CS50's
Introduction to Game Development

OpenCourseWare

Colton Ogden (https://www.linkedin.com/in/colton-ogden-0514029b/)
cogden@cs50.harvard.edu

David J. Malan (https://cs.harvard.edu/malan/)
malan@harvard.edu
(https://www.facebook.com/dmalan) (https://github.com/dmalan)
(https://www.instagram.com/davidjmalan/) (https://www.linkedin.com/in/malan/)
(https://orcid.org/0000-0001-5338-2522) (https://www.quora.com/profile/David-J-Malan) (https://www.reddit.com/user/davidjmalan)
(https://www.tiktok.com/@davidjmalan) (https://davidjmalan.t.me/)
(https://twitter.com/davidjmalan)

Lecture 8: Helicopter Game 3D

Today's Topics

- Unity
  - The ecosystem in which we'll be working for the remaining lectures.
- C#
  - The language we'll be primarily using alongside Unity.
- Blender
  - A free, open source package that we can use to create 3D models.
- Components
  - In the context of Unity, components are little pieces of behavior that can combine to drive the behavior of a larger object.
- Colliders and Triggers
  - In the context of our helicopter game, colliders and triggers will be relevant to how our helicopter interacts with the world (coins, buildings, planes, etc.).

- Prefabs and Spawning
  - Prefabs are pre-fabricated objects that you can customize and create in the Unity editor rather than having to code them up manually, and then spawn them in the world programmatically.
- Texture Scrolling
  - This is how we'll implement the infinite scrolling behavior for our game, similarly to how we did for Flappy Bird, but with a new approach this time.
- Audio
  - We'll take a look at how to add audio to our first Unity game.

Downloading demo code

- github.com/games50/helicopter (https://github.com/games50/helicopter)

Downloading Unity

- unity3d.com/get-unity/download (https://unity3d.com/get-unity/download)
- unity3d.com/unity/beta (https://unity3d.com/unity/beta)

Unity

- 3D and 2D game engine maintained and created by Unity Technologies.
- One of the top game engines in use, alongside engines like Unreal, Godot, CryEngine, and others.
- Free to use in its entirety with revenue-based restrictions (paid plans begin at $100k gross revenue).
- Very strong mobile and VR presence compared to other game engines.
- Primarily scripted in C#, a statically-typed object-oriented language.

C#

- Statically-typed object-oriented language created by Microsoft, very similar to Java.
- Primary language used to script any and all components and objects in Unity; prior languages Boo and UnityScript have been deprecated.
- Very widely used outside of Unity for games, GUI applications, .NET applications, and Mono applications.

Blender

- Completely free state-of-the-art 3D modeling software.
- Open-source.
- Not required to use in this course, but a tremendous tool to have as one starting out in 3D game development.

GameObjects

- The core class in Unity; everything is a GameObject.
- GameObjects are comprised of MonoBehaviours, which are effectively components in an Entity-Component System (ECS).
- MonoBehaviours are programmed in C# and give GameObjects their behavior by operating in tandem.

Components and the Unity Editor

- When looking at the Unity editor, "components" are found on the right-hand column (Transform, Camera, Flare Layer, etc.).
- As mentioned earlier, components essentially drive the behavior of GameObjects.
- If we create a brand-new GameObject using the Unity editor, we can manually assign components to it.
    - For example, if we give our GameObject a Transform, we can begin to interact with its position in the game.
    - If we assign it a camera component, we can modify how it looks in the game as well (e.g., a `perspective` projection would mirror how cameras look in the real world, whereas an `orthographic` projection would look more flat and less life-like).
- Another way in which components can be useful is by allowing us to organize GameObjects in complex ways.
    - For example, you might imagine that if we wanted to create a particular villain for our game, we might do so by using inheritance: a `Creature` class might be extended by a `Monster` class, a `Goblin` class, a `Goblin Chief` class, and so on.
    - However, by modeling our villain as a GameObject instead, we can then assign it multiple components: `CreatureComponent`, `MonsterComponent`, `GoblinComponent`, `ChiefComponent`, thereby achieving the same behavior in a more flexible way.
- Most of this can be done within the Unity editor. That is, without having to touch a single

line of code!

MonoBehaviours

- The way in which new components are created is by programming what are known as "MonoBehaviours".
  - Components and MonoBehaviours are essentially the same thing- the main distinction is that a MonoBehaviour is how a component is actually illustrated in code).
- MonoBehaviours are the core of all behavior in Unity; they are attached to GameObjects and updated on a frame-by-frame basis.
- MonoBehaviours implement a particular interface with pre-determined methods that Unity expects, such as `Update`, `Start`, `OnTrigger`, and more.
  - For more documentation, do refer to docs.unity3d.com/ScriptReference/MonoBehaviour.html (https://docs.unity3d.com/ScriptReference/MonoBehaviour.html).
- All customized behavior for every object in a game's scene will generally be implemented in code via some type of MonoBehaviour, with the exception of built-in components that can be modified through the GUI.

Colliders and Triggers

- A Collider defines the shape of an object in our scene for the purposes of 3D collision detection.
  - It's generally a good idea to keep Colliders simple (e.g., boxes, spheres, cylinders, etc.) so that collision-detection does not become an expensive computation in your game.
  - You might assign multiple simple Colliders to a complicated figure (a helicopter, a person, etc.) in order to create a fair hitbox.
- A Trigger is an object in our scene which will fire `OnTrigger` (implemented in a MonoBehaviour) when it detects collision with a Collider.
- By combining Colliders and Triggers, we can simulate many kinds of interactions between objects and the game space, as well as objects between objects.
  - In our helicopter game, the helicopter and the coins are examples of Colliders, whereas the airplanes and skyscrapers are example of Triggers.

Prefabs and Spawning

- Prefabs are "prefabricated" GameObjects that can be spawned at will in the game scene.
- Prefabs allow content creators to easily assemble all of the components they will want their GameObjects to have in the Unity Editor rather than in code and then spawn them in their code as needed.
- In our game, we have Prefabs for our airplanes, coins and skyscrapers, which we dynamically instantiate with `spawner` scripts.
  - These scripts contain `Start` and `Update` methods just like MonoBehaviours, but the main part of these scripts is contained within an `IEnumerator` method, which we haven't seen before.
  - This is a feature offered by a special type of function, known as a `Coroutine`.
  - At a high level, a Coroutine is essentially a function capable of "yielding" during its execution, rather than returning upon completion.
  - In our case, we use Coroutines in our spawner scripts to asynchronously spawn our Prefabs throughout the duration of the game.
  - To do so, our `IEnumerator` method loops infinitely, first instantiating a Prefab, and then yielding for a few seconds.
  - Our Prefabs themselves are stored in a `public` list, which by nature of being `public`, allows us to interact with it from within the Unity editor.

Texture Scrolling

- To achieve infinite scrolling in a 3D context, we need to associate a texture with a 3D object and then perform a translation of its UV coordinates.
- UV coordinates are effectively the mapping of a texture's image data to a 3D surface.
- By offsetting the UV coordinates by some amount over time (the X axis in our case), we can simulate the appearance of an infinitely scrolling surface, assuming our object is flat.

Audio

- In Unity, an AudioSource is the method by which we can trigger audio playback based on conditions in our code.
- AudioSource is just a component provided to us by Unity, so we can access it, once attached to an object, simply by grabbing it from the GameObject in question with `GetComponent()` and calling its `Play()` method (much like we've done with audio

sources in LÖVE until this point).

- To hear an AudioSource being played, an AudioListener needs to be present in the scene (usually attached to the default camera).

Asset Store

- Tremendous selection of free and paid assets to aid in game creation.
- Complete projects, models, effects, editor tools, and more.