DT2140 Multimodal Interaction - Lab 1 Webcam

Updated 2024-10-03 Livia Qian

Updated 2023-10-30 Anna Deichler

Updated 2022-11-18 Tania Amel Sayyah

Updated 2022-11-07 Tania Amel Sayyah

Updated 2022-10-27 Tania Amel Sayyah

Updated 2022-10-26 Tania Amel Sayyah

Updated 2022-10-25 Tania Amel Sayyah

Updated 2021-11-11 by Ioana Cotutiu

Updated 2021-10-29 by Ioana Cotutiu

Updated 2021-10-11 by Ioana Cotutiu

Updated 2021-10-08 by Ioana Cotutiu

Updated 2021-10-07 by Ioana Cotutiu

Updated 2021-10-06 by Ioana Cotutiu

Updated 2020-11-11 by Caroline Borg

Updated 2020-11-10 by Caroline Borg

Updated 2020-11-09 by Caroline Borg

Updated 2020-10-25 by Caroline Borg

Updated 2020-10-24 by Caroline Borg

Updated 2019-11-11 by Alex Wennberg

Updated 2018-11-02 by Alex Wennberg

Updated 2018-10-10 by Alex Wennberg

Dear student,

We will check this document **before the lab session starts**. Feel free to skim through beforehand but be aware that we might make some changes (to help you succeed).

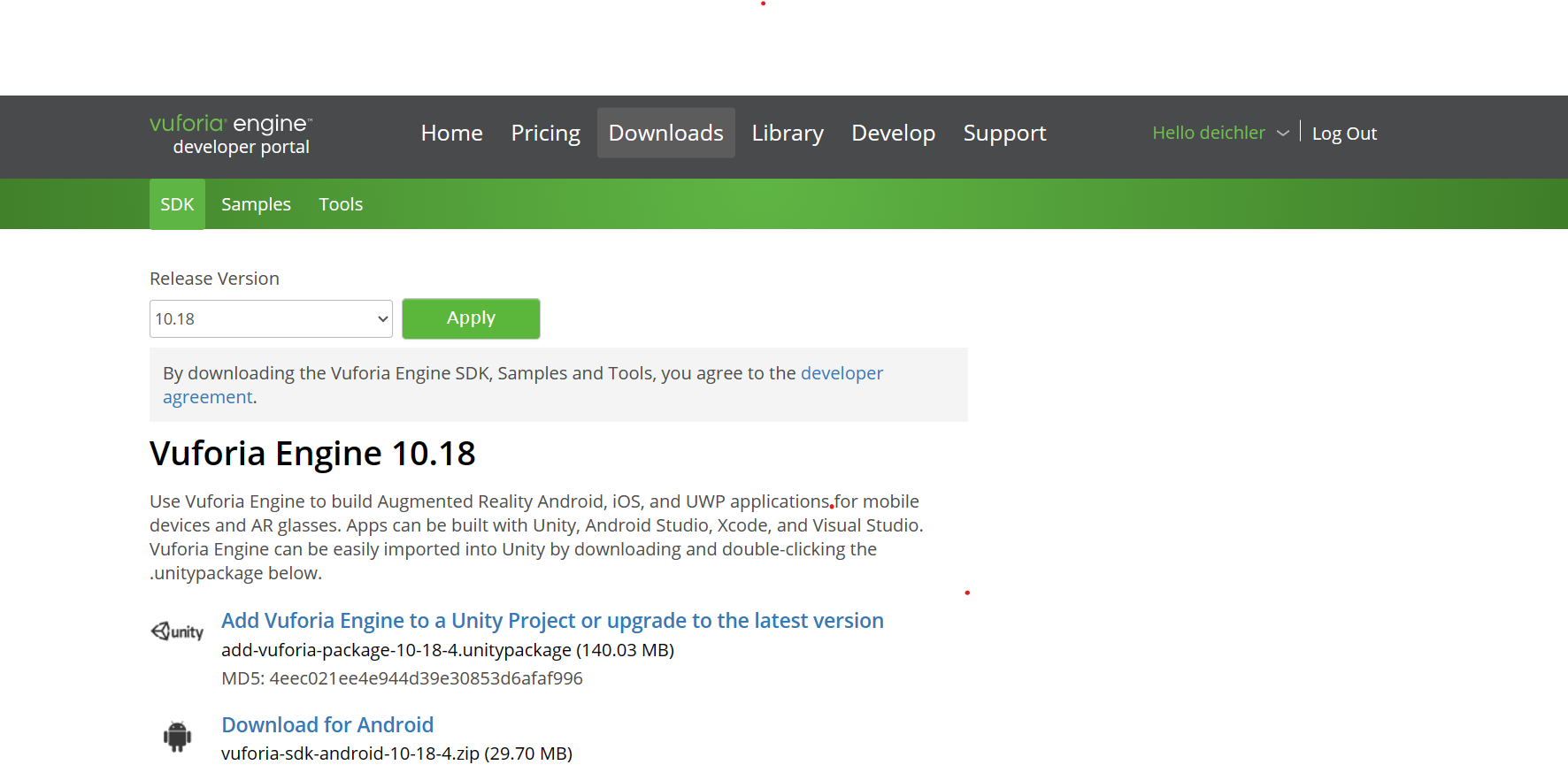
Introduction - Preparatory work

In this lab, you will set up a development environment and create a simple application that utilizes the Vuforia Augmented Reality engine in Unity. The instructions below have been confirmed to work with **Unity 2022.3.49f1 (latest tested)** and **Vuforia Engine 10.25**.

Be aware that your **group** needs a **laptop with a webcam** for this lab, so try to group up so that every group has at least one. If you choose **Android**, check the instructions in the [other document](https://docs.google.com/document/d/1KMnnrT9URrgUm8tZO6oMutXrpNh3DAwnkRzga7n9kS4/edit?usp=sharing).

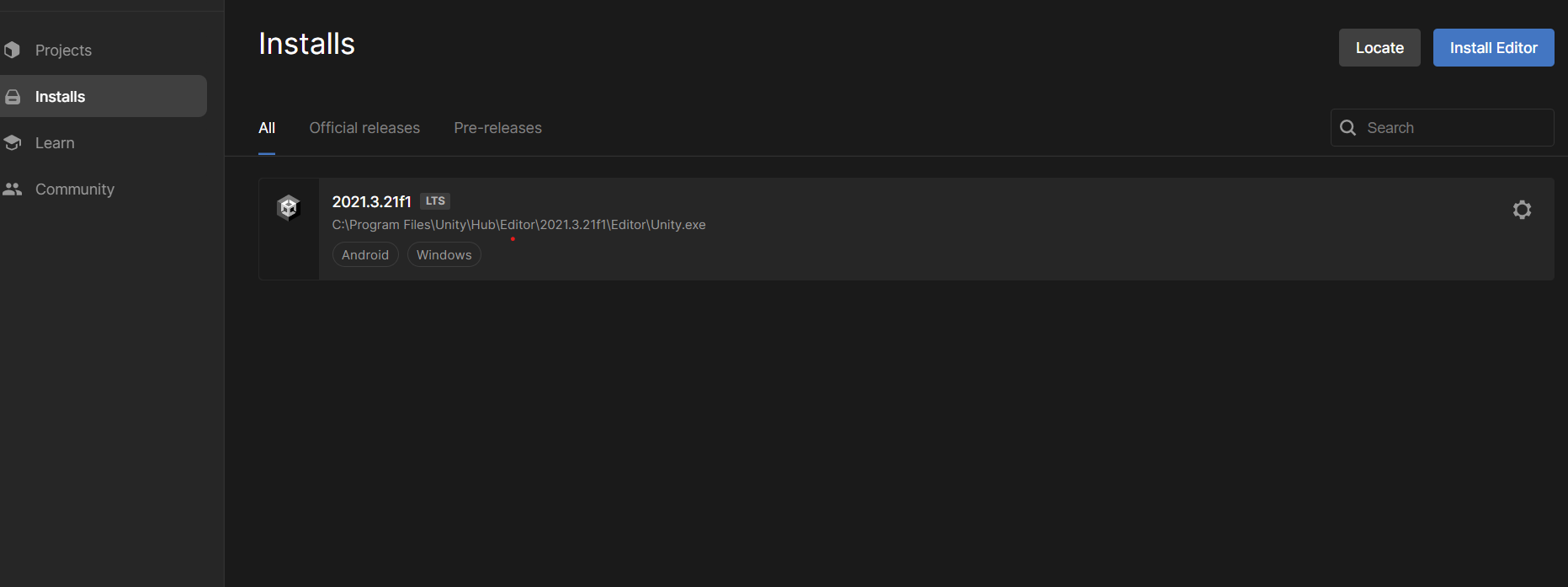
You have to have one of [these](https://library.vuforia.com/platform-support/supported-versions) operating systemson the computer that you will use for development (preferably Windows 10 or 11). Lastly, you’ll **need to register an account** [with Vuforia](https://library.vuforia.com/articles/Solution/How-To-Register-as-a-Vuforia-Developer.html).

The next thing is to download the Vuforia SDK. You can just download the first package (*add-vuforia-package-10-18-4.unitypackage*). Add it to a designated working folder which you will use later in the lab to store everything.

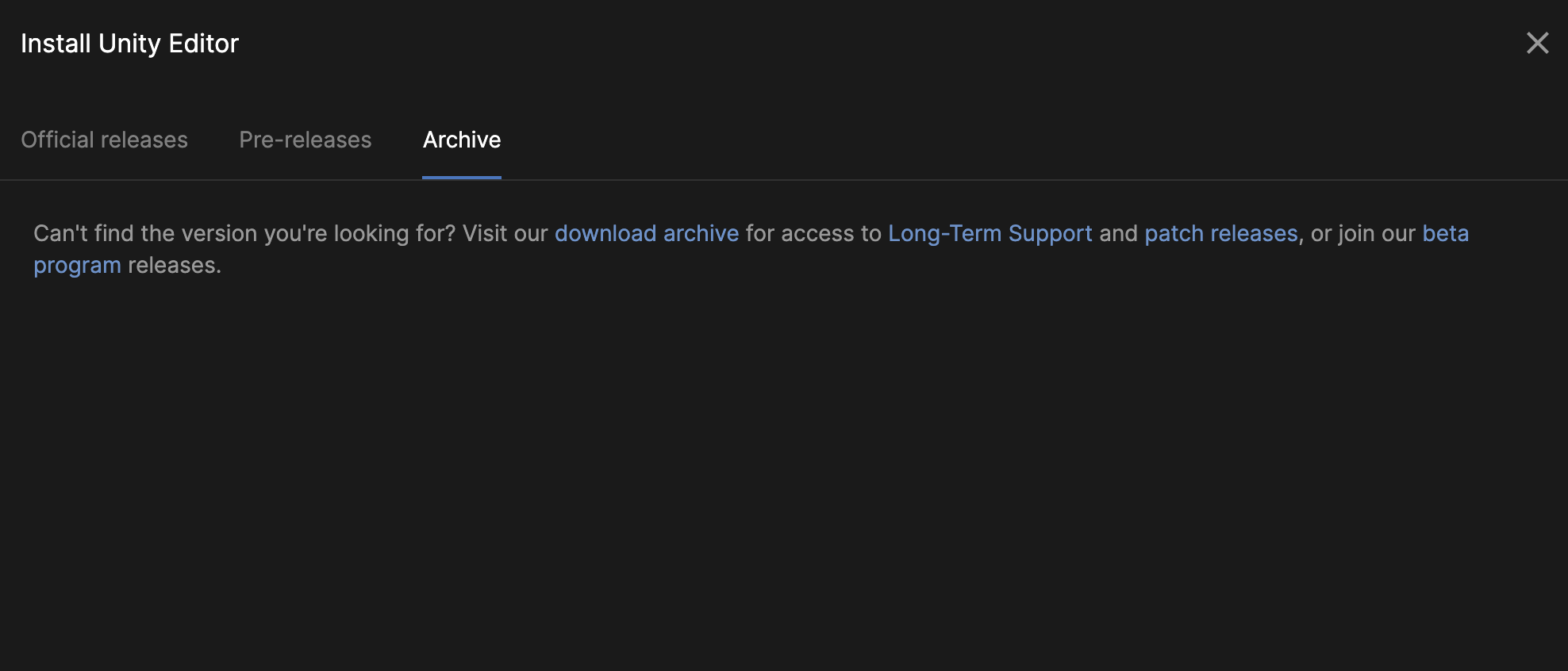


Unity installation:

1. First, download Unity Hub [here!](https://unity3d.com/get-unity/download)
2. Install Unity editor in Unity Hub as seen below when clicking “Install editor” at the top right corner.



1. Look for the newest Unity version, alternatively for **Unity 2020.3.20f1** in **Archive** when pressing **“download archive”.**  This takes you to the **Unity website**, find the correct version and open in Unity Hub, which will start the installation.



1. Once you’re ready to start, start Unity.

Part 0 – Introduction to Unity

Since we will use the Unity engine for this lab, this part contains a short introduction to this game engine. If you are already familiar with it, feel free to skip to **Part 1**.

**Unity** is a cross-platform game engine that provides a set of predefined components and is mainly used to create 3D and 2D virtual environments for games and other interactive applications with similar needs. In this lab, we are going to work with the following:

**Scenes** are mainly the working space in which you arrange various objects that make up your application. Larger applications often split complex content into independent scenes which are then loaded one at a time. The Unity editor displays your currently loaded scene with all its components in the center of the screen. Using a set of tools (hand tools, rotate tool, scale tool, and more), you can move around the scene and manipulate its content. Here you will use a single scene.

**Game Objects** are the fundamental entities of the Unity engine that represent characters and scenery. Each object has a position, a scale, and a rotation within the Scene. Essentially, Game Objects act as containers for various Components, which implement actual functionalities.

**Components** are a vast collection of different functional attributes (functionalities) that can be attached to Game Objects. Most functionalities in Unity, such as 3D mesh rendering, collision detection, and physics simulations, are implemented by adding/attaching predefined components to different Game Objects in your scene.

**Scripts** are an essential component of most Unity applications. Scripts are snippets of C# code that are mainly used to capture and respond to user input and to control and schedule events in the gameplay. In addition to this, scripts can be used to control the object's physical or generate graphical effects. For example, using scripts you can define and associate variables and functions to each instance of a GameObject. Each function can be called at specific times (when a Collider Component registers a collision) or between frame updates.

**Project Settings** are a collection of global variables that have various impacts on the project, but that will not be an interactive component of the application after compilation.

Part 1 – Hello World AR

Your first task is to create a simple app that will recognize a target image and show a virtual 3d model next to it.

**Start with a new project.**

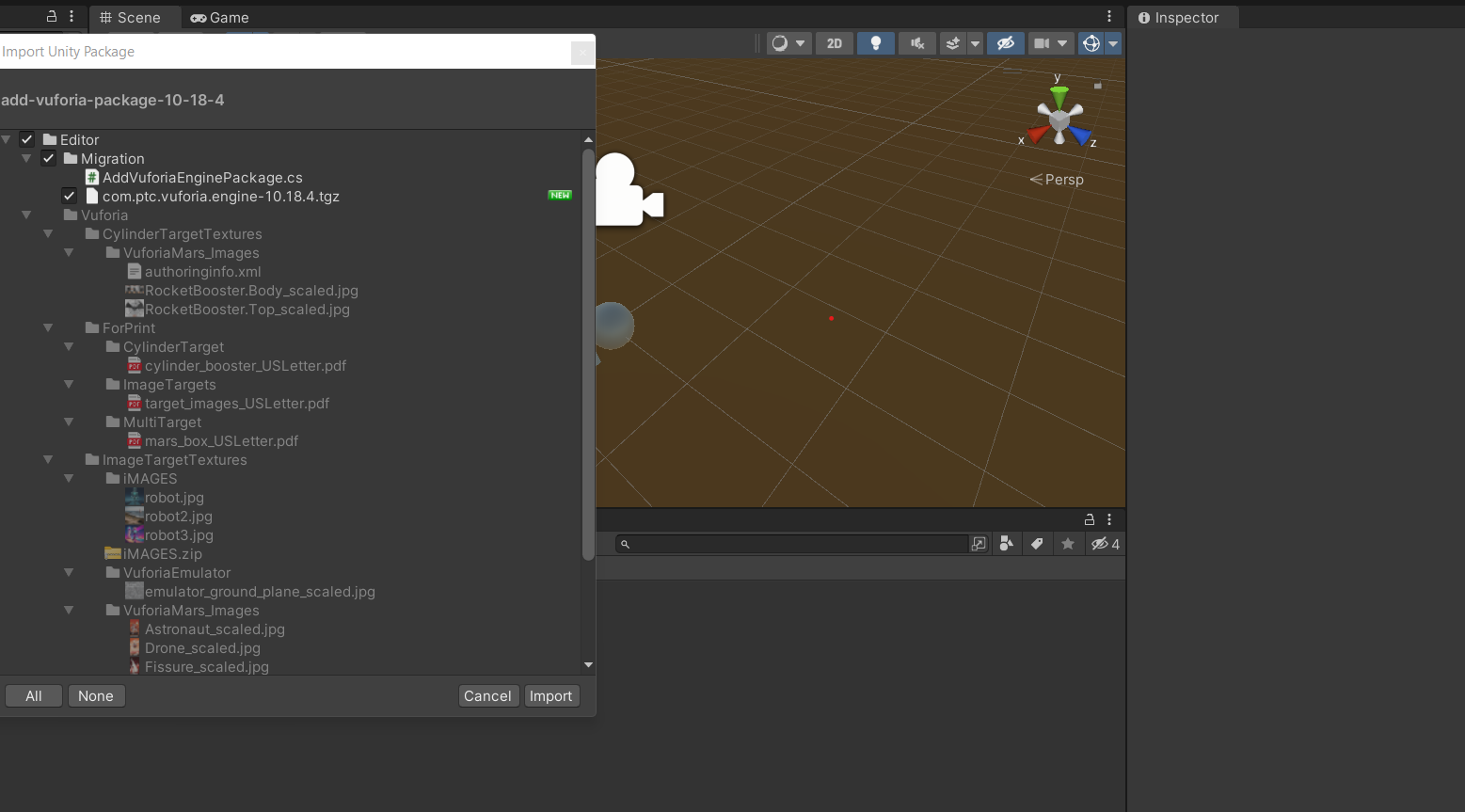
Create a **new project in 3D** and give the path to the working folder where you have the Vuforia SDK.

1. **Import the Vuforia Engine**

Now let’s import our Vuforia package! Double click on the downloaded file, “*add-vuforia-package-..*.”. This will open the import package window (lists your Unity package which should all be checked). Click on “Import”.

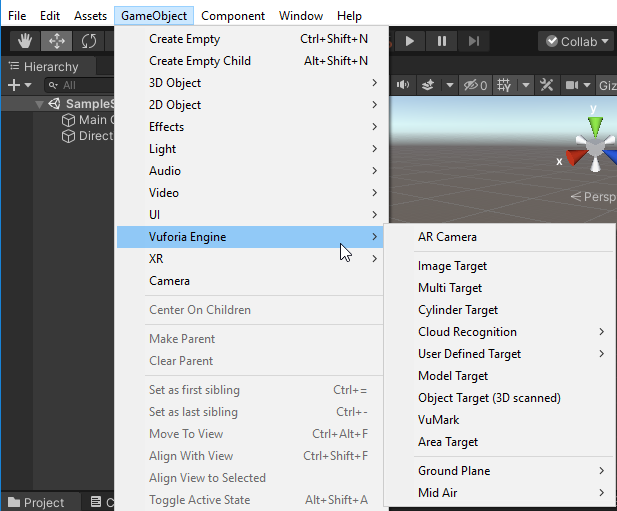
Alternatively:

**Assets → Import Package → Custom Package…**, Find your “*add-vuforia-package-...*” downloaded from [here](https://developer.vuforia.com/downloads/sdk) and selected “Open”. You should see the same pop-up window.



Now after this, you might get another Unity popup window that says “Add Vuforia Engine Package”. Make sure to select “Update”.

You should now be able to find the Vuforia Engine in the GameObject drop-down menu:



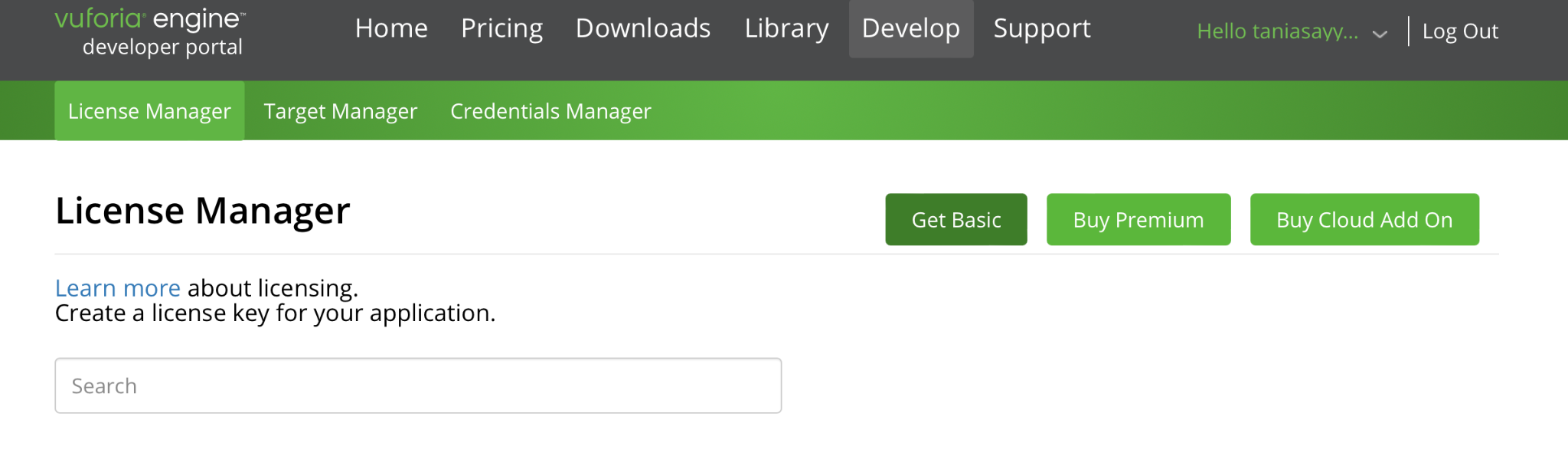
**If you don’t, try building the project and check again!**

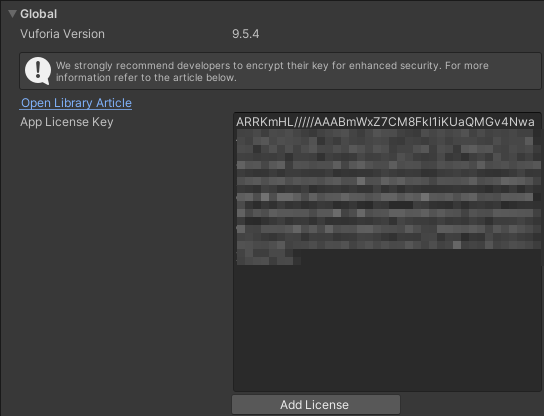
1. **Register your copy of Vuforia**

Go to Vuforia Configurations found at **Window → Vuforia Configuration** and submit your [App License Key](https://developer.vuforia.com/vui/develop/licenses). If you click on the Add License button, you should be redirected to the correct website.

Unity might first ask you to accept the Vuforia Developer Agreement from **Help → Vuforia Engine → Show Developer Agreement**.

Create a license key by clicking “Get Basic” and enter a short name. When created, pressing the license will give the license key to copy into Unity.





1. **Replace the old camera with a Vuforia camera (AR Camera)**

The first step you need to take in the actual scene is to remove the old camera (Main Camera) and import the AR camera into your application.

You can remove the **Main Camera** object by either:

* clicking the entry in the **Hierarchy** view on the left-hand side of the Unity window, and pressing the **Delete** key
* or clicking the camera in the scene and pressing the **Delete** key

To create a new **AR Camera** object through the drop-down menu:

**Game Object → Vuforia Engine → AR Camera**

There should now be one camera called AR Camera and make sure it’s on the top above the other layers. While on the AR Camera layer (click on the object or object name), adjust the position in the **Inspector** panel  **→ Transform**. The easiest way is to put the **Positions X, Z to 0,** and **Y to any above 0**. Change **X Rotation to 90**, so the camera will angle down. You don’t want the camera too close to the Image, so you can have space for your 3D objects.

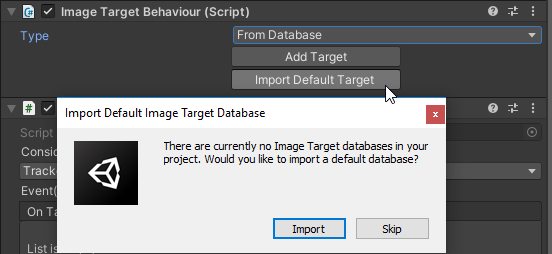
1. **Add an image marker to the scene**

Now that we have an AR Camera in the scene, add an image object to your scene in the same way:

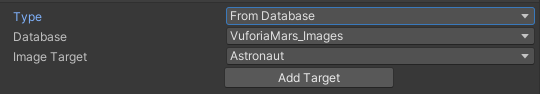
**Game Object → Vuforia Engine → Image Target**

You don’t need to move the image or scale it in any way in the Unity scene.

For now, you can use the default database that Vuforia provides:



After importing the default database, the **Image Target Behavior** should look like this:



[We are using these images here.](https://library.vuforia.com/sites/default/files/vuforia-library/docs/target_samples/unity/mars_target_images.pdf)

1. **Add a 3D model to augment the marker**

When Vuforia recognizes the **Image Target** created in the previous step it will show any children of that Game Object, and hide them otherwise.

To do this you need to add a **3D Object** of your choice to the scene.:

**Game Object → 3D Object**

If you don’t see the object it might be because your scene camera is placed inside of it. You will most likely need to **scale down the recently created 3D Object**.

To do this, **select the 3D Object in the scene or in the hierarchy tab and change the default Scale variables (Inspector → Transform → Scale).**

I’ve found that **(0.5, 0.5, 0.5)** is a suitable size for most of the standard objects. This might have to change if the image is larger or smaller. See that the object has the same position as the image.

The 3D object will be displayed in size and position relative to the image.

To declare the **3D Object** as a child of the **Image Target** you click and drag the entry for your 3D Object onto the Image Target in the hierarchy list on the left-hand side of the Unity interface.

1. **Compile and run!**

Click on the Play button on the top middle of the Unity Program to run the scene. You can click it again to go back to this scene.

On your phone or any other surface visible to the camera, open the picture you chose as Image Target from the [sample database](https://library.vuforia.com/sites/default/files/vuforia-library/docs/target_samples/unity/mars_target_images.pdf). Try aiming the camera at the picture.

Can you see your chosen Game Object floating in front of it? If not, try not sitting behind direct light and it should work after a while.

**Potential issues:**

* If you can’t get it to project your Game Object, try another picture.
* See if the whole image is visible in the scene when clicking the AR camera. If not, play around with the positions so that it is visible and try again.
* Change the scale of the Game Object.
* Go back to Vuforia Configurations and see if the Licence key is still there!
* Save the project (Ctrl+S).
* Try someone else’s computer too.
* Other problems? Ask for help!

Part 2 – Setting up a custom image database

Now that you have a functioning yet barebone AR application, the next step for you is to find or create your own set of images for your application to use as markers. For Vuforia to recognize these images you need to generate your own database for the engine to use.

1. **Finding appropriate images**

Search for a set of images you want to use for your own targets: about 2-4 should be enough. The images need to be in **.jpg** or **.png** file formats and **less than 2MB** in size. *Make sure you’re allowed (have the right) to use them for this lab.* Try to find a variety of images! Different complexity levels and contrasts can often impact how easily an image can be recognized by the Vuforia engine.

1. **(Optional) Print your images**

Print your images so you have tangible targets for testing. It is possible to display the images you want to use on your phone screen, but you will not be able to, for example, fold the image to see how image distortion impacts tracking quality.

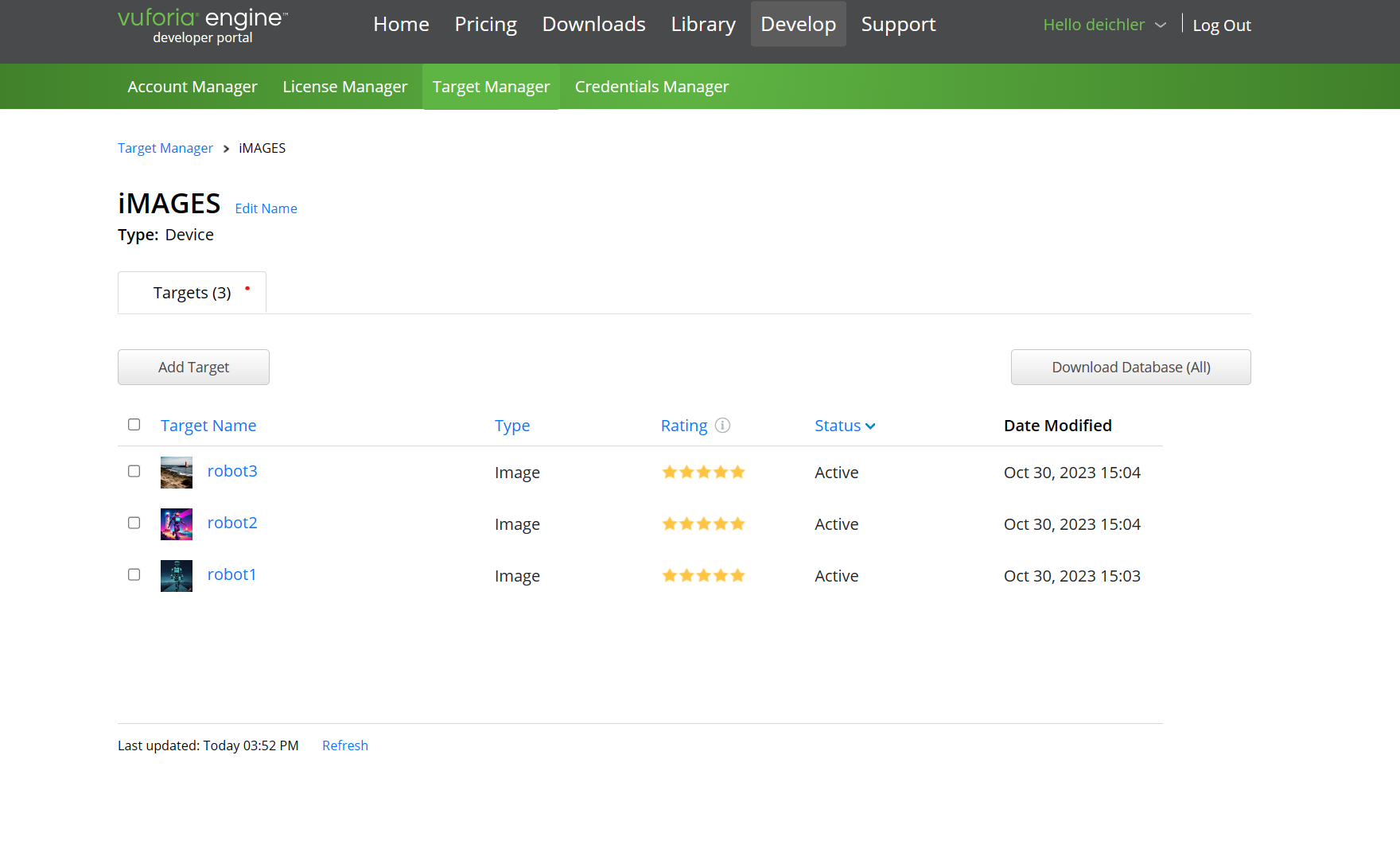
1. **Create a database and add it to your project**

For Vuforia to recognize your new targets you need to generate a database through Vuforia’s developer portal and add the images. To do this:

In Vuforia go to the **Develop > Target Manager > add database.**

Click on your new database then **“Add Target”** on the next screen.

Select **“Single Image”**. In **“Width”** let’s just keep the default to “1”. In the database you’ll see your image attached below along with a rating. Sometimes it might take a second for the “rating” to appear or you’ll have to click on it. If you have 4 or fewer stars, this means your image has a lower probability of the image being detected by the camera. Aim to have at least 4 stars on each image.



You should see something like this.

Download the database once you’re done uploading your images, and make sure to tick the right boxes and “Unity Editor” to **generate a Unity database**.

​​(​​Now a pop-up should appear asking how you want to name it then select **Device > Create**. Save the database in the working folder!)

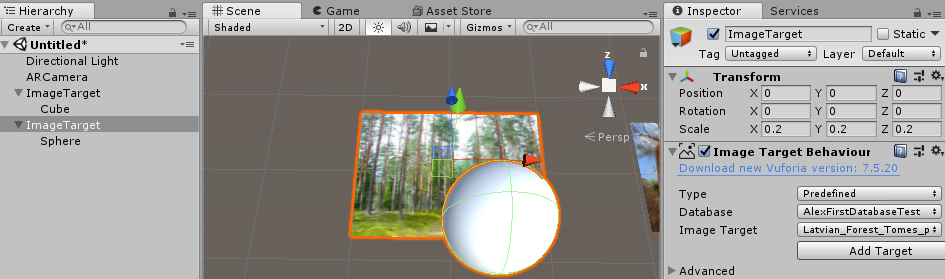
In Unity, you can now import your database through the drop-down menu:

**Assets → Import Package → Custom Package… (or double click on downloaded file)**

***Make sure to save the same images on your phone as well to be able to use them as a target when running the scene!***

1. **Create your markers in Unity**

Once you have imported the database, you should be able to create images and assign content to them just as you did in the previous step. You can change which image in the database the Image object tracks by editing the **Image Target Behavior** **component** of the **Image** object.



1. **Increase number of trackable images**

Since you will have multiple Image Targets (at least 2) you want to track at the same time you need to increase the maximum number of trackable images:

**Windows → Vuforia Configuration**

Change the value in the field **Max Simultaneous Tracked Images**.

You can extend the inspector window by clicking and dragging the edge, to see the whole line of text.

1. **Try your markers out**

Now you should have a project that you can compile into an app that should be able to track the images you’ve used to create your database.

Make sure to try your application out on an image displayed on a computer screen, so you can see how that impacts tracking.

**Potential issues:**

* If the Game object isn’t showing up, try with a darker setting on the phone.
* See if the whole image is visible in the scene when clicking the AR camera, if not play around with the positions so that it is visible and try again.
* If you still can’t get it to project your Game Object, try another picture with a higher rating.
* Double check that the settings for **Max Simultaneous Tracked Images** is still 2 in Vuforia Configurations and that the license key is still there.
* Save the project (Ctrl+S).

Other problems? Ask for help!

Part 3 – Adding content

Now that you have a framework that lets you define your markers, we want to add some content to your scene. This part contains *separate* instructions for updating/manipulating the visuals of the virtual 3D object.

Specifically, we are interested in detecting “clicks” on the virtual object, adding sound effects emulating 3D space, and implementing 2D features within the surface space of the screen. Implement as many features as you can. If you want to try to add something not covered by this part of the lab, such as a video clip that is rendered as a virtual object over your marker, you are free to do so as well.

Changing color and clicking on AR objects

In this part, you will implement a function that checks if the user pressed the screen, and if they have, detect whether they clicked on a 3D Object. Once the user clicks on the object, its color will change.

1. **Make a new script**

As all scripts are components, they need to be part of a game object. It is good practice to create a new empty game object to keep track of scripts that handle global input (shared by all/most objects). However, you can also add your new script to a game object that you know will always be present in the scene (e.g. ARCamera in this example).

You can create a new script from:

the **Create** menu by clicking on the **+** icon at the top left corner of the **Project panel** and select  **C# Script** (the Project panel is usually at the bottom)

OR

by selecting **Assets → Create → C# Script** from the main menu.

1. **Check for clicking**

Now that you created an empty script you will need to edit it.

Base script skeleton in Unity that includes a public class followed by your name for the C# script and two base functions:

| using System.Collections;  using System.Collections.Generic;  using UnityEngine;  public class ChangeObjectColorOnClick : MonoBehaviour  {  // Start is called before the first frame update  void Start()  {    }  // Update is called once per frame  void Update()  {    }  } |
| --- |

For Unity to be able to call the script, it should be derived from the class called **MonoBehaviour**. The standard functions of the Unity script are the **Start()** and **Update()** functions, which will be called at the start and regularly throughout the game. The easiest way to check for user input is in the **Update()** function. You can check for click events with the following if statement in the **Update function**:

| if (Input.GetMouseButtonDown(0)){} |
| --- |

1. **Edit the material**

When updating/changing components of a game object from a script, you first need to find the game object you want to modify.

For testing the material change/color change, the easiest approach is to find the Game Object manually. Once you have the clicking implemented, you can find the Game Object through raycasting.

Declare the following variable ***above*** the **Start()** function but still within the **public class**:

**public GameObject testObj;**

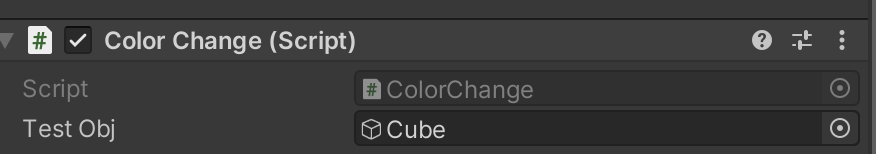
To change the visual appearance of a 3D object, you need to change the Renderer component. The easiest way to change its color is to give this Renderer a new material to use, by copying an already existing material and updating its color properties. The following code helps you change the material when the button has been pressed:

| //Fetch the Renderer from the GameObject Renderer rend = testObj.GetComponent<Renderer>();  //Create a new Material Material material = new Material(Shader.Find("Standard")); material.color = Color.green;  //Switch to new material rend.material = material; |
| --- |

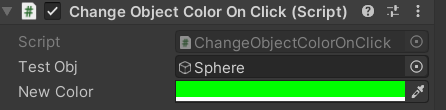
**Note:**

**You have to save the script in the text editor to make sure it recompiles. Unity should reload when saving the script.**

Add the script to the object you want to manipulate by dragging the script’s icon to the object’s **Inspector** section. You should now be able to see an entry in the script component in the Unity editor.You can drag a game object from the list of game objects here to define the variable as a reference to that object. It should look like below.



Or it can look like this if you define the color in a separate variable, e.g. public Color newColor = Color.green;



**Try your program** out to see if it can detect mouse clicks and change the color of your 3D object!

The code will only set the material to a single color and only work with a single predetermined 3D object (the one you defined as a reference in your code). If you already have changed the standard color of your 3D object to green, you can change the color you set the material to in the code.

1. **Starting with Raycasting**

Now we will try another method called Raytracing. Create a new Script called Soundcon.

To see if the user clicked on an object in the scene, we will fire a “ray” from the position rendered on the screen where the user pressed straight into the scene to see if it hits anything. Since you already detect click events (keep if (Input.GetMouseButtonDown(0)){} in this script too) all you need to do is to create the ray and trace it. The ray isn’t visible to us and is only a theoretical line in 3D space.

Start by **casting** the ray. You can get the position and direction in the scene from the (first) point hit on the screen with the following code:

| Ray ray = Camera.main.ScreenPointToRay(Input.mousePosition);  RaycastHit Hit; |
| --- |

From here, you can use the **Physics.Raycast** function to cast the ray into the scene and see if it hits any colliders.

**Parameters: a ray; an output structure; and a distance limit, after which the function will stop tracing.**

This function will return true if it detects a collision and false otherwise. When you use it in an ***if*** statement, everything on the ***then*** branch will execute if the ray hits anything.

| if (Physics.Raycast(ray, out Hit)) |
| --- |

**Leave the code as is and continue to the next section to get an output** from the raycasting!

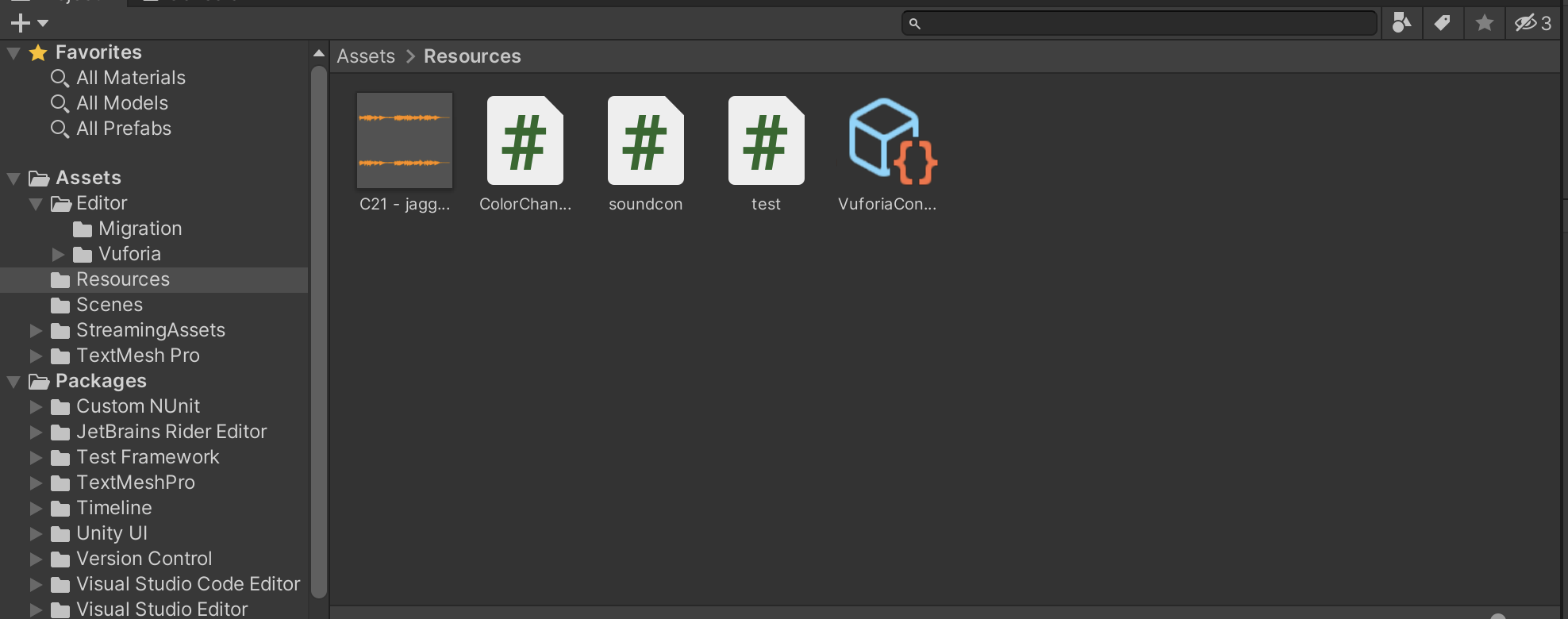
**If you have any problems with the color change (e.g., it stays purple, that is, the standard color), you should try changing the color from within the renderer’s material!**

AR Object as audio source

Unity provides a sound engine that can play sounds from specific **Audio Sources** that have a position in the 3D scene as if heard from a particular position - the position of an **Audio Listener**. In this part, you will implement a functionality that plays sounds from the 3D objects when they are clicked.

1. **Find a few sounds you want to use**

Start by finding sounds that you want to play in your scene. Unity supports a [number](https://docs.unity3d.com/Manual/AudioFiles.html) of different sound formats, in particular .mp3 and .wav. Drag a sound you like to the resources folder in unity. We will use this later! (You can try the recent text-to-audio model [Bark - a Hugging Face Space by suno](https://huggingface.co/spaces/suno/bark), to get an example .wav file.)



1. **Adding Audio Sources**

Audio Sources are components that are added to Game Objects in the scene. A good choice for **Audio Source** is the game object that you used for the **Vuforia Image Target** (can be another object too, e.g. the **ARCamera**).

**Click on/Select** this game object and add a component through the **Add Component** button in the **Inspector**. **Add** an **Audio Source** component. The easiest way to find the correct component type is to search the name ***Audio Source***to filter the collection of available components.

Now the **Audio Source** component should be right below the **Soundcon (Script)** component in the object’s Inspector view. With the default configuration of the audio source, the sound will play as soon as the virtual object is created. Vuforia doesn’t have a destroy or create a mechanism for any virtual objects. Instead, as it detects the markers, it only hides them from view. This means the sound effect will play as soon as the program starts. To change this, you need to:

**Uncheck the “Play On Awake” box in the Audio Source component**

Now that you have prevented the sound from playing as soon as the scene starts, you need a way to get the sound to play when clicking an object with raycasting.

1. **Editing the Script**

Open up the Soundcon script and add a public GameObject that will be an array with audio clips:

| **public AudioClip[] aClips;** |
| --- |

Another public variable will be added which is the Audio Source so that we can refer to our Audio Source in Unity.

| **public AudioSource myAudioSource;** |
| --- |

We feed in the aClips array through Unity so that the script can change the audio source to the (in our case) first audio clip we provide, ignoring the effect of whatever audio source we provided previously. This array is useful if you want to use multiple clips (e.g., you want to switch between them). Make sure that you provide at least one element in aClips in Unity (it sees aClips as a list). **If you don’t want to change the sound that you set previously as the Audio Source, ignore aClips completely.**

We will be working with something called a “switch case” to determine the name of the object we hit so we will also need a string to define a button name:

| **string btnName;** |
| --- |

Don’t forget the testObj variable!

In the **start function** we want to set up our Audio Source:

myAudioSource = GetComponent<AudioSource>();

We will now continue where we left off in the last step (step 4). In this if statement we can see if the raycast has actually hit the button name. Complete the previous implementation of the Update functions with the following updates:

btnName = Hit.transform.name;

Now we want to set up the switch case by writing the trigger which in this case is a sphere but it depends on the gameobject you will use to click and get a sound. This will trigger the first sound in our array:

switch (btnName)

{

case "Sphere":

myAudioSource.clip = aClips[0];

myAudioSource.Play();

break;

default:

break;

}

We will only need one sound for this lab but feel free to apply this script to multiple objects but remember to add cases and more audio clips (aClips) in Unity.

1. **Add script to Image Target and add your audio clip**

Save the script and drag it to the inspector for the ARCamera (or any other component) where you have your audio source. You should now be able to add an audio clip and the Image Target as an audio source.

1. **Compile and Run!**

You should now be able to run this part and hear your audio clip when you press the object!

**After you’re done, you can do the optional part. In any case, send the resources (code, images, sound) to the current TA per mail (email address in assignment description) after presenting. Make sure to include the names of all your group members. Please be prepared to present**

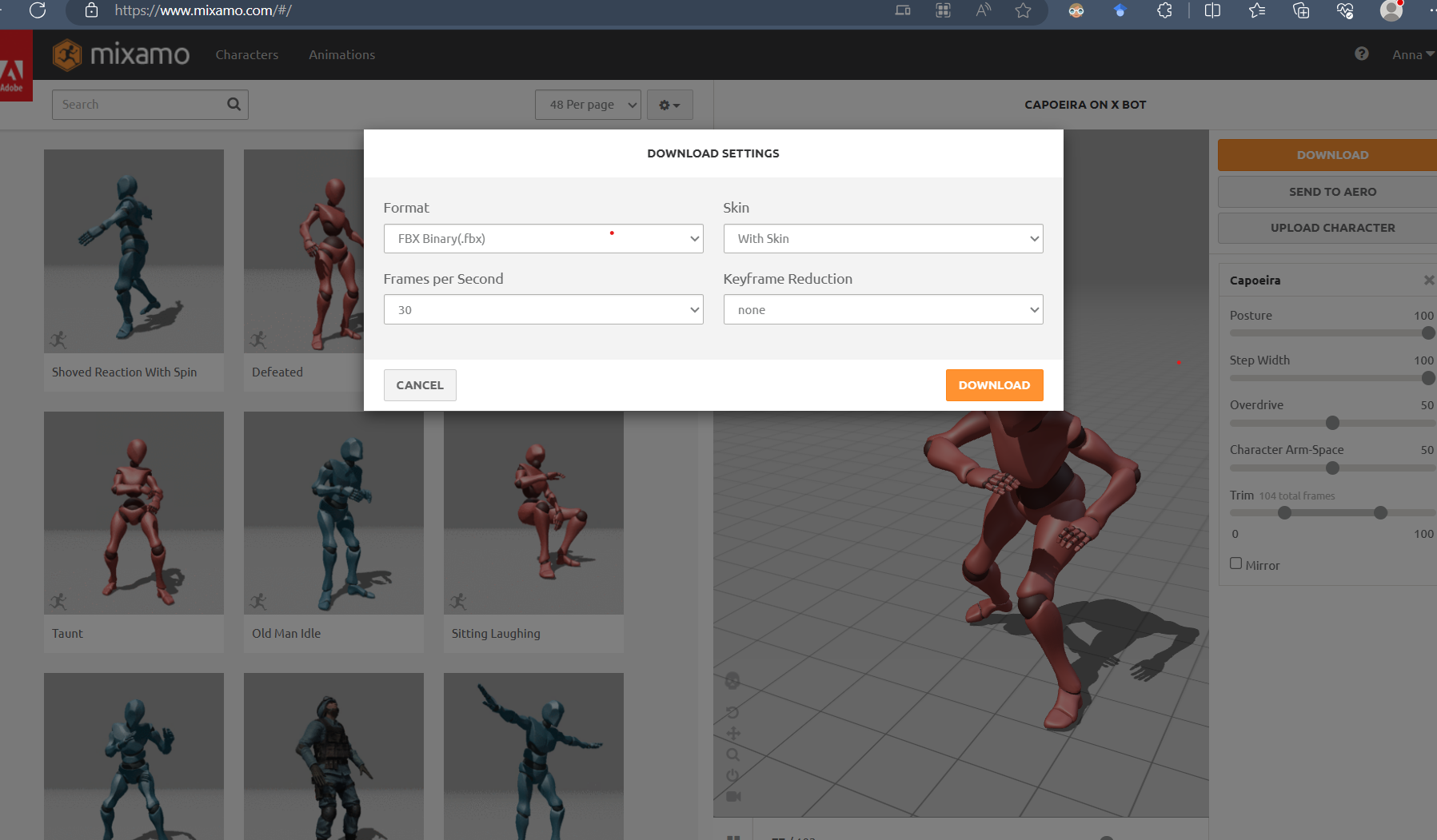
1. **two images in the same run,**
2. **show two game objects that pop up,**
3. **play the sound and change the color of at least one of the objects.**

(OPTIONAL) Animations (if you have time - fun, takes about 5 mins)

1. **Add animated character on surface**

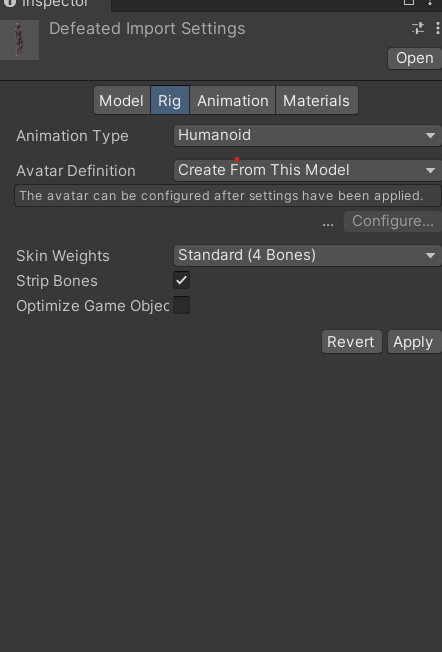
**1.Creating Avatar**

Go to [Mixamo](https://www.mixamo.com/#/), select character animation (e.g. Capoeira). Download the FBX model with the following settings.

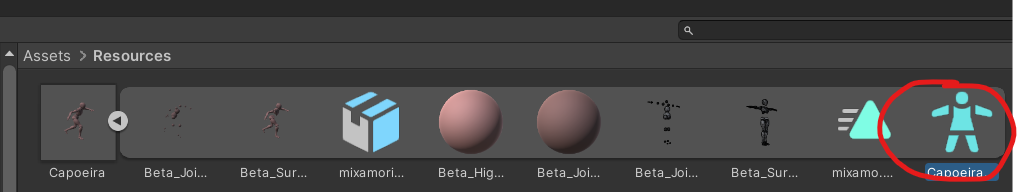
****

Put/drag the downloaded .fbx model in your Unity project’s Resources folder.

Select/Click on the imported 3D model, and in inspector, create an avatar, by selecting Humanoid and Create From this Model.



The new avatar will appear under your character in the Resources folder.



1. **Repeat previous steps for creating Image Target**

Create new Image Target, choose one of your custom images (not used before) in Image Target Behavior.

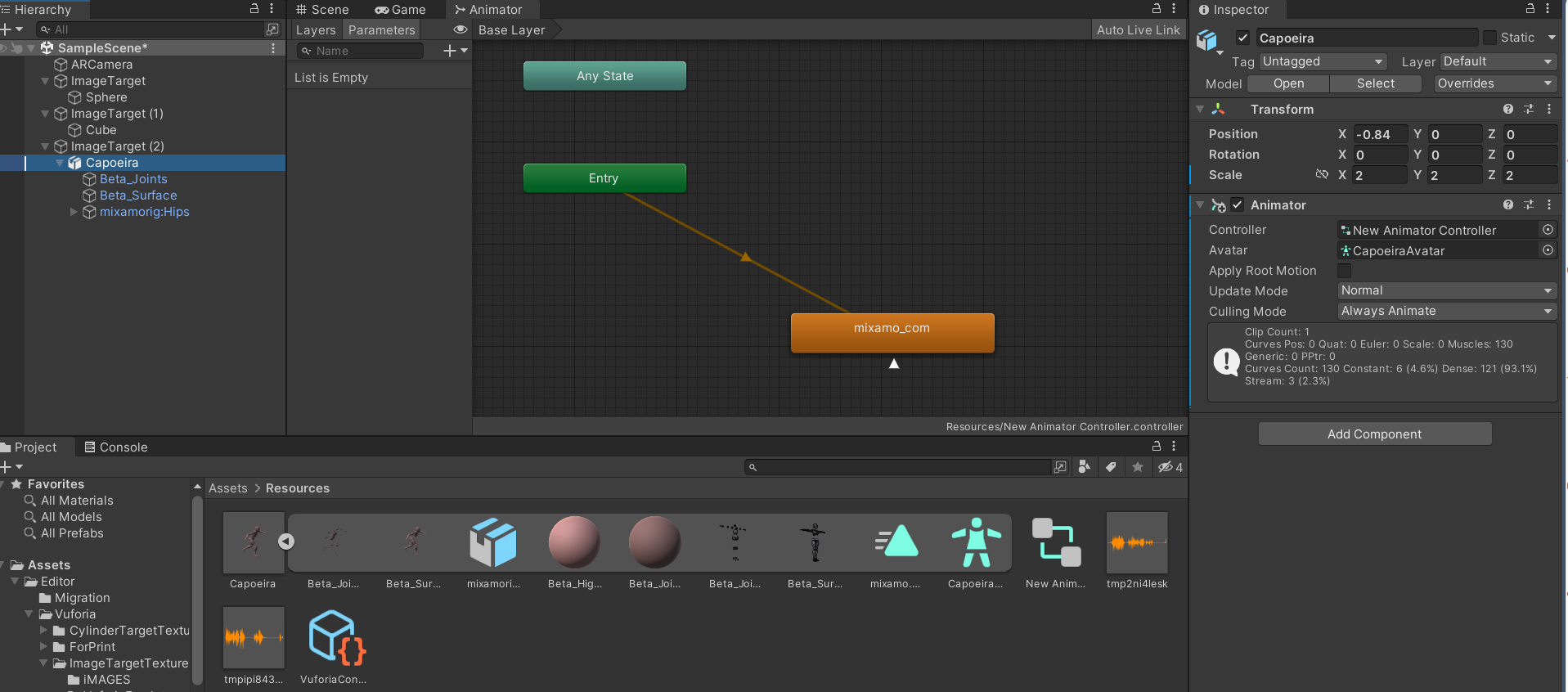
Instead of a standard 3D object, drag the Capoeira (or other 3D animation) as a child to the new Image Target.

1. **Adding animation to your 3D character.**

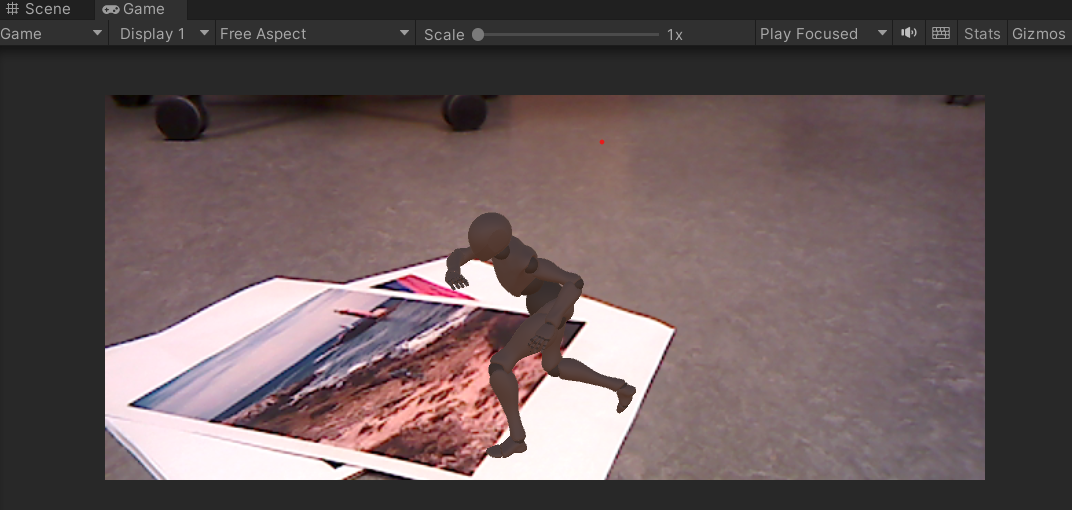
Add Animator component in inspector to the 3D object (now Capoeira model).

You need to add two elements: Avatar and Controller.

1. Assign the Avatar created in step 1 (screenshot above) to the Avatar selectable.
2. You will also need a controller, for this go to your project and click on Create/Animator Controller (this will be named ‘New Animation Controller’ by default. Clicking on it brings up the Animator tab. You can get the mixamo animation by dragging the triangle shaped animation under your character to the Animator interface. This automatically connects the animation to the entry point (right click on mixamo.com animation - add transition and right click again on the mixamo\_com will loop your animation.)



In the end, you should see the character dancing/moving on the chosen image surface.

****

[Vuforia-Mixamo - YouTube](https://www.youtube.com/watch?v=iIk0T_aeq3I) it moves!

Part 4 – Clean Up

**Make sure to save your project if you want to use it for later.**

**If you're finished please let me know so I can see your prototype and then you can go!**