unity devlog: jr programmer courses

adressing the formating

as i do courses my formating does change from start to finish, i will attempt to adhear to a standard or the standard however i will not be updating my formating on older doccuments so hand in there

unit 1.1 game 1 dev log

part 1/day 1

☑ welcome

    ☑ is a video saying that the course exists

☑ watch, then do

    ☑ is a video saying watch t hen do what the guy does for the tutuorals lol

☑ apparently the content will no longer be available as of July 31, 2023 soooooo that's my deadline :D

☑ install unity

    ☑ I'm using unity 2021.3.24f the lts version

    ☑ with visual studio code editor 2023 edition

    ☑ already logged in

☑ if u complete courses u gain xp????

    ☑ +10 Unity Essentials xp &

    ☑ +60 programming xp?????????

    ☑ u need 500 xp to level up past lvl 1

    ☑ kinda cringe imo

☑ UNIT 1 3d truck driving game where u avoid or crash into boxes

    ☑ I need this coz camera control and also 3d models not working in gnz

Educator Plan
By the end of this lesson, you will be able to:

- Create a new project through Unity Hub
- Navigate 3D space and the Unity Editor comfortably
- Add and manipulate objects in the scene to position them where you want
- Position a camera in an ideal spot for your game
- Control the layout of Unity Editor to suit your needs

☑ download Prototype 1 - Starter Files.zip

    ☑ does this link even work in the future???

    ☑ https://connect-prd-cdn.unity.com/20210923/c709e76b-3e93-4140-8675-f694b9f04399/Prototype%201%20-%20Starter%20Files.zip

☑ omg they talk so slow, 3x speed it is, tldr make new unity app/folder

- ☑ extract assets from starter files, then import the unity file in the zip
- ☑ the Project window, in *Assets* > *Scenes* > double-click on the Prototype 1 scene to open
- ☑ delete sample scene?
- ☑ roblox camera control to look around XD
    - ☐ whoa everything is so small , I was right my models were too big I'm gnz
- ☐
- ☐

day 2 unity jr programercourse

Man the guy talks slowly,
unit 1.1
important info

your home view if you will. it centers your camera on the object that you have selected.
god I love how smooooooooooooooooothe these Phoenix switches are on my keyboard and wish I could just put them on my k3 but no ;-;

the f key enables focus mode, this is like when u double click the middle mouse button in fusion to get to a decent reference frame for your object/file. it is primarily used to lock onto an object in space to move the camera and make it work

- alt left click drag allows you to rotate your camera around the object that you have selected
- alt right-click drag allows you to zoom but idk if I like using it but hay information is information

I can't believe it's an 8-minute video explaining this to the user

unit 1.1-4
in the transform component of an object you can select the menu button and click reset to reset the cords to all default

apparently, in unity, it's commonly accepted knowledge that each unit is one metre.

meaning yes my objects for my gnz project day 2/3 was indeed too big

they also tell us that we can rename the object in the component's window or in the hierarchy window

this happened on accident but you can maximize a screen by doing shift+space
they also want us to name the obstacle of our choice obstacle

a 3.5 min video saying

ctrl+p starts/stops play mode

and play mode does not save your game state
if you click the arrows on the cube on the top right part of the scene thoe arrows will align your
camera with the axis that you select.
also the circular tool by the move or pan tool is for rotation
the bottom one is for all types of transformation, scale rotation move

uhhh the project folder area if you don't need to see the assets but just need to see file
names/folder system the menu button for the project window has a 1 column layout for efficiency if
you don't have the screen real estate for your desired view
also on the top right you can click layout and change the layout by clicking the presets, personally,
I don't want to use those for what I'm doing but it might be useful for certain game layouts,
troubleshooting or games you are making

you can also save custom layouts

omg, finally I can do 1.2 :D
c# scripting?

just so its easier on other devs and you so it doesn't create problems down the line
DONT RENAME YOUR SCRIPTS
because if u do it might cause problems coz of how its written
just name it what its intended to do
then make more
or merge and fix the bugs

in the tutorial, u can just create a blank script in the project folder area and then drag it to the
object but I can't? maybe it's a new unity thing
BUT
 you can just create one as a new component
c# = c sharp or .cs
in the line of code which apparently doesn't need a ;
public class PlayerController : MonoBehaviour
so the : kind of means x inherits from y
so, in this case, the 3rd word "PlayerController" inherits from the class "MonoBehaviour"
so ig ill write in shorthand
idrk what the tutorial guy means but the void is a method but sure ill go with it
    void Start() is called before the game starts (sort of) or before the first "frame"

    void Update() is called every "frame"

ofc // means everything is a comment on that line from that // mark
It's common practice to capitalize certain things and not capitalize others, ex.
code with capital letters means its class
but lowercase letters are components
        transform.Translate(0 , 0, 1);

so you write code in this way

```
component.Method(arguments);
```

literally make it go vrooooomm and move forwards
it uses x y z and its 1 unit per frame

okok so now

4.Use a Vector3 to move forward
      transform.Translate(Vector3.forward);
is literally the same thing
but how do we make it so that its more than just along the z axis???
idk its just shorthand for that
we use the Vector3 library ig

we can modify that code to be like 20m/s with this
      transform.Translate(Vector3.forward * Time.deltaTime * 20);

coz delta means change in y
so delta time means change in time
so this means using the vector3 library we will move forward by 1 unit * 1 unit of time (second) * 20
or 1 unit per second * 20
or 20 units per second
or 20u/s
and since each unit is 1m
its 20m/s

OKOKOK
so now **RigidBody**'s
they are component that means the object its assigned to are a physics object, we can give it gravity and other things with this
sphere colider and mesh colider make it a coliable object and has different ways of interacting with the player/ a given object
mass is done in kilos :D

New Page

`public float _speed_ = 5.0f;`

the word public can be changed to private, this denotes the accessability of the variable/class/other

`private float _speed_ = 5.0f;`

also this is done to initialize the default value of the variable in the script's component tab
they are also known as access modifiers :D

when we do 5 as a variable, its only valid as an intiger, a whole number, so if we want to do fractional speeds we need to make it a floating point, a decimal number, so we could do 5.0, but its not understandable because its a decimal number

so since its not an intiger we need to use float to do decimals coz its a floating point number
so then can do 5.0f to turn 5.0 into a readable understandable number for the game
:D
what happens when we do dword :D it doesn't tell us ;-;

anyways next line of code
this is similar to the public float speed = thing BUT
it doesn't give us the number, or object its just a name

normally this doesn't work unless you assign a value in some programming languages but with c# if the script is assigned to an object ex. the camera you can drag an object into the player variable to reference what the player variable is, so ex we want the tank to be the player, (later the player model)

we can just put this code in the first level of the script in the public class and it will allow us to assign an object to the variable. whoa python is hard lmfao

    public GameObject Player;

ok so next line

in the void update() area

    transform.position = Player.transform.position;

so basically this transforming the position of the camera to be exactly the same as the Players, including its transformed(updated) position
so since we want to offset that
we can add

`+ new Vector3(0, 5, -7)`

we need to use a new vector3 variable (xyz)
but since we need to be able to change that

I messed with the code more and got the ideal maybe ggz angels

image.png

image not found or type unknown

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class FollowPlayer : MonoBehaviour
{

    public GameObject Player;
    private Vector3 offset;
    public float camera_x_offset = 0.0f;
    public float camera_y_offset = 5.0f;
    public float camera_z_offset = -7.0f;

    // Camera rotation offsets
    public float camera_x_rotation_offset = 0.0f;
    public float camera_y_rotation_offset = 0.0f;
    public float camera_z_rotation_offset = 0.0f;

    // Start is called before the first frame update
    void Start()
    {

    }

    // Update is called once per frame
    void Update()
    {
        offset = new Vector3(camera_x_offset, camera_y_offset, camera_z_offset);
        transform.position = Player.transform.position + offset;

        // Apply the rotation offsets
```

```
        Quaternion rotationOffset = Quaternion.Euler(camera_x_rotation_offset, camera_y_rotation_offset,
    camera_z_rotation_offset);

        transform.rotation = Player.transform.rotation * rotationOffset;

    }

  }
```

this still follows the player when they move away and jump but hayyy its what I want later on when I deny up/down / in / out movement

okok so important thing
to midiage or remove studder we could make a smoother with another script
OR fix the problem at the core sortof
do void LateUpdate() instead of Update()
this will make the camera's movement update be after the players position updates :D
BRO
someone should've told me sooner you can make the playmode color change wayyy more noticable

> From the top menu, go to Edit > Preferences (Windows) or Unity > Preferences (Mac)
>
> In the left menu, choose Colors, then edit the "Playmode tint" color to have a slight color
>
> Play your project to test it, then close your preferences

programming/unity essentials xp +5 each

temp

```
using System.Diagnostics;
using System.Collections;
using UnityEngine;

[DebuggerDisplay("{" + nameof(GetDebuggerDisplay) + "(),nq}")]
public class PlayerController : MonoBehaviour
{
    public float speed = 5.0f;
    // Start is called before the first frame update
    void Start()
    {

    }


    // Update is called once per frame
    void Update()
    {
        //move the vheicule forward
        transform.Translate(Vector3.forward * Time.deltaTime * speed);
    }

    private string GetDebuggerDisplay()
    {
        return ToString();
    }
}
```

Introduction to project management and teamwork + design documentation

What you will learn
Absolute Beginner Project Management
-Plan projects in the real-time development cycle
-Manage projects in the real-time development cycle

Summry
Most jobs in the design and development world require teamwork and will use project management and planning tools to ensure successful project delivery. In this tutorial, you'll be introduced to key concepts and best practices used for project planning, production phases, project management, and working in teams.

stolin unity design doc

[because its stolen here's a link to where I got it](#)

https://connect-prd-cdn.unity.com/20201117/bb422f61-9b42-4521-9828-910aae22e39c/Example%20of%20Project%20Charter.pdf

image.png

Image not found or type unknown

so ig this is what it would look like :D

Stolen Unity Design doc v2

again since its stolen heres the link

https://connect-prd-cdn.unity.com/20201117/99fd6c6b-0d44-42f5-9373-eb8a1887cacf/Project%20Design%20Doc%20TEMPLATE%20-%20[GDOC].pdf

image.png

image not found or type unknown

1.Overview

Project management and teamwork are often referred to as "soft skills" for design and development roles. Although project management can get technical, and there are several different project management methodologies you might encounter, it primarily takes teamwork — including communication and organization skills — to be a good project manager. Effective project management and teamwork involves:

- Clear communication with teammates, clients, and other stakeholders
- Delivery of a final product to deadline
- A final product or output that satisfies the requirements established for the project and meets the defined goal(s)

In this tutorial, you'll explore some introductory best practices for managing projects and working in teams.

2.Recap the phases of production

Individuals, teams, companies, and organizations will each have their particular methods for project management, but generally most projects break down into phases.

As you explored when learning about the **real time production cycle**, typically the phases of production are:

1. **Pre-production**: The work done before full-scale production begins, such as planning, prototyping, pipeline setup, and initial designs.

1. **Production:** Creation of the product and assets within it, including the creation of final 2D images and 3D models, audio, lighting, and user experience.

1. **Post-production:** Work done after the final product appears to be complete, including quality assurance (QA), editing, testing, bug fixing, and final polishing.

1. **Operations**: Ongoing work after a product has been released to keep it running, such as sales, monetization, updates, and continued maintenance.

image.png
image or type unknown

Properly planning for each phase of the production cycle is crucial when you need to deliver a project that is on time and works as expected.

3.Overview of project planning

hen working on projects, you need to plan appropriate schedules, manage time, communicate, and collaborate. Even if you're working on a project independently, it's likely that you'll be communicating with others at some point in the production process, especially when engaging in **user testing**.
Let's explore some guidance for successfully completing and delivering projects, as an individual or member of a larger team.
Documentation and tracking

Documentation plays a critical role in project management. A **design document** acts as a single source of truth, where you can record the specific requirements for the experience that you want to create. The design document also supports producers and project managers to track design and development progress.

When managing projects, consistently use design documents in the following ways to guide the design and development process:

- Identify the purpose, audience, and goals of your project.
- Identify the necessary project steps by creating a project plan.
- Within the project plan, create a timeline with specific deliverables and due dates.
- Consistently track your milestones in order to produce deliverables and meet deadlines.
- Assign roles when working in teams, and define and prioritize tasks for you and all teammates
- Make sure you and all teammates are following up and following through on roles and responsibilities.

Time management

Effective time management is critical to timely delivery and completion of the project.
When managing projects:

- Scope time for each phase of the design and development process.
- Review your project plan daily, and give brief daily status reports to the project manager or producer (often these daily updates are called "stand-ups").
- Use contingency planning for any unforeseen delays; reprioritize and update the tasks and deliverables in your project plans when needed.

- Identify which project planning and management tools are suitable for you or your team; when you've chosen them, use them consistently.

Communication

Without clear communication, collaboration is challenging at best - and impossible at worst. Communication is one of the most important skills for working with others in any capacity, and is vital for effective project management.

When communicating with others on your project:

- Be clear about your progress and any issues that impact the work of others, whether they are members of your team or supporting external collaborators.
- Be respectful of others' time — and of your own.
- When critiquing work, remain constructive and sensitive to the feelings of others. Focus on making your feedback helpful, specific, and respectful.
- Be open to feedback yourself by actively listening and engaging with the person delivering the feedback. Reflect honestly on how their feedback can be addressed.

Respect and professionalism

Even if you're completing a project for personal development or fun, when working with others it's important to respect them and have a professional attitude in your work.

When collaborating on a project with others:

- Be punctual.
- Promptly reply to collaborators, peers, and clients.
- Listen to others' opinions and contributions.
- Actively engage in collaborative work.

4.Design documents and project plans

In the pre-production phase at the start of the project, design documents are created to help define and scope what you are going to create. These documents include:

- Game (or Experience) Design Documents (GDDs)
- Target user personas
- Project charters
- Technical specifications

Design documents

Design documents contain the blueprint for your project. They include:

- High level overviews; for example, an overall project vision in the Game (or Experience) Design Document
- Requirements and standards for particular pipelines in the project
- Detailed design specifications for particular features

In your high level design document, you should identify:

- The goal and purpose for the project
- The intended users and audience
- Key features of the project
- The final form of delivery

A design document can include diagrams, flow charts, sketches, inspirational images from similar products, and reference images to capture the intended look and feel of the product.

If you don't already have a design document, you can download a PDF version of the design document template from the materials section or you can make a [Google document copy](#).

image.png
Image not found or type unknown

Project charters

Once the design document has been approved and the project is greenlit to be produced, a **project charter** is useful, especially when you are working in a team. This is a formal document that describes the entire project, including the project's objectives, how it will be carried out, and who the stakeholders are. The project charter is used in both the pre-production and production phases.

You can download a PDF version of an example project charter in the materials section or make a [Google document copy](#).

The project charter usually includes:

- Reasons for the project
- Objectives and constraints of the project
- Who the main stakeholders are
- Risks identified
- Benefits of the project
- General overview of the budger

Additional documentation

Depending on your project, you may need to create additional supporting documentation during the pre-production phase to help you scope and plan for production. You can iterate on these documents throughout pre-production and production, as needed.

These include:

- **Technical documentation:** This includes documents that specify the architecture and functionality of the technical parts of a project.
- **Meeting notes:** If you're working in a team, it's useful to capture meeting notes to record what the whole team is working on, dependencies, progress, and blockers.
- **Proposal or pitch document:** If you need to get approval from your company, or if you are asking investors to fund your project, a formal proposal or pitch document is recommended.

5.Managing projects and tracking progress

Once you have your project clearly scoped out, it is time to start planning how to manage a project by planning appropriate schedules and managing a **project plan**. While the project plan is usually created in pre-production, it is used and iterated upon throughout the production phase to track the project's progress.

The project plan takes the elements identified in the design document and project charter and sets specific tasks and deliverable dates. The producer or project manager will often create this plan and update it as necessary when changes need to be made. They will also create a **project tracker**, which can be part of the plan itself or created separately, to measure progress against the defined tasks and required deliverables. If you don't create and follow a project plan, the result is usually scope creep — the continuous adding of new ideas and features to your plan — which can lead to the project failing, being late, or going over budget.

Some guidelines to successfully track projects, for both individuals and teams, include:

- Identify the necessary project steps.
- If working in a team, identify and assign specific project roles and responsibilities.
- Create a timeline with specific deliverables and due dates.
- Identify common problems and issues in project management, such as scope creep and overly ambitious design plans, and tight time constraints.
- Determine when the completed project is due.
- Set due dates for each smaller piece of the project that will build to the overall project.
- Create reasonable time frames for each project phase.
- If working in a team, designate task owners for each task on the list.

Creating projects for clients

If you are creating a project for a client, there are some additional considerations that can help support effective project management:

- Identify clear project roles — this includes identifying the person who serves as the liaison between the client and the group, and key stakeholders for different elements of the project.
- Understand and clarify client expectations.
- Plan to support clear communication with the client — identify the ways you will communicate with your client and any online collaborative tools needed to review the project with their client.

6.Exercise: Identify tools to support your project management and tracking

There are a variety of project management tools that can help you to identify and plan all the steps of a project . Products like [Trello](#), [Favro](#), [Airtable](#), [Monday.com](#), [Google Apps](#), [Jira](#), and [GitHub](#) offer various features to support individuals and teams to plan and track their workstreams so that they execute and deliver on time. Many of these project management tools have templates that you can re-purpose to your particular project needs.

[image.png](#)
image could not be found or type unknown

If you don't already use one of these tools, take some time to investigate the options available to you and identify a tool to trial for your next project.

If you do already use one of these tools but it's not quite right for you, this is a great opportunity to investigate whether there are new options or features available that might better support you elsewhere.

Share your experience

If you've had a great experience with a particular tool, or discovered a really useful tool in your research, share your insights in the comments to support other learners!

7.Preparing to publish

In the post-production phase, you'll prepare to publish your project. Your project plan should account for quality assurance (QA) testing, content editing and revision, bug fixing, and final polishing of the deliverables.
Let's consider quality assurance testing in a little more detail.

Quality assurance testing

During the QA testing process, there are two important project management considerations: QA testing scripts and bug reporting processes.
Test scripts A QA test script is a series of steps for a tester to follow and record their observations. These scripts are usually created before formal testing starts, and are used to evaluate the product against specific project requirements and use cases. They may also be used to test the team's assumptions about the user experience of the product.

Bug report tracking Bug tracking and resolution is a vital part of polishing the product for delivery. The tools used to track bug reports vary — Jira and Github are popular choices, but many individuals and teams use a spreadsheet.
Typical bug reports include:

- **Title / brief summary:** A descriptive title for the bug.
- **Identified frequency**: How often does this bug occur (every time, most times, randomly, rarely)?
- **Reproduction steps**: Precise steps on how to reproduce the bug. This should be detailed enough that a developer can recreate the bug themselves without any additional information.
- **Detailed description:** A more detailed summary of the bug and its impact on user experience.
- **Any other observations**: These could include thoughts on what may be causing the bug and other relevant details (for example, "error occurs inconsistently, but more frequently at night than during the day").

If you want to learn more about bug reports, review The Art Of The Bug Report at the Ministry of Testing.

8.Releasing your project

Once your project has been tested, fixed, and validated, you are ready to publish. Your project plan should account for when and where you are publishing your product — ideally, you've been sticking to your deliverable date, and you are publishing your project on the date you originally identified.

Even if you don't hit the exact date you identified in pre-production, by following a clear project plan and making adjustments along the way, you should be within a reasonable range of your original release date.

9.Operations activities and retrospectives

Operations activities

The final phase of the production cycle, operations, includes work typically done after a product has been released to keep it running. Although operations activities generally don't start until the release date, they should be identified and planned during pre-production. These activities include ongoing sales, monetization, updates, and continued maintenance.

Project retrospectives

Finally, a retrospective should be planned once the project is launched so that you (and your team, if there was one) can reflect back on the design and development process. Retrospectives help you identify areas that went well, areas that didn't go well, and ways to improve for future projects. One simple approach to conducting a retrospective can be to make a chart with three columns and identify things that you think you should:

- Start doing
- Stop doing
- Continue doing

There are lots of tools and frameworks available for organizing and facilitating retrospectives if you'd like to find something a little different that best suits your particular project.

10.Summary

In this tutorial, you've explored the basics of project management. These are applicable to any project you work on: big or small, for real-time experience creation, or for any other kind of project! When you start your next project, use these recommendations to help you stay on track and deliver on time. Why not set up some documentation and a simple project plan or tracker for your next portfolio project?

Unit 2

https://learn.unity.com/tutorial/unit-2-introduction?uv=2021.3&pathwayId=5f7e17e1edbc2a5ec21a20af&missionId=5f71fe63edbc2a00200e9de0&projectId=5cdcc312edbc2a24a41671e6#5d1ba822edbc2a002175788d

Introduction

we will make a top down prototype where you shoot things that come after you, kindof like galiga except in 3d and not at all like galiga XD

Lesson 2.1 - Player Positioning

Summary

Overview:

You will begin this unit by creating a new project for your second Prototype and getting basic player movement working. You will first choose which character you would like, which types of animals you would like to interact with, and which food you would like to feed those animals. You will give the player basic side-to-side movement just like you did in Prototype 1, but then you will use if-then statements to keep the Player in bounds.

Project Outcome:

The player will be able to move left and right on the screen based on the user's left and right key presses, but will not be able to leave the play area on either side.

Materials

[Prototype 2 - Starter Files.zip](#)

Select your Unity version
2021.1 - 2021.3

1.Create a new Project for Prototype 2

- Open **Unity Hub** and create an empty "Prototype 2" project in your course directory on the correct Unity version. If you forget how to do this, refer to the instructions in **Lesson 1.1 - Step 1**

- Click to download the **Prototype 2 Starter Files**, **extract** the compressed folder, and then **import** the .unitypackage into your project. If you forget how to do this, refer to the instructions in **Lesson 1.1 - Step 2**

- From the Project window, open the **Prototype 2** scene and **delete** the SampleScene

- In the top-right of the Unity Editor, change your **Layout** from Default to your custom layout

2.Add the 3d assets + first few lines of code

1. If you want, drag a different **material** from *Course Library > Materials* onto the Ground object
2. Drag 1 **Human**, 3 **Animals**, and 1 **Food** object into the Hierarchy
3. Rename the character "Player", then **reposition** the animals and food so you can see them
4. Adjust the XYZ **scale** of the food so you can easily see it from above

- **Attach** the script to the Player and open it
- In your **Assets** folder, create a "Scripts" folder, and a "PlayerController" script inside
- At the top of PlayerController.cs, declare a new
- [SerializeField]
- *private float horizontalInput*
- In *Update()*, set *horizontalInput = Input.GetAxis("Horizontal")*, then test to make sure it works in the inspector
-

```
using UnityEngine;

public class Movement : MonoBehaviour
{
    [SerializeField]
    private float horizontalInput;
    // Start is called before the first frame update
    void Start()
    {


    }


    // Update is called once per frame
    void Update()
    {
        horizontalInput = Input.GetAxis("Horizontal");
```

```
    }
  }
```

4.Move the player left-to-right

- here we can get our input with this basic code, we use [SerializeField] for sanitization and allow us to see the value in the inspection panel to make sure we didn't fck up lol
- Declare a new **public float speed = 10.0f;**
- **here's my code**

```
using UnityEngine;

public class Movement : MonoBehaviour
{
    [SerializeField]
    private float horizontalInput;
    [SerializeField]
    private float speed = 30.0f;
    // Start is called before the first frame update
    void Start()
    {

    }

    // Update is called once per frame
    void Update()
    {
        horizontalInput = Input.GetAxis("Horizontal");
        transform.Translate(Vector3.right * horizontalInput * Time.deltaTime * speed);
    }
}
```

5+6. Keep the player inbounds

- In *Update()*, write an **if-statement** checking if the player's left X position is **less than** a certain value
- In the if-statement, set the player's position to its current position, but with a **fixed X location**
- Repeat this process for the **right side** of the screen
- Declare new *xRange* variable, then replace the hardcoded values with them
- Add **comments** to your code or not lol

```csharp
using UnityEngine;

public class Movement : MonoBehaviour
{
    [SerializeField]
    private float horizontalInput;
    [SerializeField]
    private float speed = 10.0f;
    [SerializeField]
    private float xRange = 20.0f;
    // Start is called before the first frame update
    void Start()
    {

    }

    // Update is called once per frame
    void Update()
    {
        if (transform.position.x < -xRange)
        {
            transform.position = new Vector3(-xRange, transform.position.y , transform.position.z);
        }
        if (transform.position.x > xRange)
        {
            transform.position = new Vector3(xRange, transform.position.y, transform.position.z);
        }
        horizontalInput = Input.GetAxis("Horizontal");
        transform.Translate(Vector3.right * horizontalInput * Time.deltaTime * speed);
    }
}
```

New Functionality

The player can move left and right based on the user's left and right key presses

The player will not be able to leave the play area on either side
New Concepts & Skills
Adjust object scale
If-statements
Greater/Less than operators
Next Lesson, We'll learn how to create and throw endless amounts of food to feed our animals!

now use chat gpt to teach me thing and fix a bug that can occur when you move fast

float clampedX = Mathf.Clamp(desiredpoz.x, -Xrange, Xrange) become the min/max

```csharp
using UnityEngine;
//needed a bit of editing coz cgpt3.5 is stooooooooopid
public class Movement : MonoBehaviour
{
    [SerializeField]
    private float speed = 10.0f;
    [SerializeField]
    private float xRange = 10.0f;
    [SerializeField]
    private float horizontalInput;
    // Start is called before the first frame update
    void Start()
    {

    }


    // Update is called once per frame
    void Update()
    {
        horizontalInput = Input.GetAxis("Horizontal");


        // Calculate the desired position based on the input
        Vector3 desiredPosition = transform.position + Vector3.right * -horizontalInput * Time.deltaTime * speed;


        // Clamp the desired position within the x range
        float clampedX = Mathf.Clamp(desiredPosition.x, -xRange, xRange);
        desiredPosition = new Vector3(clampedX, desiredPosition.y, desiredPosition.z);
```

```
        // Move the object to the clamped position
        transform.position = desiredPosition;
    }
}
```

3. Instantiation

The first thing we must do is give the projectile some forward movement so it can zip across the scene when it's launched by the player.

1.Make the projectile fly forwards

- Create a new "MoveForward" script, **attach** it to the food object, then open it
- Declare a new **public float speed** variable**;**
- In **Update()**, add **transform.Translate(Vector3.forward * Time.deltaTime * speed);** , then **save**
- In the **Inspector**, set the projectile's **speed** variable, then test

```
using UnityEngine;


public class Pizza_gun : MonoBehaviour
{
    public GameObject Ammo;


    public float Weapon_Bullet_Projectile_Speed = 60.0f;
    // Start is called before the first frame update
    void Start()
    {
        Ammo = GetComponent<GameObject>();
    }


    // Update is called once per frame
    void Update()
    {
        transform.Translate(Vector3.forward * Time.deltaTime * Weapon_Bullet_Projectile_Speed);
    }
}
```

- Create a new "Prefabs" folder, drag your food into it, and choose **Original Prefab**
- In PlayerController.cs, declare a new **public GameObject projectilePrefab;** variable

- **Select** the Player in the hierarchy, then **drag** the object from your Prefabs folder onto the new **Projectile Prefab box** in the inspector
- Try **dragging** the projectile into the scene at runtime to make sure they fly
- I added the ammo = get component thing for maybe powerups
- In PlayerController.cs, in **Update()**, add an **if-statement** checking for a spacebar press: **if (Input.GetKeyDown(KeyCode.Space)) {}**
- Inside the if-statement, add a comment saying that you should **// Launch a projectile from the player**
- Inside the if-statement, use the **Instantiate** method to spawn a projectile at the player's location with the prefab's rotation
- **Select** all three animals in the hierarchy and *Add Component* > **Move Forward**Rotate all animals on the Y axis by **180 degrees** to face down
- Edit their **speed values** and **test** to see how it looks
- Drag all three animals into the **Prefabs folder**, choosing "Original Prefab"
- **Test** by dragging prefabs into scene view during gameplay
-

```
using UnityEngine;

public class Movement : MonoBehaviour
{
    [SerializeField]
    private float speed = 10.0f;
    [SerializeField]
    private float xRange = 10.0f;
    [SerializeField]
    private float horizontalInput;
    [SerializeField]
    public GameObject ProjectilePrefab;
    // Start is called before the first frame update
    void Start()
    {

    }

    // Update is called once per frame
    void Update()
    {


        horizontalInput = Input.GetAxis("Horizontal");
        // Calculate the desired position based on the input
        Vector3 desiredPosition = transform.position + Vector3.right * -horizontalInput * Time.deltaTime * speed;
```

```
    // Clamp the desired position within the x range

    float clampedX = Mathf.Clamp(desiredPosition.x, -xRange, xRange);

    desiredPosition = new Vector3(clampedX, desiredPosition.y, desiredPosition.z);

    // Move the object to the clamped position

    transform.position = desiredPosition;

    if (Input.GetKeyDown(KeyCode.Space))

    {

        // shoot pizza

        Instantiate(ProjectilePrefab, transform.position, ProjectilePrefab.transform.rotation);

    }

  }

}
```

5.Make animals into prefabs

- **Rotate** all animals on the Y axis by **180 degrees** to face down/ towards the player
- **Select** all three animals in the hierarchy and *Add Component > **Move Forward***
- Edit their **speed values** and **test** to see how it looks
- Drag all three animals into the **Prefabs folder**, choosing "Original Prefab"
- **Test** by dragging prefabs into scene view during gameplay

6.Destroy projectiles offscreen

- Create "DestroyOutOfBounds" script and apply it to the **projectile**
- Add a new ***private float topBound*** variable and initialize it ***= 30;***
- Write code to destroy if out of top bounds ***if (transform.position.z > topBound) { Destroy(gameObject); }***
- In the Inspector **Overrides** drop-down, click **Apply all** to apply it to prefab

7. Destroy animals offscreen

- Create **else-if statement** to check if objects are beneath ***lowerBound***: ***else if (transform.position.z < lowerBound)***
- **Apply** the script to all of the animals, then **Override** the prefabs
-

8.Lesson Recap

New Functionality

- The player can press the Spacebar to launch a projectile prefab,

Projectile and Animals are removed from the scene if they leave the screen

New Concepts & Skills

- Create Prefabs
- Override Prefabs
- Test for Key presses
- Instantiate objects
- Destroy objects
- Else-if statements

Next Lesson

- Instead of dropping all these animal prefabs onto the scene, we'll create a herd of animals roaming the plain!

(

Lesson 2.2 - Food Flight
Skills practiced:
Absolute Beginner Code Comprehension
Interpret simple code
Improve simple code using the features of an IDE
Absolute Beginner Application Scripting
Use common logic structures to control the execution of code.
Write code that utilizes the various Unity APIs
Implement appropriate data types
Write code that integrates into an existing system
Implement a code style that is efficient and easy to read
Prototype new concepts
)

Random Animal Stampeed

1. Create a spawn manager

- In the Hierarchy, create an **Empty object** called "SpawnManager"
- Create a new script called "SpawnManager", attach it to the **Spawn Manager**, and open it
- Declare new ***public GameObject[ ] animalPrefabs;***
- In the Inspector, change the **Array size** to match your animal count, then **assign** your animals by **dragging** them from the Project window into the empty slots **Note:** Make sure you drag them from the **Project** window; not the Hierarchy! If you're going to spawn objects, you need to make sure you're using Prefabs, which are stored in the Project window.

2. Spawn an animal if S is pressed

- In ***Update()***, write an if-then statement to **instantiate** a new animal prefab at the top of the screen if **S** is pressed
- Declare a new ***public int animalIndex*** and incorporate it in the **Instantiate** call, then test editing the value in the Inspector

3. Spawn random animals from an array

- In the if-statement checking if S is pressed, generate a random ***int animalIndex*** between 0 and the length of the array
- Remove the global ***animalIndex*** variable, since it is only needed locally in the **if-statement**

4. Randomize the spawn location

- **Replace** the X value for the Vector3 with ***Random.Range(-20, 20)***, then test
- Within the **if-statement**, make a new local ***Vector3 spawnPos*** variable
- At the top of the class, create ***private float*** variables for ***spawnRangeX*** and ***spawnPosZ***

5. Change the perspective of the camera

- Toggle between **Perspective** and **Isometric** view in the Scene view to appreciate the difference
- Select the **camera** and change the **Projection** from "Perspective" to "Orthographic"

6. Lesson Recap

New Functionality

- The player can press the S to spawn an animal
- Animal selection and spawn location are randomized
- Camera projection (perspective/orthographic) selected

New Concepts & Skills

- Spawn Manager
- Arrays
- Keycodes
- Random generation
- Local vs Global variables
- Perspective vs Isometric projections

Next Lesson

- Using collisions to feed our animals!

Collision Decisions + GAME OVER

1. Make a new method to spawn animals

- In **SpawnManager.cs**, create a new **_void SpawnRandomAnimal() {}_** function beneath **_Update()_**
- Cut and paste the code from the **if-then statement** to the **new function**
- Call **_SpawnRandomAnimal();_** if **S** is pressed

2. Spawn the animals at timed intervals

- In **_Start()_**, use **_InvokeRepeating_** to spawn the animals based on an interval, then **test**.
- Remove the **if-then statement** that tests for **S** being pressed
- Declare new **_private startDelay_** and **_spawnInterval_** variables then playtest and tweak variable values

3. Add collider and trigger components

- Double-click on one of the **animal prefabs**, then _Add Component > Box Collider_
- Click **Edit Collider**, then **drag** the collider handles to encompass the object

- Check the "**Is Trigger**" checkbox
- Repeat this process for each of the **animals** and the **projectile**
- Add a **RigidBody component** to the projectile and uncheck "use gravity"

4. Destroy objects on collision

- Create a new **DetectCollisions.cs** script, add it to each animal prefab, then **open** it
- Before the final **}** add an **OnTriggerEnter** function using **autocomplete**
- In **OnTriggerEnter**, put **Destroy(gameObject);**, then test
- In **OnTriggerEnter**, put **Destroy(other.gameObject);**

5. Trigger a "Game Over" message

- In DestroyOutOfBounds.cs, in the **else-if condition** that checks if the animals reach the bottom of the screen, add a Game Over messsage: **Debug.Log("Game Over!")**
- Clean up your code with **comments**
- If using Visual Studio, Click *Edit > Advanced >* **Format document** to fix any indentation issues (On a **Mac**, click *Edit > Format > Format Document)*

6. Lesson Recap

New Functionality
- Animals spawn on a timed interval and walk down the screen
- When animals get past the player, it triggers a "Game Over" message
- If a projectile collides with an animal, both objects are removed

New Concepts & Skills
- Create custom methods/functions
- InvokeRepeating() to repeat code
- Colliders and Triggers
- Override functions
- Log Debug messages to console

Pizza_gun

```
using UnityEngine;
```

```csharp
public class Pizza_gun : MonoBehaviour
{
    [SerializeField]
    public float Weapon_Bullet_Projectile_Speed = 60.0f;
    [SerializeField]
    private float topBound = -60;


    // Update is called once per frame
    void Update()
    {
        transform.Translate(Vector3.forward * Time.deltaTime * Weapon_Bullet_Projectile_Speed);


        if (transform.position.z < topBound)
        {
            Destroy(gameObject);
        }


    }
    private void OnTriggerEnter(Collider other)
    {
        Destroy(gameObject);
        Destroy(other.gameObject);
    }
}
```

Spawn_manager

```csharp
using System.Collections;
using System.Collections.Generic;
using UnityEngine;


public class SpawnManager : MonoBehaviour
{
    public GameObject[] animalPrefabs = new GameObject[10];
    [SerializeField]
    private float spawnRangeX = 20.0f;
    [SerializeField]
    private float spawnPozY = -20.0f;
    [SerializeField]
    private float startDelay = 5f;
    [SerializeField]
```

```csharp
    private float spawnInterval = 2f;
    // Start is called before the first frame update
    void Start()
    {
        // invoke repeating, will make the method or void def function with name "nameInQuotes"
        // it will call that function after , delay, then chozen interval
        InvokeRepeating("SpawnRandomAnimal", startDelay, spawnInterval);
    }

    // Update is called once per frame
    void Update()
    {
        if (Input.GetKeyDown(KeyCode.S))
        {
            SpawnRandomAnimal();
        }
    }
    void SpawnRandomAnimal()
    {
        // Select a random index within the array range
        int random_animal_Index = Random.Range(0, animalPrefabs.Length);
        Vector3 spawnPos = new Vector3(Random.Range(-spawnRangeX, spawnRangeX), spawnPozY, -30);
        // Instantiate the selected animalPrefab
        Instantiate(animalPrefabs[random_animal_Index], spawnPos,
            animalPrefabs[random_animal_Index].transform.rotation);
    }
}
```

- movement

```csharp
using UnityEngine;

public class Movement : MonoBehaviour
{
    [SerializeField]
    private float speed = 10.0f;
    [SerializeField]
    private float xRange = 10.0f;
    [SerializeField]
```

```
    private float horizontalInput;
    [SerializeField]
    public GameObject ProjectilePrefab;
    // Start is called before the first frame update
    void Start()
    {

    }

    // Update is called once per frame
    void Update()
    {


        horizontalInput = Input.GetAxis("Horizontal");
        // Calculate the desired position based on the input
        Vector3 desiredPosition = transform.position + Vector3.right * -horizontalInput * Time.deltaTime
* speed;
        // Clamp the desired position within the x range
        float clampedX = Mathf.Clamp(desiredPosition.x, -xRange, xRange);
        desiredPosition = new Vector3(clampedX, desiredPosition.y, desiredPosition.z);
        // Move the object to the clamped position
        transform.position = desiredPosition;
        if (Input.GetKeyDown(KeyCode.Space))
        {
            // shoot pizza
            Instantiate(ProjectilePrefab, transform.position, ProjectilePrefab.transform.rotation);
        }
    }
}
```

Animal Movement

```
using UnityEngine;

public class AnimalMovement : MonoBehaviour
{


    public float Animal_Movement_Speed = 50.0f;
    public float lowerBound = 50;
```

```csharp
    // Start is called before the first frame update
    void Start()
    {

    }

    // Update is called once per frame
    void Update()
    {
        transform.Translate(Vector3.forward * Time.deltaTime * Animal_Movement_Speed);
        if (transform.position.z > lowerBound)
        {
            Destroy(gameObject);
            Debug.Log("In the vernacular of your people... \n Game Over!");
        }
    }
}
```

-

unity quizes (cheet sheets if fail)

questions i got wrong on all of my unity quizzes

Question 5

If you want to move the character up continuously as the player presses the up arrow, what code would be best in the two blanks below:

Image not found or type unknown

GetKey(KeyCode.UpArrow)

GetKeyDown(UpArrow)

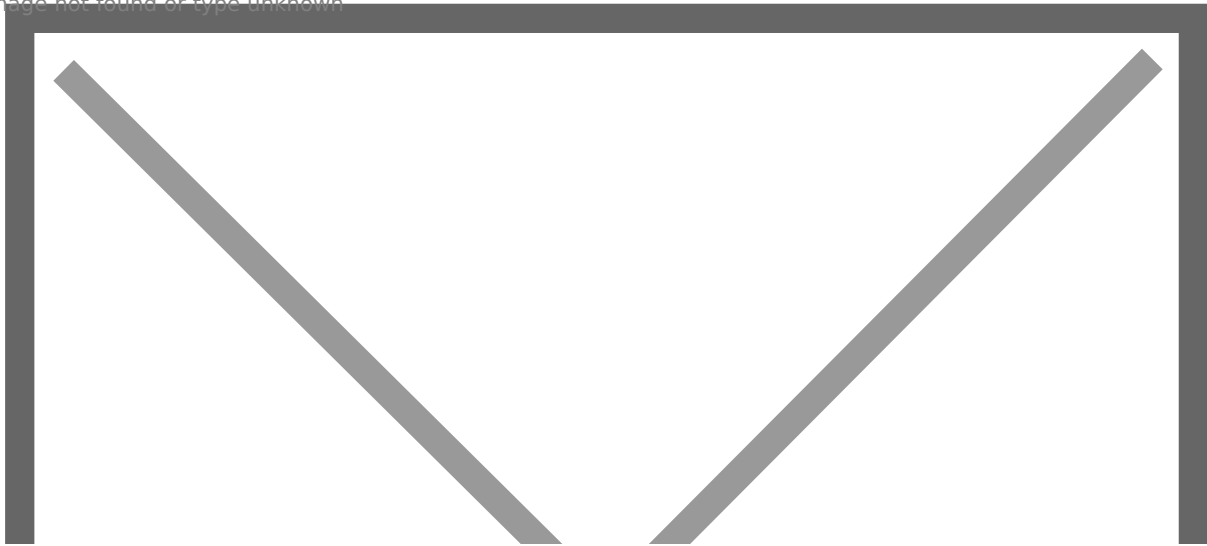GetKeyUp(KeyCode.Up)

GetKeyHeld(Vector3.Up)

Incorrect

"Input.GetKey" tests for the user holding down a key (as opposed to KeyKeyDown, which test for a single press down of a Key).

Question 10

You're trying to create some logic that will tell the user to speed up if they're going too slow or to slow down if they're going too fast. How should you arrange the lines of code below to accomplish that?

4, 6, 1, 2, 5, 9, 7, 8, 3

void Update() { if (speed < 10) {Debug.Log(speedUp); } else if (speed > 60) { Debug.Log(slowDown); } } private float speed; private string slowDown = "Slow down!"; private string speedUp = "Speed up!";

6, 1, 2, 5, 7, 8, 3, 4, 9

if (speed < 10) { Debug.Log(speedUp); } else if (speed > 60) { Debug.Log(slowDown); } private speed; private string slowDown = "Slow down!"; private string speedUp = "Speed up!"; void Up { }

7, 8, 3, 4, 6, 5, 2, 1, 9

private float speed; private string slowDown = "Slow down!"; private string speedUp = "Speed void Update() { if (speed < 10) { Debug.Log(slowDown); } else if (speed > 60) { Debug.Log(speedUp); } }

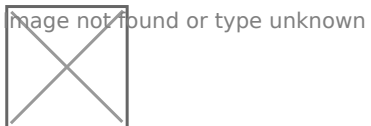7, 8, 3, 4, 6, 1, 2, 5, 9

private float speed; private string slowDown = "Slow down!"; private string speedUp = "Speed void Update() { if (speed < 10) { Debug.Log(speedUp); } else if (speed > 60) { Debug.Log(slowDown); } }

Incorrect

All variables should be declared first, then the void method, then the if-condition telling them to speed up, then the else condition telling them to slow down.

Question 6

Using the information shown in the autocomplete pop-up window below, which of the following would be a valid call of the Instantiate method?

Image not found or type unknown

Select image to expand

```
Instantiate(objectPrefab.transform, spawner, true);
```

```
Instantiate(objectPrefab, spawner, false);
```

```
Instantiate(objectPrefab, spawner.transform, true);
```

```
Instantiate(objectPrefab, spawner, bool);
```

Incorrect

You can see from the tooltip that the Instantiate method can receive 3 parameters with types in the following order: Object, Transform, bool. That means that the first parameter can be an object (objectPrefab) the second can be a transform of an object (spawner.transform) and the third must be a bool value (e.g. true or false).

Lab 2 - New Project with Primitives

1.Create a new Unity Project and rename your scene

"Just like we did with the Prototype, the first thing we need to do is create a new blank project "

1. [Create a new Unity project](#) called "Personal Project" using the **3D template** and setting the location to the "**Create with Code**" folder
2. After Unity opens, select your custom **Layout**
3. In the Project window, *Assets > Scenes*, rename "**SampleScene**" to "My Game"

2.Create a background plane

"To orient yourself in the scene and not feel like you're floating around in mid-air, it's always good to start by adding a background / ground object"

1. In the Hierarchy, *Right-click > 3D Object > **Plane*** to add a plane to your scene
2. In the Plane's Inspector, in the top-right of the Transform component, click on the three dots icon **> *Reset***
    1. **Note**: the three dots will appear as a gear icon in older versions of Unity.
3. Increase the **XYZ scale** of the plane to (5, 1, 5)
4. Adjust your position in Scene view so you have a good view of the Plane

3.Create primitive Player with a new material

"Now that we have the empty plane object set up, we can add the star of the show: the player object"

1. In the Hierarchy, *Right-click > 3D Object > **Sphere,*** then rename it "Player"
2. In Assets, *Right-click > Create > **Folder*** named "Materials"
3. Inside "Materials", *Right-click > Create > **Material*** and rename it "Blue"
4. In Blue's Inspector, click on the **Albedo color** box and change it to a blue
5. **Drag** the material from your Assets onto the Player object

4.Position camera based on project type

"Now that we have the player in there, we need the best view of it, depending on our type of project"

1. For a **top-down** game, position the camera at (0, 10, 0) directly over the player and rotate it 90 degrees on the **X axis**
2. For a **side-view** game, rotate the **Plane** by -90 degrees on the **X axis**
3. For an **third-person** view game, move the camera up on the **Y and Z axes** and increase its **rotation on the X axis**

5.Enemies, obstacles, projectiles & materials

"*Now that we know how to make primitives, let's go ahead and make one for each object in our project*"

1. In the Hierarchy, create new **Cubes**, **Spheres**, and **Capsules** for all other main objects, **renaming** them, **repositioning** them, and **scaling** them
2. In your Materials folder, create **new materials** for as many colors as you have unique objects, editing their color to match their name, then **apply** those materials to your objects
3. Position all of your objects in locations relative to each other that make sense

6.Export a Unity Package backup file

"*Since we're going to be putting our hearts and souls into this project, it's always good to make backups*"

1. **Save** your Scene
2. In the Project window, Right-click on the *"Assets" folder* > **Export Package**, then click Export
3. Create a **new "Backups" folder** in your Personal Project folder, then **save** it with your name and the version number (e.g. Carl_V0.1.unitypackage")

7.Lesson Recap

New progress

- New project for your Personal Project
- Camera positioned and rotated based on project type
- All key objects in scene with unique materials

New concepts & skills:

- Primitives
- Create new materials
- Export Unity packages

Mod the Cube

Summary : In this exercise, you'll be challenged to modify a provided script to change the behavior of an object it's applied to.

Type:Tutorial

Level:Beginner

Xp: 60
Estimated time to complete: 30 Mins

Materials: ModTheCubeChallengeAssets.zip

1.Overview

In this mission you learned how to open, read, and modify scripts. You practiced fundamental programming challenges, and hopefully you're starting to feel comfortable with basic scripting. In this exercise, you'll be tasked with modifying the behavior of an already existing script. We're leaving this challenge open-ended, so now's the time to get creative!

2.Download the provided assets and import them into a new Unity Project

1. Download ModTheCubeChallengeAssets.zip.
2. To unzip the contents of the folder, right-click the folder and select **Extract All** (macOS: double-click the zipped folder). Inside you will find ModTheCube.unitypackage.

   Note: While it is possible to view the contents of a folder while it's still zipped, the data is still compressed. Attempting to import the contents of the ModThCubeChallengeAssets folder while it's still zipped will result in an error message in Unity. Ensure that you have unzipped the folder before proceeding to the next step.
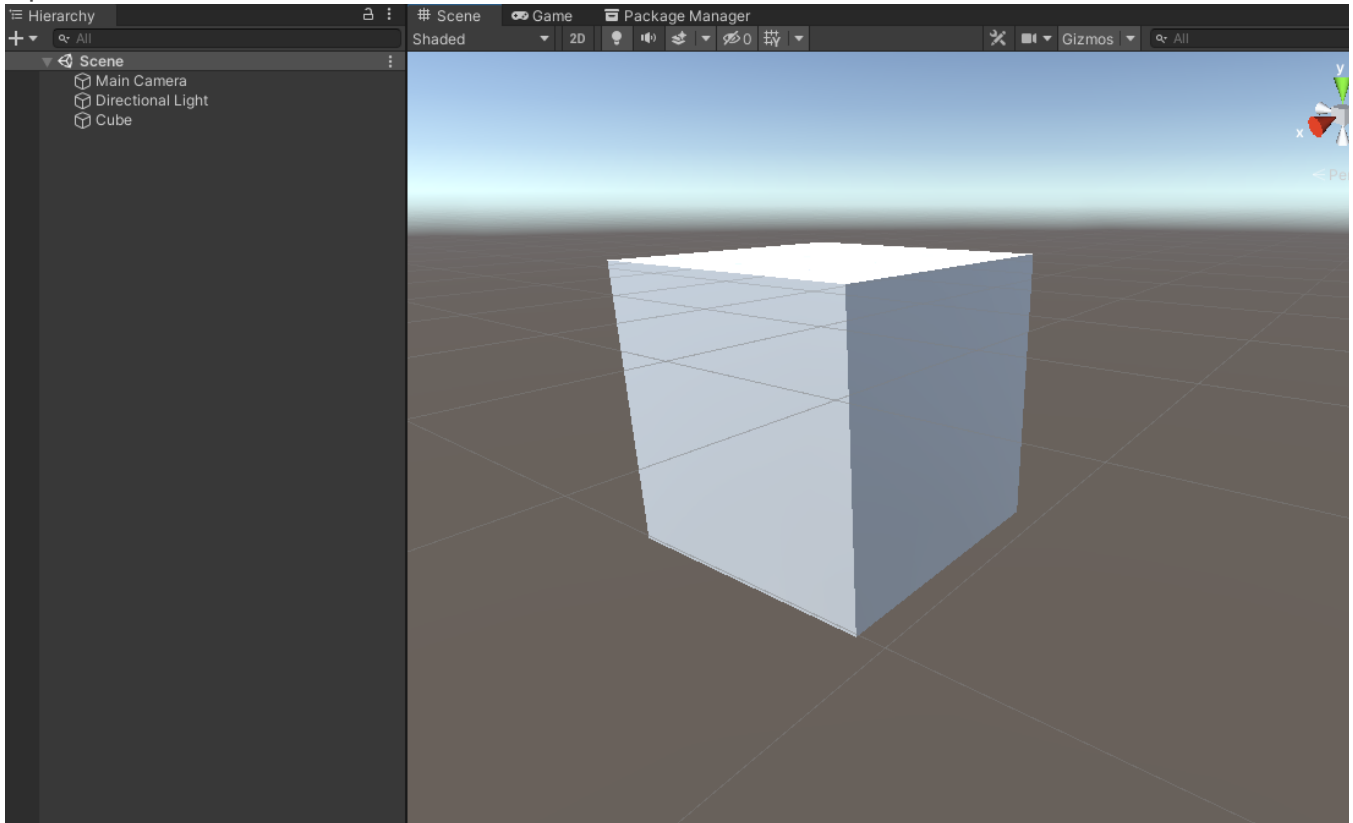
3. Create a new Unity project using the 3D template - remember to use one of the supported versions of Unity (2018.4LTS, 2019.4LTS, or 2020.3LTS).

4. [Import](#) ModTheCube.unitypackage into your new 3D template project.

3.Observe the default script functionality

Before you begin editing the code, take a few minutes to understand what the script does by default. The imported .unitypackage contains three items: a C# script, a material, and a scene.

1. Open the scene located in the ModTheCube folder.

The only thing in the scene aside from the Camera and Directional Light is a cube. Attached to the cube is the script you will be modifying.

2. Enter Play mode (select the play button on the toolbar) and observe the default behavior of the script assigned to the cube.
When you enter Play mode, you will observe several of the cube's properties change. All of these changes are hardcoded into the cube.cs script attached to the cube in the scene. This means that the cube will be modified in the same way each time the scene is run.

4.Modify the cube.cs script

Now that you understand what the cube script is doing, you can begin to customize it. As previously mentioned, all of the changes the script is making to the cube are specified in the code. It would be easier to control the cube from the Unity Editor instead of editing the script. You can make this possible by changing variables to be public. Public variables appear in the Inspector window where you can modify their values.

1. Open the cube.cs script in your code editor and observe the code.
2. Identify at least two changes that you can make to the cube's behavior.
   - Change the cube's location (transform).
   - Change the cube's scale.
   - Change the angle at which the cube rotates.
   - Change the cube's rotation speed.
   - Change the cube's material color.
   - Change the cube's material opacity.If you're feeling confident, you can attempt the following as well:
   - Modify any of the changes above so they change randomly each time the scene is played.
   - Add extra functionality to the cube. For example, how might you change the color of the cube over time?
3. Implement your planned changes in the code.
4. Test the scene regularly as you work to ensure your code is performing as expected.

5.Create a WebGL project showing off your modded cube

Now that you have edited your script and thoroughly tested your changes, your remaining task is to submit your project here and show off your work! You will create a WebGL build of your project, upload it to Unity Play, and then complete this challenge by linking your submission below.

1. If you haven't already done so, frame your camera in the scene so it's easy to see the cube in the Play view.
2. Save your scene.
3. Ensure that your version of Unity has WebGL installed. Not sure if it does, or don't know how to add it? Refer to step 2 of the [Create and publish WebGL Builds](#) tutorial.
4. Install the WebGL Publisher package. This package will allow you to easily publish your project on Unity Play. If you aren't sure if you have the package or don't know how to add it, refer to the tutorial linked in the step above.
5. Create a WebGL build of your project. This may take a few minutes, depending on your computer. If you need help creating the build, refer to step 4 of [Create and publish WebGL builds](#).
6. Publish your WebGL build on Unity Play. If you need help publishing on Unity Play, refer to step 6 of [Create and publish WebGL builds](#).
7. Copy your Unity Play link, and paste it below!

6.Criteria

You'll know you're successful if you've met these criteria:

- You've modified the cube script so that at least two changes are made to the cube's behavior.
- You've created a WebGL build of your project.
- You've uploaded the WebGL build to Unity Play.

- You've copied your Unity Play link for submission.

7.Next steps

Congratulations on completing the first mission in Junior Programmer! You now have a strong foundational understanding of the structure of basic scripts and how to make simple modifications. As you continue through the pathway, consider taking time to think about how you might modify other scripts you encounter to customize them to your liking. Sometimes the best way to reinforce what you've learned is to apply it in different ways on your own!

8.Submission

Paste your Unity Play link below!

[link to the original unity course](#)