Module 1.0

Getting Started With Unity

Introduction

Unity is a cross-platform game engine that provides an Integrated Development Environment (IDE) with standard *assets* for developing both 2-D and 3-D games. It is like a "Photoshop for game development" in that it doesn't require a lot of programming to develop complex games. Unity also features an extensive online community of developers as well as an extensive *asset store* that gives you access to a lot of free (and paid) content (graphics, models, audio, etc.). In this module, you will familiarize yourself with the basic usage of Unity.

1 Installation

For your convenience Unity is already installed on the machines in the classroom. Still, you may want to install it on your own laptop or home machine in order to develop wherever you like. Developing in Unity doesn't require a high-end machine, so whatever hardware you have will likely be sufficient.

Download and install **Unity Hub** from the following URL. This installs a manager application that allows you to download, install, and manage multiple versions of Unity.

`https://unity.com/download`

Start Unity Hub and create a Unity ID (email access required), then login to the Unity program. Install the Unity Editor (version 2020.3.25f1 as of 2022/01/07); this will install a .NET editor as well.

When you install, be sure to install all relevant "build support" modules (generally Windows will be sufficient, but you may want more).

2 Development

We'll start by running through a tutorial provided by the Unity development team. Many tutorials are available at `https://learn.unity.com/` but to get started, we'll run through the Roll-A-Ball tutorial:

`https://learn.unity.com/project/roll-a-ball`

In total, this may take 2 hours to complete. Watch the videos and develop along with them. Feel free to explore and make a few customizations if you like. Note that the tutorial uses an older version of Unity and so instructions (mostly for setup) may vary a bit.

3 Collaboration: Unity & GitHub

Eventually you will need to collaborate with your fellow students to work on your Unity project. There are several ways you can do this, but we'll cover how to use the *GitHub for Unity* (`https://unity.github.com/`) plugin available on the Unity Asset Store.[1] Git is a distributed version control system that allows you and your collaborators to make contributions and changes to a project and *push* them to a common *repository* that can be accessed anywhere. Likewise, changes your collaborators make can be *pulled* from the repository and *merged* with your local copy.

Note: for the remainder of this module, ask the instructor(s) to team you up with another students who has completed the Roll-a-Ball game. You should both follow the instructions below with your own accounts/projects until the instructions indicate you should do otherwise. **Help each other out as you go along**.

3.1 Getting Started

Before you begin:

- Sign up for a GitHub account: `https://github.com/`
- Install a git client. If you already have one great, if not we're going to recommend you install GitHub Desktop: `https://desktop.github.com/`

1. Open the Asset Store in Unity (either the tab or `Window` → `Asset Store`
2. Search for "GitHub for Unity" and download/import into your project.

[1] Alternatively, you can use git and/or GitHub independently or use Unity's "Unity Collaborate" feature. However, this limits the number of users you can collaborate on a project with.

3.1.1 Initializing Your Repo

A visualization of the following guide can be found in your project's GitHub plugin installation at:

`Assets/Plugins/GitHub/Editor/QuickGuide.pdf`

which is also available at:

https://github.com/github-for-unity/Unity/blob/master/docs/using/quick-guide.md#quick-guide-to-github-for-unity

1. Open the GitHub plugin (`Window` → `GitHub`
   a) Sign in the settings tab and
   b) Set your name/email in the settings tab
2. Initialize your repository: `Initialize` (tab) → `Initialize a git repository for this pro`
3. Make your first commit: in `Changes` select all files and below give a summary and description and click `Commit to [master]`
4. Click Publish to GitHub and give your repository a name (keep it public). For clarity, be sure you and your partner use different repository names.
5. Verify that it has been published to github by going to your GitHub account and viewing your project.

3.1.2 Commit & Push a Change: Add Moving Pick Up Objects

Using what you've learned in the tutorial, let's add some additional pickup objects that not only "float" and rotate, but that move around on the game board. In particular, we'll create a pick up object that moves along a pre-defined line off to the side.

1. Duplicate one of the "pick up" game objects and move it to $(7.5, 0.75, -7.5)$.
2. Add a new component (a C Sharp script) named `Pacer` and open it in an editor. Note: the `Rotator` script you wrote in the tutorial should remain; you can add as many scripts to game objects as you like.
3. Implement the script as given below.

```
using UnityEngine;
using System.Collections;

public class Pacer : MonoBehaviour {

  public float speed = 5.0f;
  private float zMax = 7.5f;
  private float zMin = -7.5f; //starting position
  private int direction = 1; //positive to start

  void Update () {
    float zNew = transform.position.z +
                 direction * speed * Time.deltaTime;
    if (zNew >= zMax) {
      zNew = zMax;
      direction *= -1;
    } else if (zNew <= zMin) {
      zNew = zMin;
      direction *= -1;
    }
    transform.position = new Vector3(7.5f, 0.75f, zNew);
  }
}
```

Let's walk through this script to understand what it does. The "Pick Up" object will travel along the z-axis between -7.5 and 7.5 (its x-axis and y-axis will remain unchanged). The speed determines how fast it "paces" the path. Since we start at $z = -7.5$ we will start by traveling in the positive direction, thus the `direction` variable is a positive 1.

In the `Update()` function, we calculate and set the new position. The new z-axis value will be the old value, `transform.position.z` plus the `direction` times the `speed` times `Time.deltaTime`. The `Time.deltaTime` is the amount of time it took to render the previous frame in Unity. Moving objects relative to time is better than moving them relative to the frame rate as the frame rate can be variable and would result in a different experience depending on the player's hardware.

Finally, we check to see if we've hit the maximum or the minimum z-axis values. If we have, we reset the z-axis value to the min or max and then change the direction by multiplying it by -1. Finally, we set the position to a new vector with the new z-axis value.

Once you are sure your changes are working, *commit* your changes and *push* them to GitHub by doing the following:

1. Go to the GitHub tab and go to the `Changes` subtab
2. Once again, select all (in general, you may only want to select the changes you want to commit instead of all of them) and include a Commit summary and description and click `Commit to [master]`
3. Your commit only commits changes to your *local repository* (remember: git is a *distributed* CVS, you have a local and remote repository and each collaborator has their own repository). Now you need to *push* your changes to GitHub: click

`Push (1)` at the top (the `(1)` indicates there is one commit to push).

4. You can observe/verify your changes by viewing them on GitHub

4 Collaborating

There are several ways you can use git/GitHub, but we'll cover how to collaborate by adding collaborators to your GitHub project.

1. In a browser, go to your GitHub project and click `Settings`.
2. Click `Collaborators`
3. Enter your partner's GitHub user name and click `Add collaborator`
4. Make sure your partner accepts the invitation.

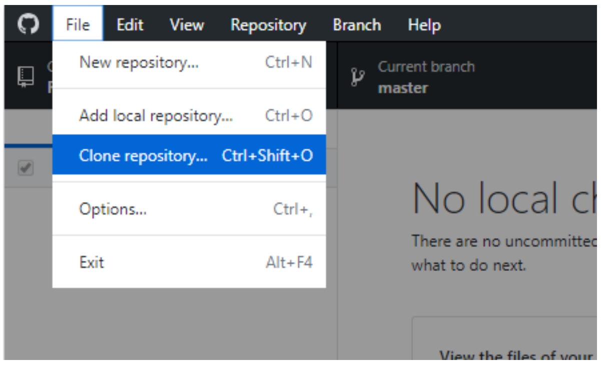
4.1 Modifying Your Partner's Project

In this subsection you and your partner will now make changes to *each other's* projects. So: close your project for now and we'll start working on your partner's project.

4.1.1 Checking out your partner's project

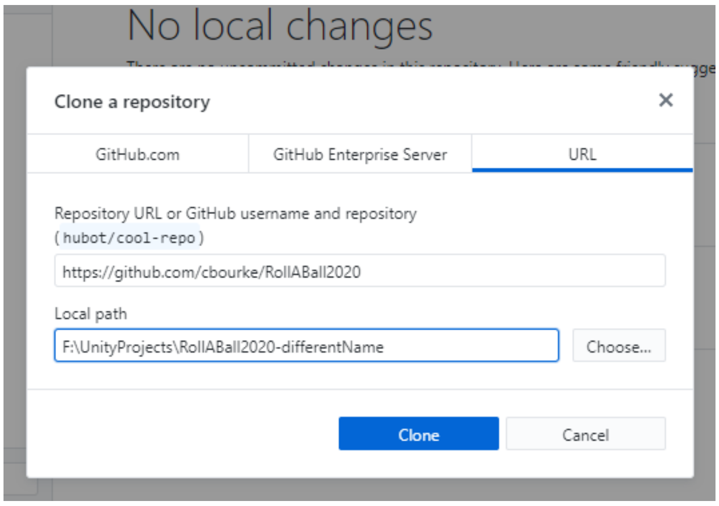
First, we need to *clone* your partner's project (hosted on GitHub) to your own computer. To do this you can use your own git client, but if you don't know what you're doing, we'll walk you through the process using the GitHub Desktop client (https://desktop.github.com/)[2]

1. Open GitHub Desktop and login with your GitHub credentials
2. Go to `File` → `Clone repository`



[2]No, the GitHub for Unity plugin does not have a clone feature. Yes, this really sucks. Yes, it is a known issue https://github.com/github-for-unity/Unity/issues/891

3. Select the `URL` tab



Repository URL or GitHub username and repository (`hubot/cool-repo`)

https://github.com/cbourke/RollABall2020

Local path

F:\UnityProjects\RollABall2020-differentName

4. Enter the URL for *your partner's* repository (that they added you as a collaborator on)
5. Be sure to set the `Local path` to the same directory as your other Unity project(s). This is why it was important to use a different name as your partner; if you failed to do this, you can rename your local repository.
6. Click clone (you may be prompted to "Initialize Git LFS" but select `Not now`.
7. You may now quit GitHub Desktop; open Unity
8. In Unity Hub, select `Add` and select the folder of the project you just cloned. Open the project. You should be able to start making changes to the project without any further setup.

4.1.2 Make Changes to your partner's project

Make a change to your partner's project. There are *lots* of creative and simple things you could do. Do *at least one* of the following suggestions and think of one additional change/feature of your own and implement it. You should work closely with your partner if you need help, but you each can do different things to each other's projects.

Once you have your working changes to your partner's project then remember to:

1. Commit your changes to your *local* repository
2. Push your changes to your partner's GitHub (remote) repository

- Add another pickup item that instead of traveling along a straight line, revolves around in a circle about the origin. Hint: use a radius r and an angle, $0 \leq \theta \leq 2\pi$ and then convert these polar coordinates to xz-coordinates using the following:

$$x = r \cdot \sin(\theta)$$

$$y = r \cdot \cos(\theta)$$

In C Sharp you can use `using System;` to bring in a floating-point math library with methods such as `Mathf.Sin()`.

- Add another pickup item that randomly appears and reappears in different locations at a reasonable pace.

  A simple way to do this is to keep a time counter and reset the xz-position to a random location when the counter hits a certain elapsed time (by adding `Time.deltaTime` on each frame render). Alternatively, you could increment a counter in a `FixedUpdate` method instead.

  A `System.Random` object can be used to generate random numbers, in particular the `NextDouble()` function returns a random number in the range $[0, 1)$.

  A more advanced technique is to use a *coroutine* which is a special method (routine) that can `yield` its execution back to the main game routine. Normally, a regular method (such as `Update()` must execute fully within a frame (thus computation intensive operations or procedural animations cannot be done without greatly affecting the gaming experience and frame rate).

  A coroutine can perform an operation and then yield control back to the main thread. At some point in the future, control is returned back to the coroutine at the point at which it yielded, allowing it to continue its execution. You can even yield for a specified amount of time using a `WaitForSeconds()` method. Full documentation an helpful examples can be found here: https://docs.unity3d.com/Manual/Coroutines.html

- Find a photo or an image of a pattern and then add it to your project as a new material for the ball and/or the ground. You may find the answer to the following question useful:

  http://answers.unity3d.com/questions/126746/make-pictures-into-a-material.html

- Research how to bring in rendered assets from a program such as Maya or 3D Studio Max and replace the pickup cubes or other objects with them.

- Research how to add sound effects and add one for when the ball picks up an object.

- Think of your own customizations that you can add/modify and then research how to do it.

4.2 Pull your partner's changes to your project

Once you have pushed your changes to your partner's project and your partner has pushed their changes to yours, close your partner's Unity project and reopen yours. Pull your partner's changes using the GitHub for Unity plugin. To do this, go to the GitHub for

Unity plugin (`Window` → `GitHub`) and click `Pull` . The changes and additions should now be part of your project (assuming no conflicts). Test your partner's changes and make sure they work.

Congratulations! You've successfully collaborated on a simple Unity game! Shake hands and move on (separately) to the next module!

Additional Notes

- Resolving merge conflicts can be challenging especially for binary assets. It is easiest to develop not on a separate branch but in a separate scene in Unity and then you can "merge" new features into your main scenes in your project.
- The GitHub plugin will automatically generate a default `.gitignore` file. It can be accessed in the root project folder.
- If you import a package from the Asset Store but only use parts of it, you should remove the parts you do not use so as not to commit them to your repo. You can hit the storage limits very quickly with binary artifacts.

Troubleshooting

- If you use your own git client be sure that you have git LFS (Large File Storage, https://git-lfs.github.com/) installed as well as Unity uses it.
- If you cannot commit changes because "Listing change files" keeps flashing and "refreshing" constantly, use the following workaround: https://github.com/github-for-unity/Unity/issues/963#issuecomment-518365134