use a parent
            // We can easily position and rotate the parent entity
            var pileOfBoxesParent = new Entity("PileOfBoxes2", new Vector3(0, 0, -2));
            pileOfBoxesParent.Transform.Rotation = Quaternion.RotationY(135);
            foreach (var entity in pileOfBoxesInstance2)
            {
                pileOfBoxesParent.AddChild(entity);
            }
            Entity.Scene.Entities.Add(pileOfBoxesParent);
        }

        public override void Update()
        {
            DebugText.Print("The original prefab", new Int2(310, 320));
            DebugText.Print("The prefab instance PileOfBoxes", new Int2(560, 370));
            DebugText.Print("The prefab instance PileOfBoxes2 with custom parent", new
Int2(565, 650));
        }
    }
}
```

# 📈 C# Intermediate

**11 lessons** **4 hours**

These tutorials cover various intermediate principles of using C# when working with the Stride game engine.

It is recommended that you complete all the C# Beginner tutorials before moving on to the intermediate tutorials.

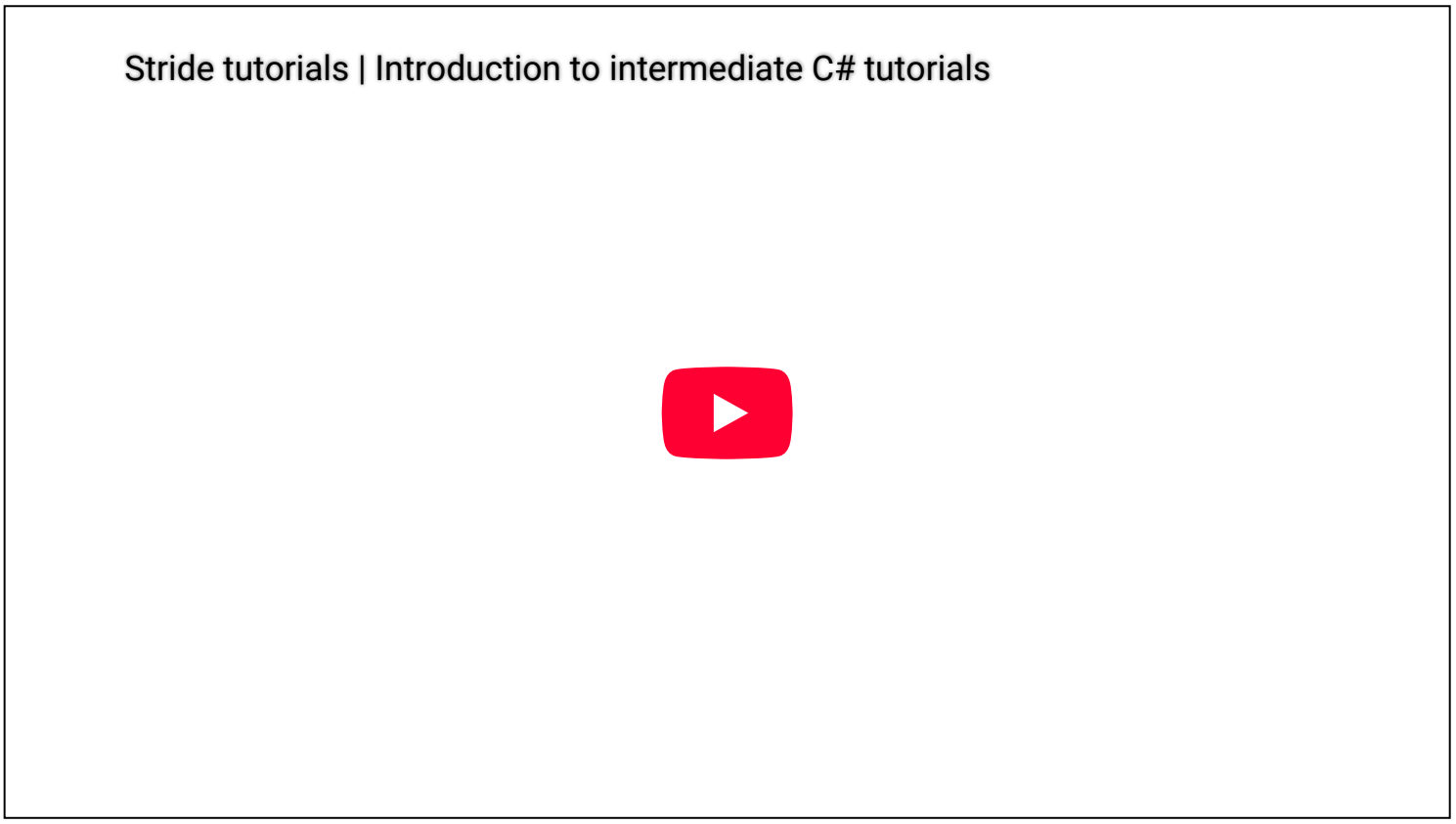
To create the C# intermediate tutorial project:

1. Start the Stride launcher
2. Create a new project
3. Select the template: Tutorials -> C# intermediate

Each tutorial has a 'Start' and a 'Completed' scene.

You can view the Completed scenes to see what the end result of each tutorial will roughly look like. If you are following along with the videos, the Start scenes serve as a good starting point. These scenes contain only the bare minimum setup.

## Stride C# intermediate YouTube tutorial series

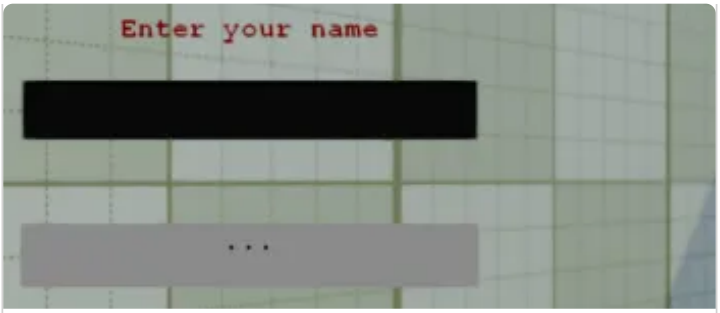



## All tutorials


## Introduction

A brief introduction to the C# intermediate tutorials for the Stride game engine.

📺 Watch the Introduction tutorial




## UI Basics

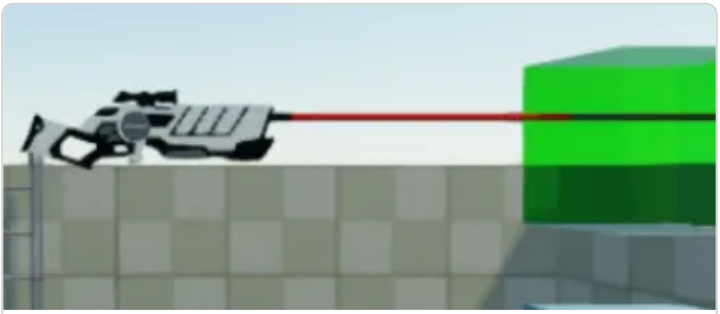
Learn about the UI editor, hooking up events, and creating UI by code.

📺 Watch the UI Basics tutorial

## Collision triggers

Explore colliders, trigger events, and colliding entities.

📺 Watch the Collision triggers tutorial



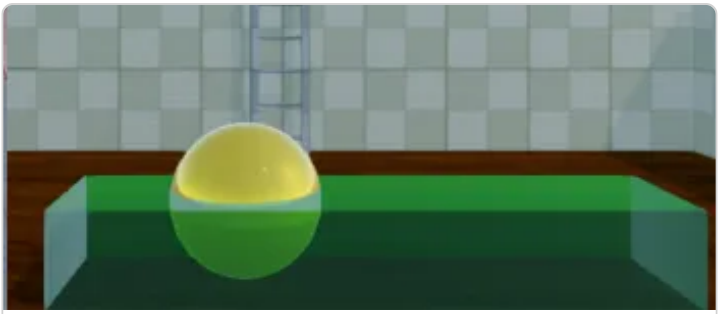
## Raycasting

Understand raycasting, collision groups, and penetrative raycasting.

📺 Watch the Raycasting tutorial


## Project and Unproject

Dive into projecting, unprojecting, and working with viewports.



## Async scripts

Discover asynchronous scripts, async collision triggers, and async web API usage.

Watch the Project and Unproject tutorial

## Scenes

Get familiar with child scenes, removing a scene, and (re)loading a scene.

Watch the Scenes tutorial

Watch the Async scripts tutorial

## Animation basics

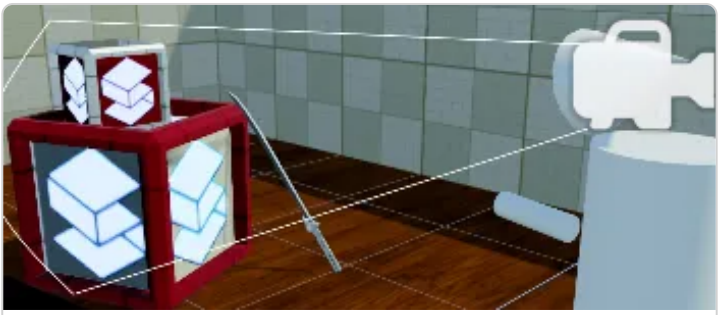
Master animation clips, playing and pausing, and cross-fading animations.

Watch the Animation basics tutorial

## Audio

Learn about sounds and music, spatialized sound, and streaming music.

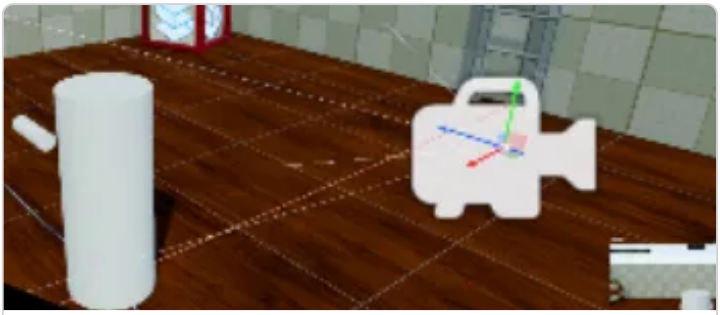
Watch the Audio tutorial



## First person camera

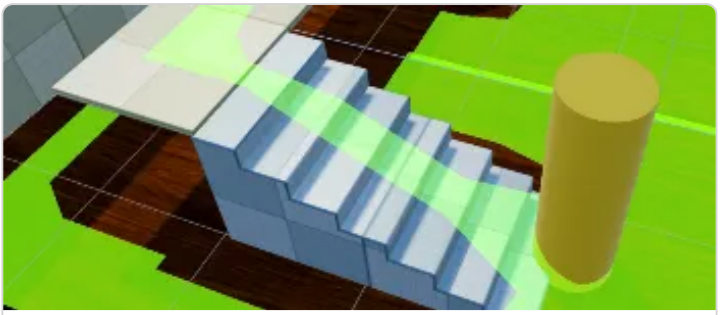
Explore mouse movement, applying rotation, and limited camera angles.

Watch the First person camera tutorial



## Third person camera

Understand third person offset, wall clamping, and first person fallback.



## Navigation

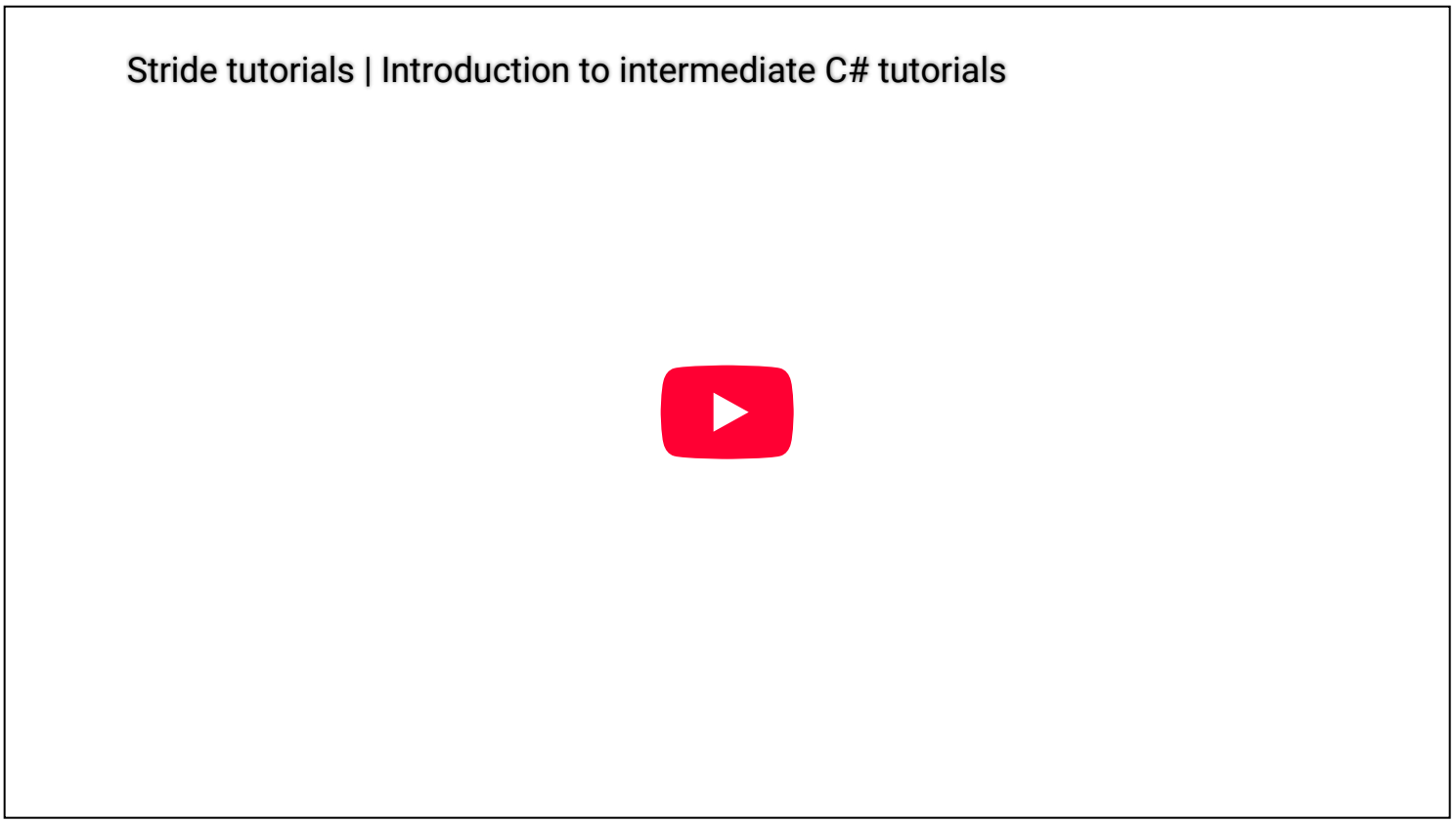
Dive into navigation meshes, navigation settings, and pathfinding.

📺 Watch the Third person camera tutorial

📺 Watch the Navigation tutorial

# Introduction

This is a brief introduction to the C# intermediate tutorials for the Stride game engine. We will cover how to set up the C# tutorials project, as well as the layout of the project and how it works.



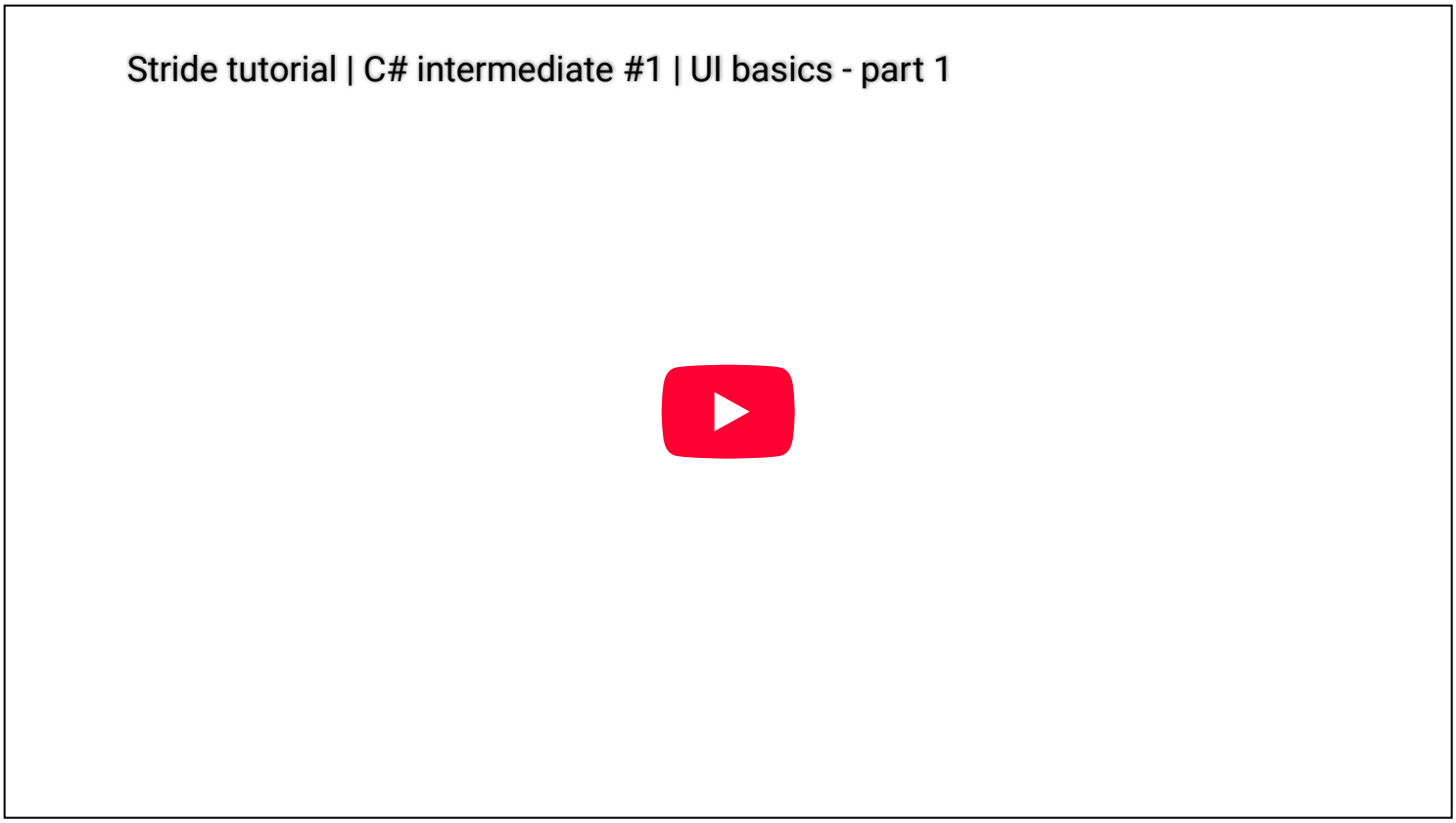
# UI Basics

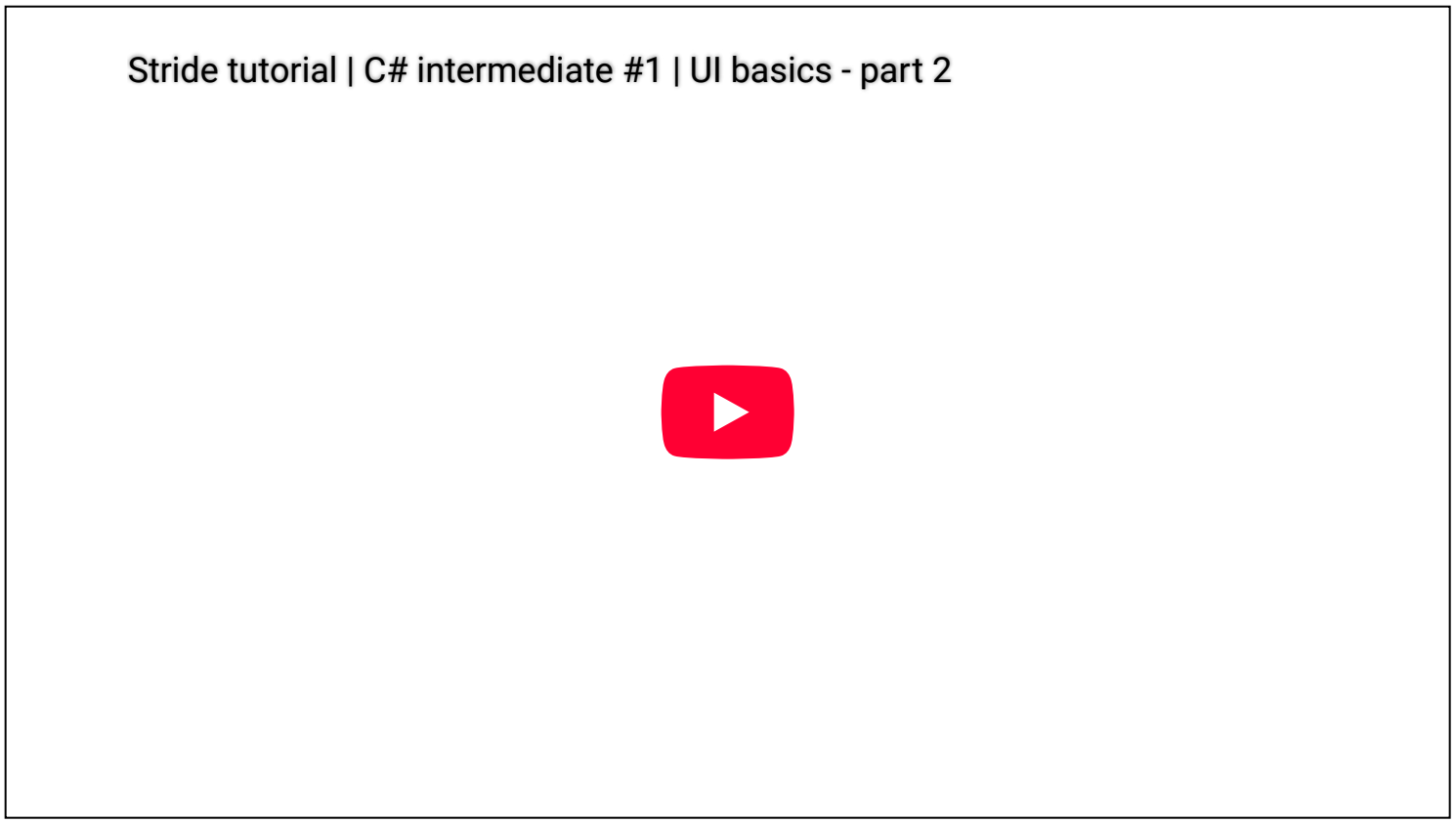
This first C# intermediate tutorial covers the basics of creating UI with Stride.

## Explanation

We will learn about the UI editor, accessing UI page elements and even how to setup UI entirely by code. The Stride editor comes with a UI editor which we can utilize to create UI pages. We can then add UI elements to these pages, like buttons and textfields.

Those UI elements can be referenced in code, so that can set up events like `button-clicked` or `text-changed`.






# Stride editor UI pages

The code below will look for a Page component that has been added to the current entity. On that page we search for UI elements like buttons and textfields. We than tell those UI elements what happens when we click on them, or that something needs to be done when a text value changes.

```
// Copyright (c) .NET Foundation and Contributors (https://dotnetfoundation.org/
& https://stride3d.net)
// Distributed under the MIT license. See the LICENSE.md file in the project root for
more information.
using Stride.Engine;
using Stride.Graphics;
using Stride.UI;
using Stride.UI.Controls;
using Stride.UI.Events;

namespace CSharpIntermediate.Code
{
    public class UIByEditor : StartupScript
    {
        public SpriteFont Font;

        private TextBlock textBlock;
        private EditText editText;
```

```
        public override void Start()
        {
            // Retrieve the page property from the UI component
            var page = Entity.Get<UIComponent>().Page;

            // Retrieve UI elements by Type and name
            textBlock = page.RootElement.FindVisualChildOfType<TextBlock>("MyTextBlock");
            editText = page.RootElement.FindVisualChildOfType<EditText>("MyEditText");

            // When the text changes, update the textblock
            editText.TextChanged += (s, e) =>
            {
                textBlock.Text = "My name is: " + editText.Text;
            };

            // When the button is clicked, we execute a method that clears the textbox
            var button = page.RootElement.FindVisualChildOfType<Button>("MyButton");
            button.Click += ButtonClicked;
        }

        private void ButtonClicked(object sender, RoutedEventArgs e)
        {
            // Changing the text triggers the TextChanged event again
            editText.Text = "";

            // We also want to reset the text in the textblock
            textBlock.Text = "...";
        }
    }
}
```

# UI pages made entirely by code

This script will create everything from scratch: a UI page, a stackpanel, a button, a textfield and the interactive logic behind it.

```
// Copyright (c) .NET Foundation and Contributors (https://dotnetfoundation.org/
& https://stride3d.net)
// Distributed under the MIT license. See the LICENSE.md file in the project root for
more information.
using System;
using Stride.Core.Mathematics;
using Stride.Engine;
using Stride.Graphics;
using Stride.UI;
```

```
using Stride.UI.Controls;
using Stride.UI.Panels;

namespace CSharpIntermediate.Code
{
    public class UIByCode : StartupScript
    {
        private SpriteFont font;
        private Button button;
        private TextBlock textBlock;

        public override void Start()
        {
            font = Content.Load<SpriteFont>("UI/OpenSans-font");
            button = CreateButton("Show me the time!");
            textBlock = CreateTextBlock("...");

            //We get or create a UI component and create a page with various elements
            var uiComponent = Entity.GetOrCreate<UIComponent>();

            uiComponent.Page = new UIPage
            {
                RootElement = new StackPanel
                {
                    Height = 200,
                    Width = 400,
                    Margin = new Thickness(0, 0, 10, 10),
                    VerticalAlignment = VerticalAlignment.Bottom,
                    HorizontalAlignment = HorizontalAlignment.Right,
                    BackgroundColor = new Color(0, 1, 0, 0.1f),
                    Children =
                    {
                        button,
                        textBlock
                    }
                }
            };
        }

        private Button CreateButton(string buttonText)
        {
            // We create a new button. The content of the button is a TextBlock
            var button = new Button
            {
                Name = "ButtonByCode",
                HorizontalAlignment = HorizontalAlignment.Center,
```

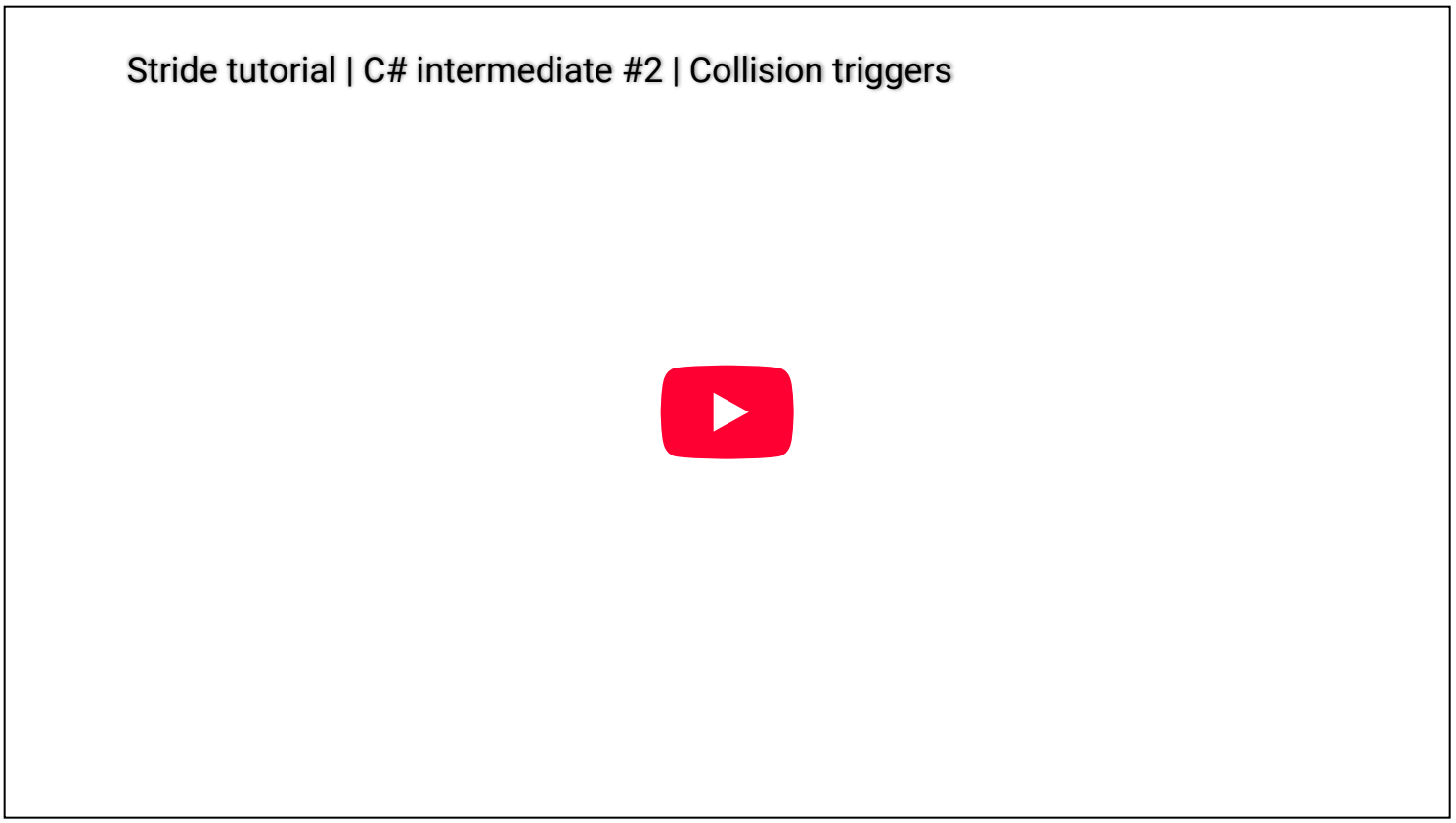
```
                BackgroundColor = Color.DarkKhaki,
                Content = new TextBlock {
                    Text = buttonText,
                    Font = font,
                    TextSize = 16,
                    TextColor = Color.Black,
                    VerticalAlignment = VerticalAlignment.Center
                }
            };

            // We send up the click event of the button
            button.Click += (sender, args) =>
            {
                textBlock.Text = $"Date: {DateTime.Now.ToShortTimeString()}";
            };

            return button;
        }

        private TextBlock CreateTextBlock(string defaultText)
        {
            var textBlock = new TextBlock
            {
                Name = "TextBlockByCode",
                Text = defaultText,
                Font = font,
                TextColor = Color.Yellow,
                BackgroundColor = Color.OrangeRed,
                HorizontalAlignment = HorizontalAlignment.Center
            };

            return textBlock;
        }
    }
}
```

# Collision triggers

This C# intermediate tutorial covers the use of collision triggers. It teaches about rigid bodies and how to set those up in the editor.

## Explanation

Rigid bodies determine how entities in our scene behave on gravity, whether they collide with other objects or in the case of this tutorial": trigger collision events in our code. We do this by setting up a collider box in our scene and letting a sphere roll through this object. The events that are triggered are then processed by the script that we will make for it.




## Rigidbodies and collisions

The code below looks for the rigidbody component that is attached to our entity. The rigidbody component contains all information we need for setting up triggers. The `IsTrigger` property determines that our collider doesn't stop other physics objects, but that it does trigger events in code (if they are set up at least).

We spawn a sphere which also has a rigidbody. This sphere has a mass and is affected by gravity. The sphere will fall down and eventually roll through our collider box. In our update loop we check if there are collisions happening. If there are collisions, we get the colliding object and print out some text on screen. Once the sphere leaves the trigger box, our update loop sees that we no longer have collisions.

Instead of using our update loop, we can also use collision events.

```
// Copyright (c) .NET Foundation and Contributors (https://dotnetfoundation.org/
& https://stride3d.net)
// Distributed under the MIT license. See the LICENSE.md file in the project root for
more information.
using System.Collections.Specialized;
using Stride.Core.Collections;
using Stride.Core.Mathematics;
using Stride.Engine;
using Stride.Physics;

namespace CSharpIntermediate.Code
{
    public class CollisionTriggerDemo : SyncScript
    {
        StaticColliderComponent staticCollider;
        string collisionStatus = "";

        public override void Start()
        {
            // Retrieve the Physics component of the current entity
            staticCollider = Entity.Get<StaticColliderComponent>();

            // When the 'CollectionChanged' event occurs, execute the
CollisionsChanged method
            staticCollider.Collisions.CollectionChanged += CollisionsChanged;
        }

        private void CollisionsChanged(object sender, TrackingCollectionChangedEventArgs
args)
        {
            // Cast the argument 'item' to a collision object
            var collision = (Collision)args.Item;

            // We need to make sure which collision object is not the Trigger collider
            // We perform a little check to find the ballCollider
            var ballCollider = staticCollider == collision.ColliderA ? collision.ColliderB
: collision.ColliderA;

            if (args.Action == NotifyCollectionChangedAction.Add)
            {
                // When a collision has been added to the collision collection, we know an
object has 'entered' our trigger
                collisionStatus = ballCollider.Entity.Name + " entered "
+ staticCollider.Entity.Name;
```

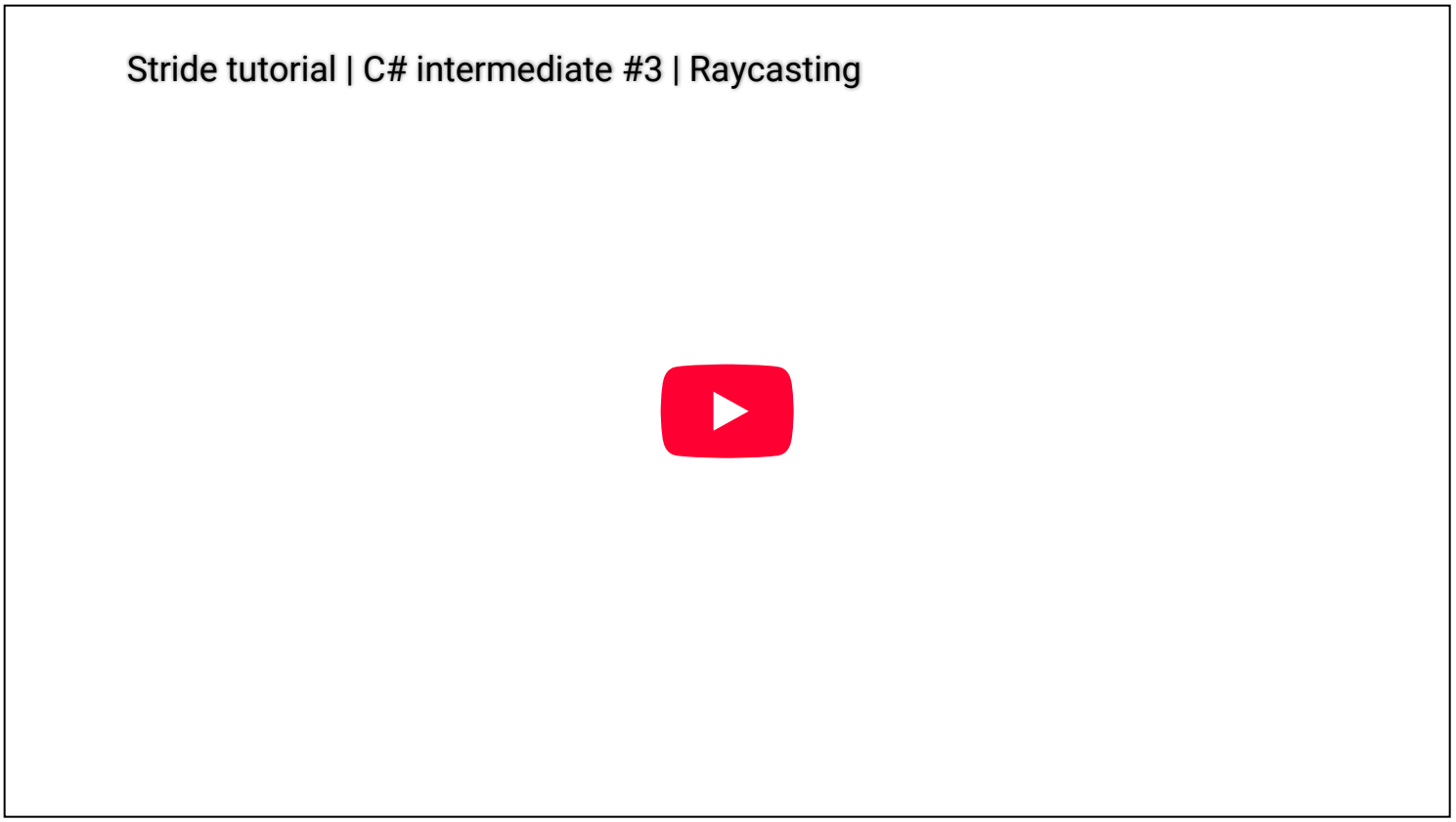
```
            }
            else if (args.Action == NotifyCollectionChangedAction.Remove)
            {
                // When a collision has been removed from the collision collection, we know
an object 'left' our trigger
                collisionStatus = ballCollider.Entity.Name + " left "
+ staticCollider.Entity.Name;
            }
        }

        public override void Update()
        {
            // The trigger collider can have 0, 1, or multiple collision going on in a
single frame
            int drawX = 500, drawY = 300;
            foreach (var collision in staticCollider.Collisions)
            {
                DebugText.Print("ColliderA: " + collision.ColliderA.Entity.Name, new
Int2(drawX, drawY += 20));
                DebugText.Print("ColliderB: " + collision.ColliderB.Entity.Name, new
Int2(drawX, drawY += 20));
            }

            DebugText.Print(collisionStatus, new Int2(500, 400));
        }
    }
}
```

# Raycasting

This C# Intermediate tutorial covers raycasting.

## Explanation

Raycasting is an essential subject in 3D games. With raycasts we can detect if and what kinds of objects are in our line of sight. This can be used for detecting enemies or how far an object really is.




## Raycast

This script sends out a raycast from the weapons barrel and sends it to an endpoint a little further. We check if we hit something along the way. If we do, we calculate the distance between the weapon barrel and the hit point. We then scale a laser to that distance to visualize the actual raycast. Depending on the collision group and filters, some objects are ignored.

```
// Copyright (c) .NET Foundation and Contributors (https://dotnetfoundation.org/
& https://stride3d.net)
// Distributed under the MIT license. See the LICENSE.md file in the project root for
more information.
using CSharpIntermediate.Code.Extensions;
using Stride.Core.Mathematics;
using Stride.Engine;
using Stride.Physics;
```

```
namespace CSharpIntermediate.Code
{
    public class RaycastDemo : SyncScript
    {
        public CollisionFilterGroupFlags CollideWithGroup;
        public bool CollideWithTriggers = false;
        public Entity HitPoint;

        private const float maxDistance = 4.0f;
        private Entity laser;
        private Simulation simulation;

        public override void Start()
        {
            //Store the physics simulation object
            simulation = this.GetSimulation();
            laser = Entity.FindChild("Laser");
        }

        public override void Update()
        {
            int drawX = 340;
            int drawY = 80;
            DebugText.Print("Press Q and E to raise/lower weapons", new Int2(drawX, drawY));

            var raycastStart = Entity.Transform.Position;
            var raycastEnd = Entity.Transform.Position + new Vector3(0, 0, maxDistance);

            drawY += 40;

            // Send a raycast from the start to the endposition
            if (simulation.Raycast(raycastStart, raycastEnd, out HitResult hitResult,
CollisionFilterGroups.DefaultFilter, CollideWithGroup, CollideWithTriggers))
            {
                // If we hit something, calculate the distance to the hitpoint and scale the
laser to that distance
                HitPoint.Transform.Position = hitResult.Point;
                var distance = Vector3.Distance(hitResult.Point, raycastStart);
                laser.Transform.Scale.Z = distance;

                DebugText.Print("Hit a collider", new Int2(drawX, drawY));
                DebugText.Print($"Raycast hit distance: {distance}", new Int2(drawX, drawY
+ 20));
                DebugText.Print($"Raycast hit point: {hitResult.Point.Print()}", new
Int2(drawX, drawY + 40));
```

```
                DebugText.Print($"Raycast hit entity: {hitResult.Collider.Entity.Name}", new
Int2(drawX, drawY + 60));
            }
            else
            {
                // If we didn't hit anything, scale the laser to match the distance between
start and end
                HitPoint.Transform.Position = raycastEnd;
                laser.Transform.Scale.Z = Vector3.Distance(raycastStart, raycastEnd);
                DebugText.Print("No collider hit", new Int2(drawX, drawY));
            }
        }
    }
}
```

# Penetrative raycast

In our first script, the raycast returns to us as soon as it hits the first object along its path. We can also send out a raycast to an endpoint, and let it return to us when it has reached its endpoint. It gives us back a list of objects that it has hit along the way. This list can be empty but also exists out of various objects. Depending on the collision group and filters, some objects are ignored.

```
// Copyright (c) .NET Foundation and Contributors (https://dotnetfoundation.org/
& https://stride3d.net)
// Distributed under the MIT license. See the LICENSE.md file in the project root for
more information.
using System.Collections.Generic;
using Stride.Core.Mathematics;
using Stride.Engine;
using Stride.Physics;

namespace CSharpIntermediate.Code
{
    public class RaycastPenetratingDemo : SyncScript
    {
        public CollisionFilterGroupFlags CollideWithGroup;
        public bool CollideWithTriggers = false;

        private Entity laser;
        private const float maxDistance = 3.0f;
        private Simulation simulation;

        public override void Start()
        {
            simulation = this.GetSimulation();
```

```
            laser = Entity.FindChild("Laser");
        }

        public override void Update()
        {
            int drawX = 700;
            int drawY = 80;
            DebugText.Print("Raycast penetration demo", new Int2(drawX, drawY));

            var raycastStart = Entity.Transform.Position;
            var raycastEnd = Entity.Transform.Position + new Vector3(0, 0, -maxDistance);

            var distance = Vector3.Distance(raycastStart, raycastEnd);
            laser.Transform.Scale.Z = distance;

            var hitResults = new List<HitResult>();
            simulation.RaycastPenetrating(raycastStart, raycastEnd, hitResults,
CollisionFilterGroups.DefaultFilter, CollideWithGroup, CollideWithTriggers);

            drawY += 40;
            if (hitResults.Count > 0)
            {
                DebugText.Print($"Raycast has hit {hitResults.Count} object(s)", new
Int2(drawX, drawY));

                foreach (var hitResult in hitResults)
                {
                    drawY += 20;
                    DebugText.Print($"- Raycast has hit: {hitResult.Collider.Entity.Name}",
new Int2(drawX, drawY));
                }
            }
            else
            {
                DebugText.Print("No collider hit", new Int2(drawX, drawY));
            }
        }
    }
}
```

# Project and Unproject

This C# Intermediate tutorial covers projecting and unprojecting coordinates from 3D to 2D and vice versa.

## Explanation

When we want to 'convert' 3D coordinates to a 2D screen, we speak 'Projecting'. The other way around is called 'Unprojecting'. Both scenarios are fairly common in 3D games.

The 3D to 2D or projecting happens for instance when you have a 3d quest marker. When the target you need to travel to is somewhere in front of you in the world, then you want to draw a 2D quest marker on screen that gives you an indication of where in the 3D world that target is located.

From 2D to 3D is often used to convert a mouse coordinate into the looking direction of the camera. This can be used for firing a weapon or setting a target on a map when playing a strategy game.


## Project

```
// Copyright (c) .NET Foundation and Contributors (https://dotnetfoundation.org/
& https://stride3d.net)
// Distributed under the MIT license. See the LICENSE.md file in the project root for
more information.
using Stride.Core.Mathematics;
```

```
using Stride.Engine;
using Stride.Graphics;

namespace CSharpIntermediate.Code
{
    public class ProjectDemo : SyncScript
    {
        public Entity projectSphere;
        public Entity projectSphereChild;
        private CameraComponent camera;

        public override void Start()
        {
            camera = Entity.Get<CameraComponent>();
        }

        public override void Update()
        {
            var backBuffer = GraphicsDevice.Presenter.BackBuffer;
            var sphereProjection =
Vector3.Project(projectSphere.Transform.WorldMatrix.TranslationVector, 0, 0,
backBuffer.Width, backBuffer.Height,0, 8, camera.ViewProjectionMatrix);
            var sphereChildProjection =
Vector3.Project(projectSphereChild.Transform.WorldMatrix.TranslationVector, 0, 0,
backBuffer.Width, backBuffer.Height,0, 8, camera.ViewProjectionMatrix);

            // Similar method using Viewports
            //var viewport = new Viewport(0, 0, backBuffer.Width, backBuffer.Height);
            //var sphereProjection =
viewport.Project(projectSphere.Transform.WorldMatrix.TranslationVector,
camera.ProjectionMatrix, camera.ViewMatrix, Matrix.Identity);
            //var sphereChildProjection =
viewport.Project(projectSphereChild.Transform.WorldMatrix.TranslationVector,
camera.ProjectionMatrix, camera.ViewMatrix, Matrix.Identity);

            DebugText.Print($"Parent", new Int2(sphereProjection.XY()));
            DebugText.Print($"Child", new Int2(sphereChildProjection.XY()));
        }
    }
}
```

# Unproject

```
// Copyright (c) .NET Foundation and Contributors (https://dotnetfoundation.org/
& https://stride3d.net)
```

```
// Distributed under the MIT license. See the LICENSE.md file in the project root for
more information.
using Stride.Core.Mathematics;
using Stride.Engine;
using Stride.Graphics;
using Stride.Input;
using Stride.Physics;

namespace CSharpIntermediate.Code
{
    public class UnprojectDemo : SyncScript
    {
        private CameraComponent camera;
        public Entity sphereToClone;

        public override void Start()
        {
            camera = Entity.Get<CameraComponent>();
        }

        public override void Update()
        {
            if (Input.IsMouseButtonPressed(MouseButton.Left))
            {
                var backBuffer = GraphicsDevice.Presenter.BackBuffer;
                var viewport = new Viewport(0, 0, backBuffer.Width, backBuffer.Height);

                var nearPosition = viewport.Unproject(new
Vector3(Input.AbsoluteMousePosition, 0.0f), camera.ProjectionMatrix,
camera.ViewMatrix, Matrix.Identity);
                var farPosition = viewport.Unproject(new
Vector3(Input.AbsoluteMousePosition, 1.0f), camera.ProjectionMatrix,
camera.ViewMatrix, Matrix.Identity);

                var hitResult = this.GetSimulation().Raycast(nearPosition, farPosition);

                // If there is a hitresult, clone the sphere and place it on that position
                if (hitResult.Succeeded)
                {
                    var sphereClone = sphereToClone.Clone();
                    sphereClone.Transform.Position = hitResult.Point;
                    Entity.Scene.Entities.Add(sphereClone);
                }
            }
        }
```

```
    }
}
```

# Async scripts

This C# Intermediate tutorial covers the usage of asynchronous scripts or `async` scripts.

## Explanation

Up until this point every tutorial has been using `sync` scripts. That means that those scripts are executed right after each other. If one particular sync script would take 1 second to complete, our game would freeze that 1 second, until the update loop is complete. All of the previously made Sync scripts can be made into an Async script.

With Async scripts we can perform heavy duty operations or reach out to an api without it freezing our application. A game can be made entirely with either Sync or Async scripts, or a combination of them both.


## Retrieving data from a web api

A common use case for async scripts is retrieving data from a web API. Depending on the speed of the API and the amount of data to be retrieved, this can take up to somewhere between 20 milliseconds and 2 seconds.

```
// Copyright (c) .NET Foundation and Contributors (https://dotnetfoundation.org/
& https://stride3d.net)
// Distributed under the MIT license. See the LICENSE.md file in the project root for
```

```
more information.
using System.Collections.Generic;
using System.Net.Http;
using System.Text.Json;
using System.Threading.Tasks;
using Stride.Core.Mathematics;
using Stride.Engine;

namespace CSharpIntermediate.Code
{
    public class AsyncWebApi : AsyncScript
    {
        private List<OpenCollectiveEvent> openCollectiveEvents;

        public override async Task Execute()
        {
            openCollectiveEvents = new List<OpenCollectiveEvent>();

            while (Game.IsRunning)
            {
                int drawX = 500, drawY = 600;
                DebugText.Print($"Press A to get Api data from
https://opencollective.com/stride3d", new Int2(drawX, drawY));

                if (Input.IsKeyPressed(Stride.Input.Keys.G))
                {
                    await RetrieveStrideRepos();
                    await Script.NextFrame();
                }

                foreach (var openCollectiveEvent in openCollectiveEvents)
                {
                    drawY += 20;
                    DebugText.Print(openCollectiveEvent.Name, new Int2(drawX, drawY));
                }

                // We have to await the next frame. If we don't do this, our game will be
stuck in an infinite loop
                await Script.NextFrame();
            }
        }

        private async Task RetrieveStrideRepos()
        {
            // We can use an HttpClient to make requests to web api's
            var client = new HttpClient();
```

```
            HttpResponseMessage response = await
client.GetAsync("https://opencollective.com/stride3d/events.json?limit=4");

            if (response.StatusCode == System.Net.HttpStatusCode.OK)
            {
                // We store the contents of the response in a string
                string responseContent = await response.Content.ReadAsStringAsync();

                // We deserialze the string into an object
                openCollectiveEvents = JsonSerializer.Deserialize<List<OpenCollectiveEvent>>
(responseContent);
            }
        }

        public class OpenCollectiveEvent
        {
            public string Name { get; set; }

            public string StartsAt { get; set; }
        }
    }
}
```

# Async Collision trigger

In a previous tutorial we made a collision trigger script that would notify the user once an object is passing through it. We can make a similar script using Async script.

```
// Copyright (c) .NET Foundation and Contributors (https://dotnetfoundation.org/
& https://stride3d.net)
// Distributed under the MIT license. See the LICENSE.md file in the project root for
more information.
using System.Threading.Tasks;
using Stride.Engine;
using Stride.Physics;
using Stride.Rendering;

namespace CSharpIntermediate.Code
{
    public class AsyncCollisionTriggerDemo : AsyncScript
    {
        private Material yellowMaterial;
        private Material redMaterial;

        public override async Task Execute()
```

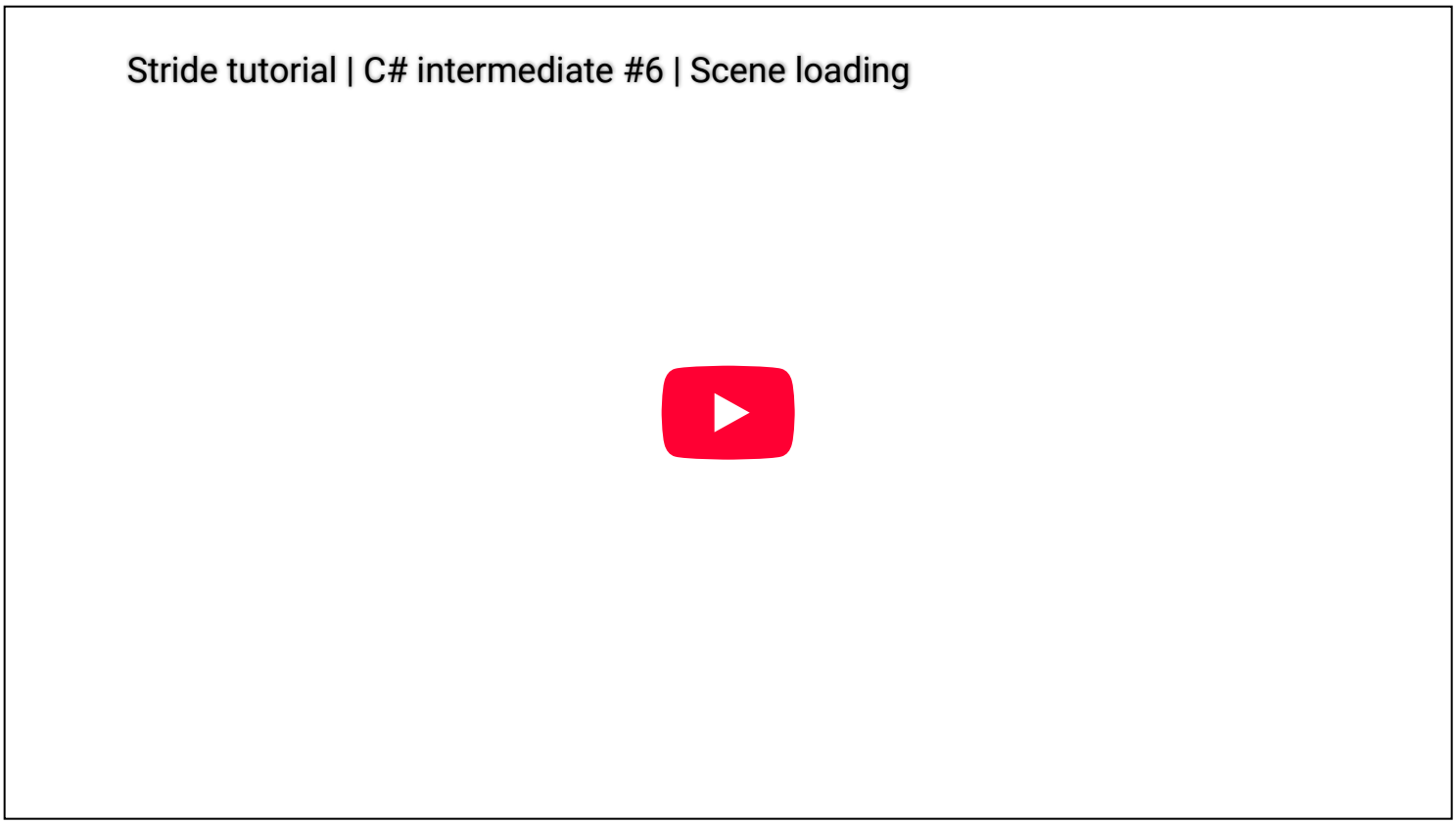
```
        {
            // Store the collider component
            var staticCollider = Entity.Get<StaticColliderComponent>();

            //Preload some materials
            yellowMaterial = Content.Load<Material>("Materials/Yellow");

            while (Game.IsRunning)
            {
                // Wait for an entity to collide with the trigger
                var collision = await staticCollider.NewCollision();
                var ballCollider = staticCollider == collision.ColliderA ?
collision.ColliderB : collision.ColliderA;

                // Store current material
                var modelComponent = ballCollider.Entity.Get<ModelComponent>();
                var originalMaterial = modelComponent.Materials[0];

                // Change the material on the entity
                modelComponent.Materials[0] = yellowMaterial;

                // Wait for the entity to exit the trigger
                await staticCollider.CollisionEnded();

                // Alternative
                // await collision.Ended(); //This checks for the end of any collision on
the actual collision object

                // Change the material back to the original one
                modelComponent.Materials[0] = originalMaterial;
            }
        }

        public override void Cancel()
        {
            Content.Unload(yellowMaterial);
            Content.Unload(redMaterial);
        }
    }
}
```

# Scenes

This C# Intermediate tutorial covers the concept of Scenes in Stride.

## Explanation

Stride allows Scenes to have an infinite amount of child scenes which on their terms also can load an infinite amount of child scenes. However, every scene loaded is unique. A scene can not be loaded twice at the same time. Both the editor and when loading scenes through code, will prevent a scene from being loaded twice at the same time.




## Loading a child scene

This script loads in a child scene by pressing a defined key. Pressing that same key again, will unload the loaded child scene. Every time we load the child scene again, we offset it a little in the Y direction to demonstrate the offsetting option for child scenes.

```
// Copyright (c) .NET Foundation and Contributors (https://dotnetfoundation.org/
& https://stride3d.net)
// Distributed under the MIT license. See the LICENSE.md file in the project root for
more information.
using Stride.Core.Mathematics;
using Stride.Core.Serialization;
using Stride.Engine;
```

```
using Stride.Input;

namespace CSharpIntermediate
{
    public class LoadChildScene : SyncScript
    {
        // We can load a scene by name, however if the scene would be renamed, this property
would not update
        //public string childSceneToLoad;

        public UrlReference<Scene> childSceneToLoad;
        private int loaded = 0;
        private Scene loadedChildScene;

        public override void Update()
        {
            DebugText.Print("Press C to load/unload child scene", new Int2(20, 60));
            if (Input.IsKeyPressed(Keys.C))
            {
                if (loadedChildScene == null)
                {
                    // loadedChildScene = Content.Load<Scene>(childSceneToLoad);
                    // Or
                    loadedChildScene = Content.Load(childSceneToLoad);
                    loadedChildScene.Offset = new Vector3(0, 0.5f * loaded, 0);
                    loaded++;

                    // Entity.Scene.Children.Add(loadedChildScene);
                    // Or
                    loadedChildScene.Parent = Entity.Scene;
                }
                else
                {
                    // Entity.Scene.Children.Remove(loadedChildScene);
                    // Or
                    loadedChildScene.Parent = null;

                    Content.Unload(loadedChildScene);
                    loadedChildScene = null;
                }
            }
        }
    }
}
```

# (Re)loading a scene

We can get the top most scene in our world which is called the RootScene. If we unload that scene, we can then load in a completely new scene in order to swap or switch to a new scene. That same script can also be used to reload the same scene in case you want to restart your scene,

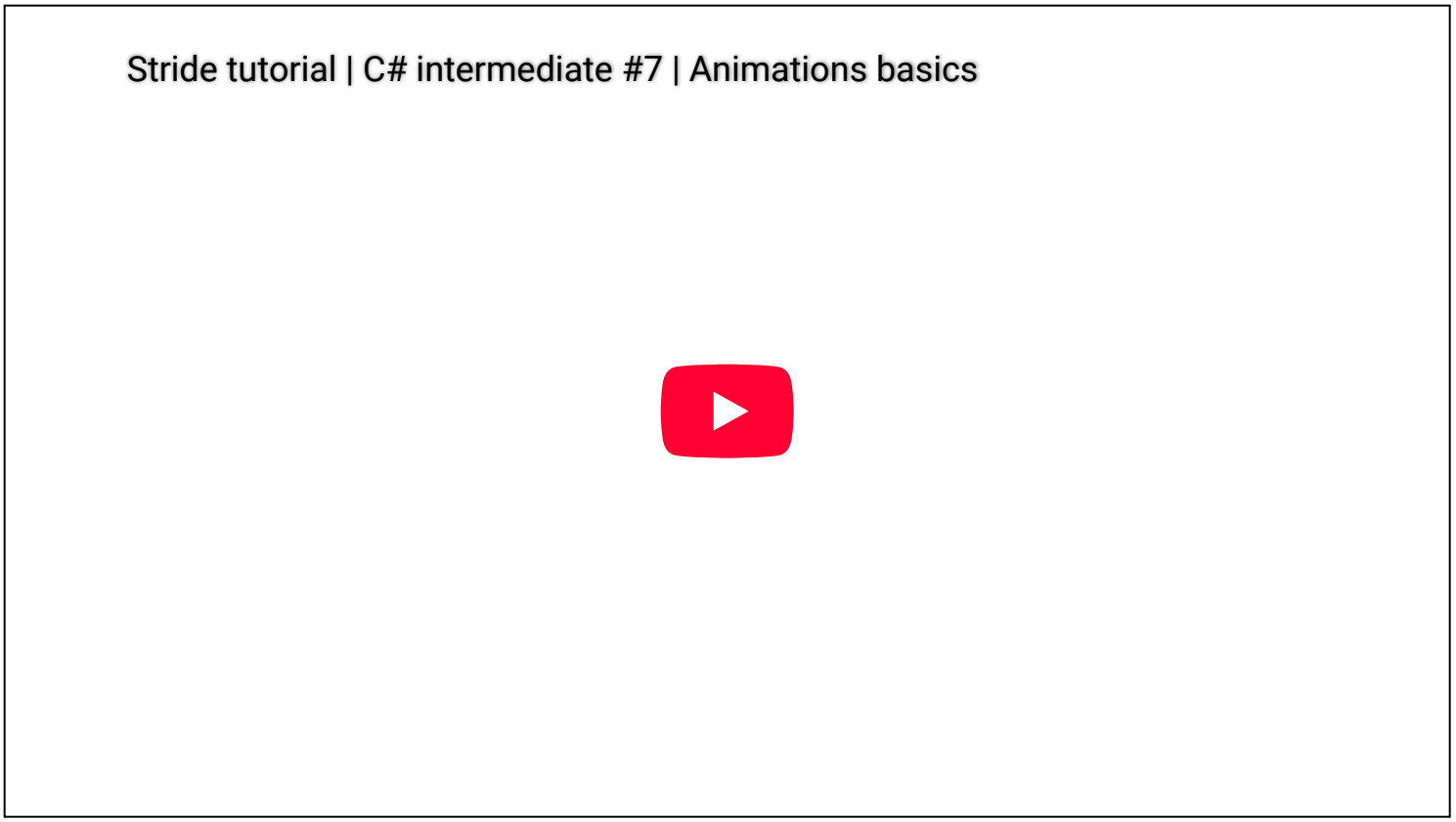
```
// Copyright (c) .NET Foundation and Contributors (https://dotnetfoundation.org/
& https://stride3d.net)
// Distributed under the MIT license. See the LICENSE.md file in the project root for
more information.
using Stride.Core.Mathematics;
using Stride.Core.Serialization;
using Stride.Engine;
using Stride.Input;

namespace CSharpIntermediate
{
    public class LoadScene : SyncScript
    {
        public UrlReference<Scene> SceneToLoad;
        public int DrawY = 20;
        public string Info = "Info text";
        public Keys KeyToPress;

        public override void Update()
        {
            DebugText.Print($"{Info}: {KeyToPress}", new Int2(20, DrawY));

            if (Input.IsKeyPressed(KeyToPress))
            {
                Content.Unload(SceneSystem.SceneInstance.RootScene);
                SceneSystem.SceneInstance.RootScene = Content.Load(SceneToLoad);
            }
        }
    }
}
```

# Animation basics

This C# Intermediate tutorial covers the basics of animation with Stride.

## Explanation

All animations exist as animation clips assets in your project. From there on we can start, pause and fade animations by using Stride's animation component.




## Code

```
// Copyright (c) .NET Foundation and Contributors (https://dotnetfoundation.org/
& https://stride3d.net)
// Distributed under the MIT license. See the LICENSE.md file in the project root for
more information.
using System;
using Stride.Animations;
using Stride.Core.Mathematics;
using Stride.Engine;
using Stride.Input;

namespace CSharpIntermediate.Code
{
    public class AnimationBasics : SyncScript
    {
```

```
        public float AnimationSpeed = 1.0f;
        private AnimationComponent animation;
        private PlayingAnimation latestAnimation;

        public override void Start()
        {
            animation = Entity.Get<AnimationComponent>();

            // Set the default animation
            latestAnimation = animation.Play("Idle");
        }

        public override void Update()
        {
            int drawX = 800, drawY = 600;

            StopOrResumeAnimations(drawX, drawY += 20);

            AdjustAnimationSpeed(drawX, drawY += 20);

            DebugText.Print("I to start playing Idle", new Int2(drawX, drawY += 20));
            if (Input.IsKeyPressed(Keys.I))
            {
                latestAnimation = animation.Play("Idle");
                latestAnimation.TimeFactor = AnimationSpeed;
            }

            DebugText.Print("R to crossfade to Run", new Int2(drawX, drawY += 20));
            if (Input.IsKeyPressed(Keys.R))
            {
                latestAnimation = animation.Crossfade("Run", TimeSpan.FromSeconds(0.5));
                latestAnimation.TimeFactor = AnimationSpeed;
            }

            // We can crossfade to a punch animation, but only if it is not already playing
            DebugText.Print("P to crossfade to Punch and play it once", new Int2(drawX,
drawY += 20));
            if (Input.IsKeyPressed(Keys.P) && !animation.IsPlaying("Punch"))
            {
                latestAnimation = animation.Crossfade("Punch", TimeSpan.FromSeconds(0.1));
                latestAnimation.RepeatMode = AnimationRepeatMode.PlayOnce;
                latestAnimation.TimeFactor = AnimationSpeed;
            }

            // When the punch animation is the latest animation, but it is no longer
playing, we set a new animation
```

```
            if (latestAnimation.Name == "Punch" && !animation.IsPlaying("Punch"))
            {
                latestAnimation = animation.Play("Idle");
                latestAnimation.RepeatMode = AnimationRepeatMode.LoopInfinite;
                latestAnimation.TimeFactor = AnimationSpeed;
            }
        }

        private void StopOrResumeAnimations(int drawX, int drawY)
        {
            DebugText.Print($"S to pause or resume animations", new Int2(drawX, drawY));
            if (Input.IsKeyPressed(Keys.S))
            {
                foreach (var playingAnimation in animation.PlayingAnimations)
                {
                    playingAnimation.Enabled = !playingAnimation.Enabled;
                }
            }
        }

        private void AdjustAnimationSpeed(int drawX, int drawY)
        {
            DebugText.Print($"Q and E for speed {AnimationSpeed:0.0}", new
Int2(drawX, drawY));
            if (Input.IsKeyPressed(Keys.E))
            {
                AnimationSpeed += 0.1f;
                latestAnimation.TimeFactor = AnimationSpeed;
            }
            if (Input.IsKeyPressed(Keys.Q))
            {
                AnimationSpeed -= 0.1f;
                latestAnimation.TimeFactor = AnimationSpeed;
            }
        }
    }
}
```

# Audio

This C# Intermediate tutorial covers the basics of audio in your game.

## Explanation

We learn about the various types of audio formats and settings. We cover how to use 3d spatialized audio and we also look at streaming audio.


## Audio sounds and spatial sounds

```
// Copyright (c) .NET Foundation and Contributors (https://dotnetfoundation.org/
& https://stride3d.net)
// Distributed under the MIT license. See the LICENSE.md file in the project root for
more information.

using Stride.Audio;
using Stride.Core.Mathematics;
using Stride.Engine;
using Stride.Input;

namespace CSharpIntermediate.Code
{
    public class AudioDemo : SyncScript
    {
```

```
        public Sound UkuleleSound;

        private SoundInstance ukuleleInstance;
        private AudioEmitterComponent audioEmitterComponent;
        private AudioEmitterSoundController gunSoundEmitter;

        public override void Start()
        {
            // We need to create an instance of Sound object in order to play them
            ukuleleInstance = UkuleleSound.CreateInstance();

            audioEmitterComponent = Entity.Get<AudioEmitterComponent>();
            gunSoundEmitter = audioEmitterComponent["Gun"];
        }

        public override void Update()
        {
            // Play a sound
            DebugText.Print($"U to play the Ukelele once", new Int2(200, 580));
            if (Input.IsKeyPressed(Keys.U))
            {
                ukuleleInstance.Stop();
                ukuleleInstance.Play();
            }

            // Press right mouse button for gun fire sound
            DebugText.Print($"Press right mouse button fire gun", new Int2(200, 640));
            if (Input.IsMouseButtonPressed(MouseButton.Right))
            {
                gunSoundEmitter.Play();
            }
        }
    }
}
```

# Streaming sounds

```
 // Copyright (c) .NET Foundation and Contributors (https://dotnetfoundation.org/
& https://stride3d.net)
// Distributed under the MIT license. See the LICENSE.md file in the project root for
more information.

using System;
using System.Threading.Tasks;
using Stride.Audio;
```

```
using Stride.Core.Mathematics;
using Stride.Engine;
using Stride.Input;
using Stride.Media;

namespace CSharpIntermediate.Code
{
    public class LoadMusic : AsyncScript
    {
        public Sound BackgroundMusic;

        private SoundInstance musicInstance;

        public override async Task Execute()
        {
            musicInstance = BackgroundMusic.CreateInstance();

            // Wait till the music is done loading
            await musicInstance.ReadyToPlay();

            while (Game.IsRunning)
            {
                // Play or pause
                DebugText.Print($"Space to play/pause. Currently:
{musicInstance.PlayState}", new Int2(800, 580));
                if (Input.IsKeyPressed(Keys.Space))
                {
                    if (musicInstance.PlayState == PlayState.Playing)
                    {
                        musicInstance.Pause();
                    }
                    else
                    {
                        musicInstance.Play();
                    }
                }

                // Volume
                DebugText.Print($"Up/Down to change volume: {musicInstance.Volume:0.0}", new
Int2(800, 600));
                if (Input.IsKeyPressed(Keys.Up))
                {
                    musicInstance.Volume = Math.Clamp(musicInstance.Volume + 0.1f, 0, 2);
                }
                if (Input.IsKeyPressed(Keys.Down))
                {
```

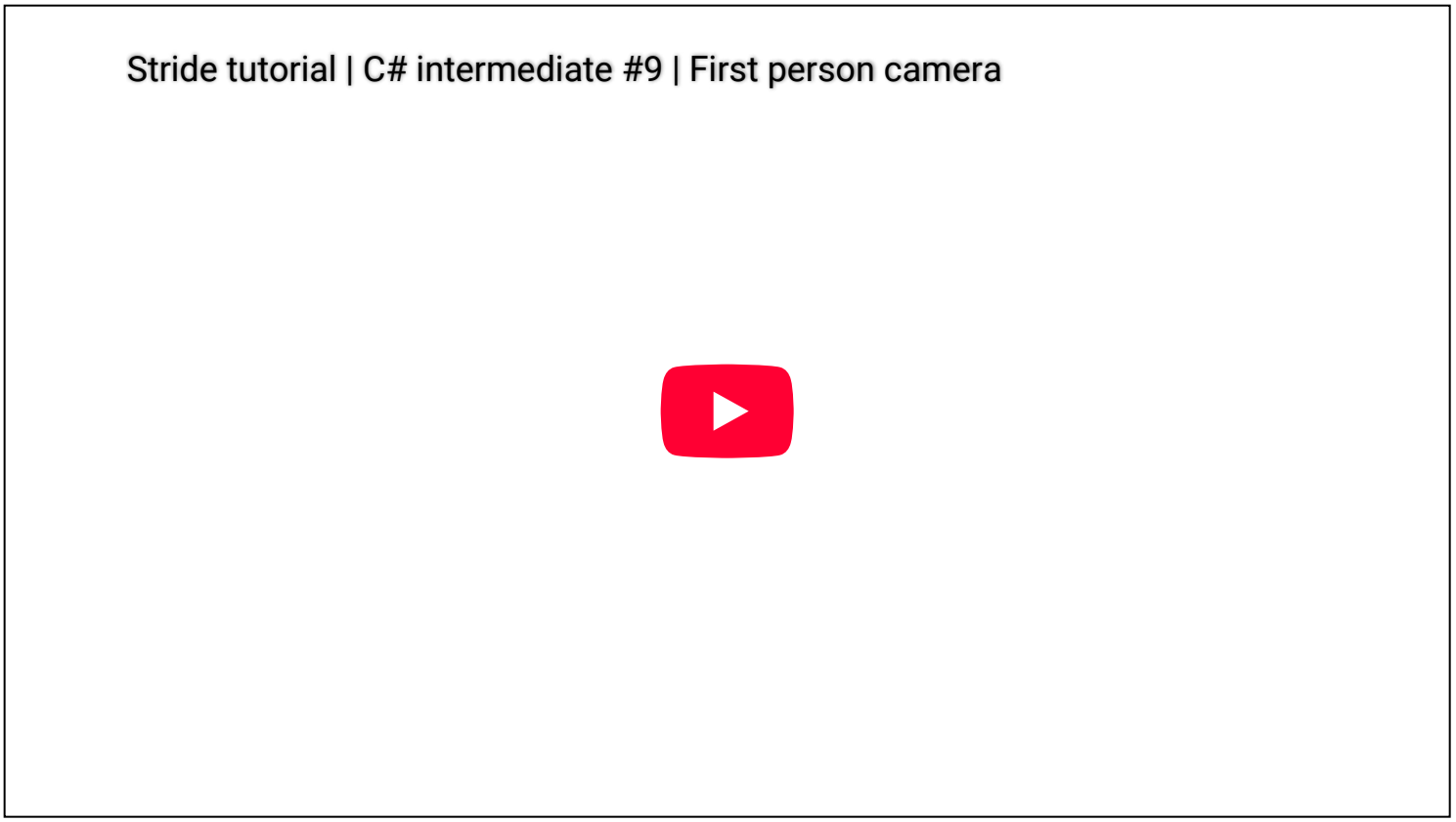
```
                musicInstance.Volume = Math.Clamp(musicInstance.Volume - 0.1f, 0, 2);
            }

            // Panning
            DebugText.Print($"Left/Right to change panning: {musicInstance.Pan:0.0}",
new Int2(800, 620));
            if (Input.IsKeyPressed(Keys.Left))
            {
                musicInstance.Pan = Math.Clamp(musicInstance.Pan - 0.1f, -1, 1);
            }
            if (Input.IsKeyPressed(Keys.Right))
            {
                musicInstance.Pan = Math.Clamp(musicInstance.Pan + 0.1f, -1, 1);
            }

            // Wait for next frame
            await Script.NextFrame();
        }
    }
}
```

# First person camera

This C# Intermediate tutorial covers the implementation of first person camera.

## Explanation

You learn about mouse movement and how to convert that into a 3d rotation. We set up camera angle limits and finally we apply movement to a first person character controller.




## Camera controller

```
// Copyright (c) .NET Foundation and Contributors (https://dotnetfoundation.org/
& https://stride3d.net)
// Distributed under the MIT license. See the LICENSE.md file in the project root for
more information.
using Stride.Core.Mathematics;
using Stride.Engine;
using Stride.Input;
using Stride.Physics;

namespace CSharpIntermediate.Code
{
    public class FirstPersonCamera : SyncScript
    {
        public float MouseSpeed = 0.6f;
```

```
        public float MaxLookUpAngle = -50;
        public float MaxLookDownAngle = 50;
        public bool InvertMouseY = false;

        private Entity firstPersonCameraPivot;
        private Vector3 camRotation;
        private bool isActive = false;
        private Vector2 maxCameraAnglesRadians;
        private CharacterComponent character;

        public override void Start()
        {
            firstPersonCameraPivot = Entity.FindChild("CameraPivot");

            // Convert the Max camera angles from Degress to Radions
            maxCameraAnglesRadians = new Vector2(MathUtil.DegreesToRadians(MaxLookUpAngle),
MathUtil.DegreesToRadians(MaxLookDownAngle));

            // Store the initial camera rotation
            camRotation = Entity.Transform.RotationEulerXYZ;

            // Set the mouse to the middle of the screen
            Input.MousePosition = new Vector2(0.5f, 0.5f);

            isActive = true;
            Game.IsMouseVisible = false;

            character = Entity.Get<CharacterComponent>();
        }


        public override void Update()
        {
            if (Input.IsKeyPressed(Keys.Escape))
            {
                isActive = !isActive;
                Game.IsMouseVisible = !isActive;
                Input.UnlockMousePosition();
            }

            if (isActive)
            {
                Input.LockMousePosition();
                var mouseMovement = Input.MouseDelta * MouseSpeed;

                // Update camera rotation values
```

```
                camRotation.Y += -mouseMovement.X;
                camRotation.X += InvertMouseY ? -mouseMovement.Y : mouseMovement.Y;
                camRotation.X = MathUtil.Clamp(camRotation.X, maxCameraAnglesRadians.X,
maxCameraAnglesRadians.Y);

                // Apply Y rotation to character entity
                character.Orientation = Quaternion.RotationY(camRotation.Y);
                // Entity.Transform.Rotation = Quaternion.RotationY(camRotation.Y);

                // Apply X camera rotation to the existing camera rotations
                firstPersonCameraPivot.Transform.Rotation =
Quaternion.RotationX(camRotation.X);
            }
        }
    }
}
```

# Character movement

```
// Copyright (c) .NET Foundation and Contributors (https://dotnetfoundation.org/
& https://stride3d.net)
// Distributed under the MIT license. See the LICENSE.md file in the project root for
more information.
using Stride.Core.Mathematics;
using Stride.Engine;
using Stride.Input;
using Stride.Physics;

namespace CSharpIntermediate.Code
{
    public class CharacterMovement : SyncScript
    {
        public Vector3 MovementMultiplier = new Vector3(3, 0, 4);
        private CharacterComponent character;

        public override void Start()
        {
            character = Entity.Get<CharacterComponent>();
        }

        public override void Update()
        {
            var velocity = new Vector3();
            if (Input.IsKeyDown(Keys.W))
            {
```

```
                velocity.Z++;
            }
            if (Input.IsKeyDown(Keys.S))
            {
                velocity.Z--;
            }

            if (Input.IsKeyDown(Keys.A))
            {
                velocity.X++;
            }
            if (Input.IsKeyDown(Keys.D))
            {
                velocity.X--;
            }

            velocity.Normalize();
            velocity *= MovementMultiplier;
            velocity = Vector3.Transform(velocity, Entity.Transform.Rotation);
            character.SetVelocity(velocity);
        }
    }
}
```

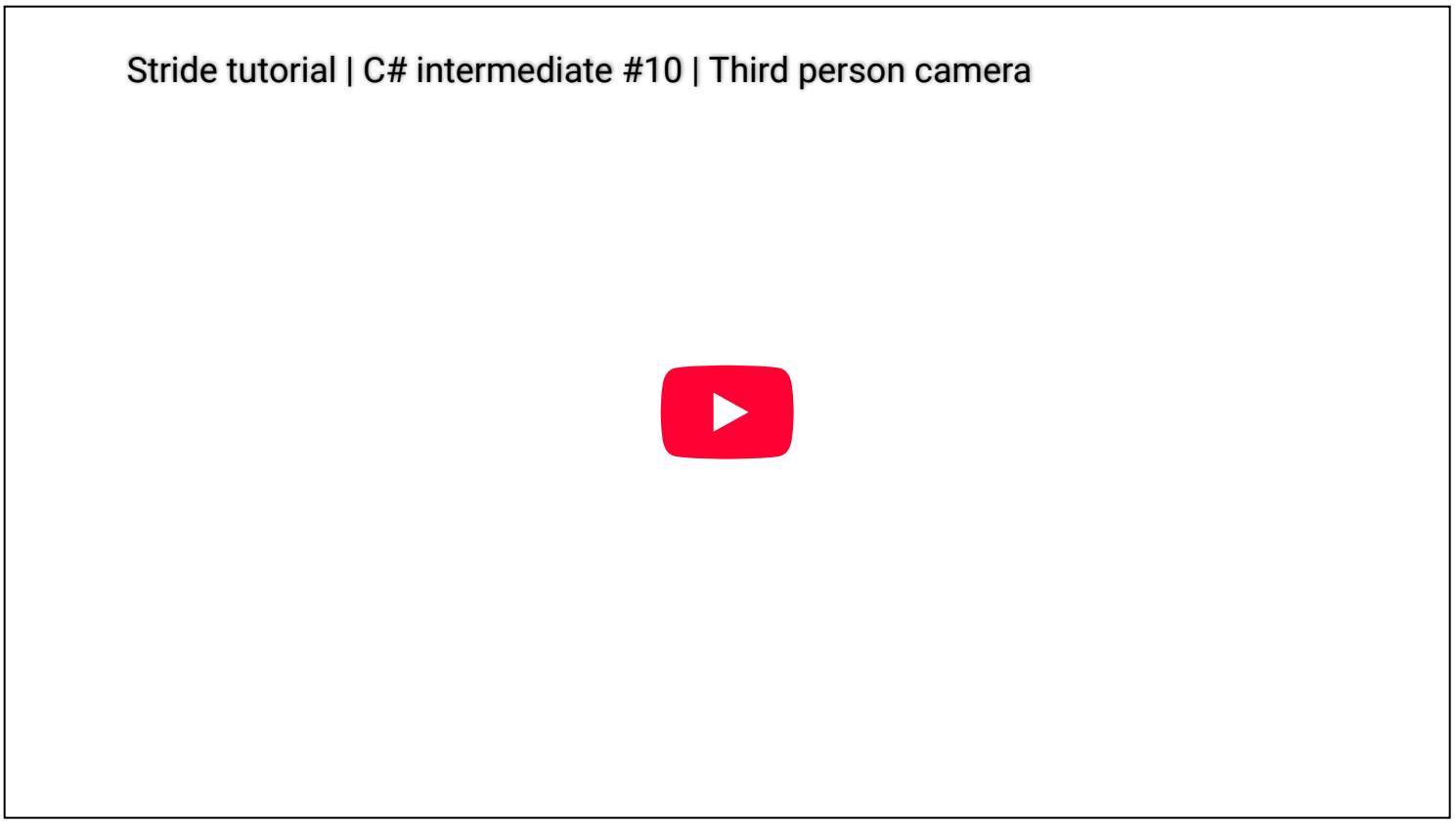
# Third person camera

This C# Intermediate tutorial covers the implementation of a third person camera.

## Explanation

Since it reuses a large portion of the First person camera, it is recommended that you watch that tutorial first.

This tutorial teaches about how to use raycasting to position the camera behind the player. If the player is to close any walls, the camera will be moved closer to the player. Too close to the player? We simply switch to first person mode.




## Third person camera

```
// Copyright (c) .NET Foundation and Contributors (https://dotnetfoundation.org/
& https://stride3d.net)
// Distributed under the MIT license. See the LICENSE.md file in the project root for
more information.
using Stride.Core.Mathematics;
using Stride.Engine;
using Stride.Input;
using Stride.Physics;

namespace CSharpIntermediate.Code
```

```
{
    public class ThirdPersonCamera : SyncScript
    {
        public bool InvertMouseY = false;
        public Vector2 MouseSpeed = new Vector2(0.6f, 0.5f);
        public float MaxLookUpAngle = -40;
        public float MaxLookDownAngle = 40;
        public float MinimumCameraDistance = 1.5f;
        public Vector3 CameraOffset = new Vector3(0, 0, -3);

        private Entity firstPersonPivot;
        private Entity thirdPersonPivot;

        private Vector2 maxCameraAnglesRadians;
        private Vector3 camRotation;
        private bool isActive = false;
        private Simulation simulation;
        private CharacterComponent character;

        public override void Start()
        {
            Game.IsMouseVisible = false;
            isActive = true;

            firstPersonPivot = Entity.FindChild("FirstPersonPivot");
            thirdPersonPivot = Entity.FindChild("ThirdPersonPivot");

            maxCameraAnglesRadians = new Vector2(MathUtil.DegreesToRadians(MaxLookUpAngle),
MathUtil.DegreesToRadians(MaxLookDownAngle));
            camRotation = Entity.Transform.RotationEulerXYZ;
            Input.MousePosition = new Vector2(0.5f, 0.5f);
            simulation = this.GetSimulation();
            character = Entity.Get<CharacterComponent>();
        }

        public override void Update()
        {
            if (Input.IsKeyPressed(Keys.Escape))
            {
                isActive = !isActive;
                Game.IsMouseVisible = !isActive;
                Input.UnlockMousePosition();
            }

            if (isActive)
            {
```

```
                Input.LockMousePosition();
                var mouseMovement = -Input.MouseDelta * MouseSpeed;

                // Update rotation values with the mouse movement
                camRotation.Y += mouseMovement.X;
                camRotation.X += InvertMouseY ? mouseMovement.Y : -mouseMovement.Y;
                camRotation.X = MathUtil.Clamp(camRotation.X, maxCameraAnglesRadians.X,
maxCameraAnglesRadians.Y);

                // Apply Y rotation to character entity
                character.Orientation = Quaternion.RotationY(camRotation.Y);

                // Apply X rotation the existing first person pivot
                firstPersonPivot.Transform.Rotation = Quaternion.RotationX(camRotation.X);

                // The third person pivot gets the same position and rotation as the first
person pivot + the camera offset
                thirdPersonPivot.Transform.Position = new Vector3(0);
                thirdPersonPivot.Transform.Position += CameraOffset;

                // Make sure that the WorldMatrix of the thirdperson pivot is up to date
                thirdPersonPivot.Transform.UpdateWorldMatrix();

                // Raycast from first person pivot to third person pivot
                var raycastStart = firstPersonPivot.Transform.WorldMatrix.TranslationVector;
                var raycastEnd = thirdPersonPivot.Transform.WorldMatrix.TranslationVector;

                if (simulation.Raycast(raycastStart, raycastEnd, out HitResult hitResult))
                {
                    // If we hit something along the way, calculate the distance
                    var hitDistance = Vector3.Distance(raycastStart, hitResult.Point);

                    if (hitDistance >= MinimumCameraDistance)
                    {
                        // If the distance is larger than the minimum distance, place the
camera at the hitpoint
                        thirdPersonPivot.Transform.Position.Z = -(hitDistance - 0.1f);
                    }
                    else
                    {
                        // If the distance is lower than the minimum distance, place the
camera at first person pivot
                        thirdPersonPivot.Transform.Position = new Vector3(0);
                    }
                }
            }
```

```
        }
    }
}
```

# Character movement

```
// Copyright (c) .NET Foundation and Contributors (https://dotnetfoundation.org/
& https://stride3d.net)
// Distributed under the MIT license. See the LICENSE.md file in the project root for
more information.
using Stride.Core.Mathematics;
using Stride.Engine;
using Stride.Input;
using Stride.Physics;

namespace CSharpIntermediate.Code
{
    public class CharacterMovement : SyncScript
    {
        public Vector3 MovementMultiplier = new Vector3(3, 0, 4);
        private CharacterComponent character;

        public override void Start()
        {
            character = Entity.Get<CharacterComponent>();
        }

        public override void Update()
        {
            var velocity = new Vector3();
            if (Input.IsKeyDown(Keys.W))
            {
                velocity.Z++;
            }
            if (Input.IsKeyDown(Keys.S))
            {
                velocity.Z--;
            }

            if (Input.IsKeyDown(Keys.A))
            {
                velocity.X++;
            }
            if (Input.IsKeyDown(Keys.D))
            {
```

```
                velocity.X--;
            }

            velocity.Normalize();
            velocity *= MovementMultiplier;
            velocity = Vector3.Transform(velocity, Entity.Transform.Rotation);
            character.SetVelocity(velocity);
        }
    }
}
```

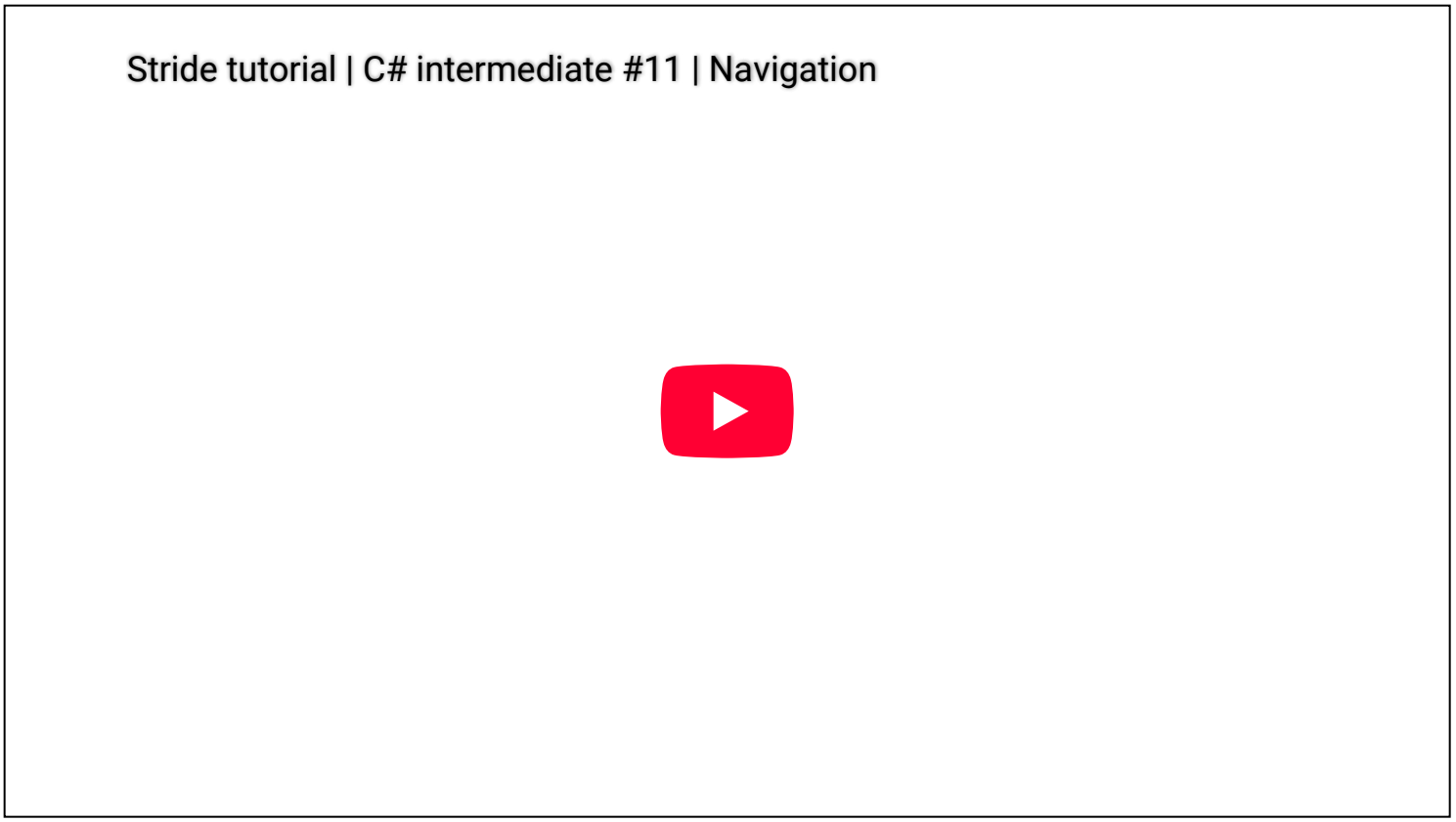
# Navigation

This C# Intermediate tutorial covers the basics of the navigation system in Stride.

## Explanation

In our world we can have so called 'navigation meshes'. These are meshes that are generated around your level geometry. The navigation mesh is used to calculate the quickest path to a destination.

We learn about the editors Navigation mesh settings, navigation bounding boxes and in code we learn how to move an object to a destination using the Navigation component that comes with the Stride engine.




## Code

```
// Copyright (c) .NET Foundation and Contributors (https://dotnetfoundation.org/
& https://stride3d.net)
// Distributed under the MIT license. See the LICENSE.md file in the project root for
more information.
using System;
using System.Collections.Generic;
using CSharpIntermediate.Code.Extensions;
using Stride.Core.Mathematics;
using Stride.Engine;
using Stride.Graphics;
```

```
using Stride.Input;
using Stride.Navigation;
using Stride.Physics;

namespace CSharpIntermediate.Code
{
    public class NavigateCharacter : SyncScript
    {
        public Entity RegularCharacter;
        public Entity PathSphere;
        public float MovementSpeed;

        private NavigationComponent navigationComponent;
        private List<Vector3> waypoints = new();
        private List<Entity> wayPointSpheres = new();
        private int waypointIndex = 0;

        public override void Start()
        {
            navigationComponent = RegularCharacter.Get<NavigationComponent>();
        }

        public override void Update()
        {
            DebugText.Print($"Left click to set Regular character target", new
Int2(200, 20));
            if (Input.IsMouseButtonPressed(MouseButton.Left))
            {
                CleanupExistingPath();
                SetTarget();
            }

            UpdateMovement();
        }

        private void UpdateMovement()
        {
            if (waypoints.Count == 0)
            {
                DebugText.Print($"No target", new Int2(200, 60));
                return;
            }

            var deltaTime = (float)Game.UpdateTime.Elapsed.TotalSeconds;
            var curPosition = RegularCharacter.Transform.WorldMatrix.TranslationVector;
            var nextWaypointPosition = waypoints[waypointIndex];
```

```
            var distanceToWaypoint = Vector3.Distance(curPosition, nextWaypointPosition);

            DebugText.Print($"Distance to waypoint {distanceToWaypoint.ToString("0.0")} ",
new Int2(200, 60));

            // When the distance between the character and the next waypoint is large
enough, move closer to the waypoint
            if (distanceToWaypoint > 0.1)
            {
                var direction = nextWaypointPosition - curPosition;
                direction.Normalize();
                direction *= MovementSpeed * deltaTime;

                RegularCharacter.Transform.Position += direction;
            }
            else
            {
                // If we are close enough to the waypoint, set the next waypoint or we are
done and we do a final cleanup
                if(waypointIndex+1 < waypoints.Count)
                {
                    waypointIndex++;
                }
                else
                {
                    CleanupExistingPath();
                }
            }
        }

        private void SetTarget()
        {
            // Determine the 3d position in our scene, based on were our mouse is
            var backBuffer = GraphicsDevice.Presenter.BackBuffer;
            var viewport = new Viewport(0, 0, backBuffer.Width, backBuffer.Height);
            var camera = Entity.Get<CameraComponent>();
            var nearPosition = viewport.Unproject(new Vector3(Input.AbsoluteMousePosition,
0.0f), camera.ProjectionMatrix, camera.ViewMatrix, Matrix.Identity);
            var farPosition = viewport.Unproject(new Vector3(Input.AbsoluteMousePosition,
1.0f), camera.ProjectionMatrix, camera.ViewMatrix, Matrix.Identity);

            var hitResult = this.GetSimulation().Raycast(nearPosition, farPosition);

            if (hitResult.Succeeded)
            {
                // Try to find the path to the hit point and store the path in the
```

```
Waypoints variable
                if (navigationComponent.TryFindPath(hitResult.Point, waypoints))
                {
                    waypointIndex = 0;

                    // For each waypoint create a waypoint sphere and add it to the scene
                    foreach (var waypoint in waypoints)
                    {
                        var waypointSphere = PathSphere.Clone();
                        waypointSphere.Transform.Position = waypoint;

                        wayPointSpheres.Add(waypointSphere);
                        Entity.Scene.Entities.Add(waypointSphere);
                    }
                }
            }
        }

        // Cleans up the waypoints in the scene and the waypoint information in mememory
        private void CleanupExistingPath()
        {
            foreach (var waypointSphere in wayPointSpheres)
            {
                Entity.Scene.Entities.Remove(waypointSphere);
            }
            wayPointSpheres.Clear();
            waypoints.Clear();
        }
    }
}
```

# ⚡ Quick Tutorials

**1 lesson** **4 minutes**

These tutorials provide bite-sized lessons to help you get up to speed with the Stride game engine in no time.

## Stride Quick tutorials YouTube series