Pickrick AR Project Documentation

created Fall 2021

**Table of Contents**

Project Setup 1

Android Build Setup 3

Video Tutorials 6

Project Overview 6

What Scene to Use 6

Unity Collab 7

Scene Hierarchy Overview 8

AR Session 8

AR Session Origin 9

GPS Hotspot Object 13

UI 15

Parent UI Object/Page Controller 15

Introduction UI 17

Hotspot UI 18

Menu UI 21

Timeline UI 22

Event\_InfoPage UI 24

Event\_MoreInfo UI 26

AR Objects 30

Audio Manager 32

Reporter 33

**Project Setup:**

The project setup for this project should be fairly simple. This project uses Unity Collab for source control which is built into Unity. Here are the steps to set up the project…

1. Download Unity Hub here (<https://unity3d.com/get-unity/download>). The latest version of Unity Hub should work. Note: You should create a Unity account if you do not have one already.
2. Open Unity Hub and go to the Installs Tab and press on Install Editor

Graphical user interface, text, application

Description automatically generated

1. Under the official releases section, press the install button for version 2019.4.31f

A screenshot of a computer

Description automatically generated with medium confidence

1. You will see a window like in the image below. Make sure you put a blue check mark and download Android Build Support, Android SDK & NDK Tools, Open JDK, and IOS Build Support, and Microsoft Visual Studio Community 2019. If you want to use Visual Studio Code instead of Visual Studio as your IDE (which I recommend), you can use this link… (<https://code.visualstudio.com/docs/other/unity>)

Graphical user interface, text, application

Description automatically generated

1. After everything is installed properly, you need to be added to the Unity Collaboration project. Once you are added, open or reopen Unity hub and go to the projects tab.
2. You should see a notification on the bottom of your screen like the one below. Once you see this, hit the refresh button.



1. Next, you should see a project in your projects tab called PickrickAR\_1230. Click on the download button next to his project to download it.



Note that the project will look something more like this and be greyed out with the download button showing

Graphical user interface

Description automatically generated

8) Now you should be able to open the project!

**Android Build Setup:**

1. Before a project is built on Android, we have to change the gradle version
2. Go to this site (<https://gradle.org/releases/>) and download version 5.6.4 of gradle. Press on “complete” to download the entire gradle version.
3. You should have a zip file called “gradle-6.9.1-all.zip”. Put this file anywhere you want, unzip the folder, and make sure you know the file path.
4. Go back into Unity and go to Edit -> Preferences -> External Tools
5. Find the Android options and uncheck “Gradle Installed with Unity”

A screenshot of a computer

Description automatically generated with medium confidence

1. Copy and paste the file path in the text box next to the word Gradle like the image above
2. Use this link as a reference if you run into any trouble (<https://developers.google.com/ar/develop/unity/android-11-build#unity_20193_20194_and_20201>)
3. To build the AR application on Android, go to File -> Build Settings
4. If needed, change the platform to Android by clicking on Android in the platform section of the build settings window

Graphical user interface, text, application

Description automatically generated

Graphical user interface, application

Description automatically generated

1. If under run device, you see “Android SDK and/or JDK directory is not set” like shown below…

Graphical user interface

Description automatically generated

Do the following…

* Find the file where your Unity version 2019.4.31f is installed
* Than follow this file path: Editor\Data\PlaybackEngines\AndroidPlayer
* Make sure this directory has all the folders and files of the following

Graphical user interface, text, application

Description automatically generated

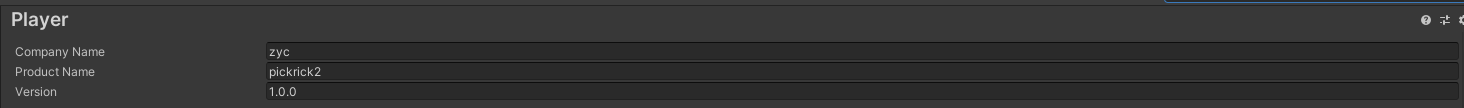
* Go back to Unity, and go to Edit -> Preferences -> External Tools
* Look under the android settings and make sure there are no warnings like shown below. Also, make sure the file path matches the folder you found in the previous steps

A screenshot of a computer

Description automatically generated with medium confidence

* If you have an error like this, you can copy the SDK, NDK, or OpenJDK folders and put them in the directory in the previous steps by finding zipped up versions of these folders under Assets -> AndroidBuildFolders in the Unity project

1. When in build settings, press the Player Settings button
2. When in these settings, you can change the product name to whatever you want the name of the built app to display as



1. Now you are ready to build to Android, plug in the device you want to build the project
2. On the top left corner of build settings, make sure the scenes in the build include the MainScene\_Updated like shown below

Graphical user interface, application

Description automatically generated

1. Under run device press, refresh
2. You should than see your device you plugged into your computer. Make sure this is your run device

Background pattern

Description automatically generated

1. Press the “Build and Run” button on the bottom right corner and name your .apk file whatever you want

**Video Tutorials:**

This link below links to a YouTube playlist containing 8 videos about how to use the project. The videos cover most of the documentation shown below and should give you a good understanding about the project, but the documentation below can be used as a reference and goes into more detail.

<https://www.youtube.com/playlist?list=PL8xj0gkjWKm3v0CkbcC7PjxrPDgdr6xWf>

**Project Overview:**

**What Scene to Use:**

This current version of this project only uses one scene. This scene is called **MainScene\_Updated**. To open this scene, in the Project window, go to **Assets->Scenes->MainScene\_Updated**. Make sure that your scene hierarchy view looks like below.

Scene Hierarchy View

Graphical user interface, text, application

Description automatically generated

**Unity Collab:**

You should also make sure Unity Collab is On. To do this, open the Services window by going to **Window->General->Services**. In the services window, turn Unity Collaborate on.

A screenshot of a computer

Description automatically generated

You should now see a button that says collab on the top right of the screen. This is how you will push changes and pull changes other people have made. When you make changes, you will see a blue error going up on the collab button, and you will see all the changes shown in the window shown below. Write a brief description of your changes and press Publish Now! To publish all your changes.

Graphical user interface, text, application

Description automatically generated

Normally before working on anything, check the collab window to see if there are any new changes from other people to pull.

**Scene Hierarchy Overview:**

This section describes what all of the Game Objects do in the scene used for this project.

**AR Session:**

This Game Object contains the AR Session and AR Input Manger scripts. These are both necessary scripts to configure AR in Unity. Just make sure this object stays in the scene.

Graphical user interface, text

Description automatically generated

**AR Session Origin:**

This Game Object contains scripts necessary to run an AR project. It contains the AR Camera, AR Floor, and some scripts from the AR Location Plugin.

Graphical user interface, application

Description automatically generated

1. **AR Session Origin (Parent Object) –** This Game Object contains the AR Session Origin and AR Plane Manager scripts which are necessary to create an AR scene. Notice AR Session Origin has a reference to the AR camera.

**A screenshot of a computer

Description automatically generated with medium confidence**

1. **AR Camera –** This game object contains a Camera component which is the default camera component in Unity. It also contains the AR Pose Driver, AR Camera Manager, and AR Camera Background scripts which are necessary scripts to have an AR scene in Unity.

**Graphical user interface

Description automatically generated**

This game object also contains a **HotspotArrowHolder** game object which contains a script called Hotspot Arrow which causes a Red Arrow 3d Model to point towards the longitude and latitude to the **GPS Hotspot Object.** The children of this game object are the actual arrow 3D model and a directional light for the model. This game object is hidden once the player is the proper location to launch the rest of the AR scene and the setup part of this app is complete.

Graphical user interface, application

Description automatically generated

Inside the **Hotspot Arrow Script,** there is a GPS Hotspot Object script which references what the arrow is pointing towards and a rotation speed parameter which determines how fast the arrow rotates when adjusting its position to point towards the GPS Hotspot object transform.

1. **AR Location Root –** This game object comes from the AR Location (<https://docs.unity-ar-gps-location.com/#main-features>) plugin used for this project. This game object is basically settings about how objects using the AR Location plugin spawn based on given coordinates. These are all the default settings, but a lot about spawning based on location can be edited here.

**Graphical user interface

Description automatically generated**

1. **AR Floor –** This game object creates a transparent floor in the AR scene that allows shadows to be shown. The plane and mesh renderer components make up the object geometry, and the ARFloor shader makes this geometry invisible but allows for shadows to still be created. The Follow Camera position is an AR Location script that allows the AR Floor to adjust based on the camera position.

**Graphical user interface

Description automatically generated**

**GPS Hotspot Object:**

This game object uses the Hotspot script from the AR Location plugin. Basically, this script detects if a player is in a certain range of a specified location. When looking at the script component, you can see that the **latitude** and **longitude** of the hotspot were specified below.

Inside the **Hotspot Settings** part of this component, you can see the **Activation Radius** parameter. This determines how far the player must be for the hotspot to be activated. For example, the activation radius is 1e + 11 which means the hotspot can activate from very far away which can be useful for testing purposes. A value around 10 is what was previously used for the activation radius in the latest build of the project. The **Prefab** parameter is the object that is spawned when the player is in the hotspot. The GameObject prefab object used for this project is simply an empty game object which is the child of this game object, which is just used as a placeholder, so the hotspot script will run properly.

The **On Hotspot Activated** and **On Hotspot Leave** events are both used in this project. On Hotspot Activated references a function called **OnReachHotspot** and On Hotspot Leave references a function called **OnExitHotspot.** These functions are referenced from a script called **UIController\_HotspotScene** found inside the **UI -> Hotspot** game object in the scene. These functions basically hide and unhide certain UI elements.

**Graphical user interface

Description automatically generated**

**UI:**

All of the UI elements contained in this scene are under this Game Object. Each child object of the UI game object represents a page type and all page types are controlled by one script called the **Page Controller** in the UI parent object.

Text

Description automatically generated

1. **UI (Parent Object)**

This game object contains a script called **Page Controller** which contains all the categories of UI this project uses. Notice there are 16 child objects and 16 pages in the page controller script. This is because each child of this game object represents a category of UI. This is not necessary to do but is very useful for organization purposes. Inside the Page Controller script, you can check or uncheck the **Debug** parameter. Checking the debug parameter can be useful if you think there is a problem with the page controller or overall UI system.

Notice the **Pages** array have elements that reference a script. Each element in the Pages array represents a **Page** script. Each of these 16 children have a page script attached which allows it to connect to the Page Controller. Each Page script is also associated with an Enum called **PageType.** You should create a PageType for every new category of UI you want to make. You can create a PageType by editing the script under Assets -> Scripts ->UI->PageType.

To control what pages are hidden and shown, you do this in a script by referencing the Page Controller script and calling functions within this script. You turn a page off by calling **TurnPageOff(PageType)** and turn a page on by calling **TurnPageOn(PageType).** Note that you can also turn a page on simultaneously while turning a page off by using an optional second parameter in the TurnPageOff function. For example, calling TurnPageOff(Type1, Type2) would turn the page with Type1 off and turn the page with Type2 on.

Finally, when looking at the Page Controller component, you can see an **Entry Page** parameter. This parameter essentially allows you to assign a PageType that appears on screen when a scene is first launched. For example, the Introduction page type is launched when first running this project.

Graphical user interface, text

Description automatically generated

1. **Introduction**

The **Introduction UI** are the first screens that the user sees when they open the app. The first and last screens of the introduction UI are shown below. The **Introduction UI Controller** script contains all the logic for this category of UI. This script adds all the children of the Introduction game object to a list in the order they appear on the scene hierarchy.

Each page for the introduction UI contains a button which runs the **GoToNextIntroductionPage()** method. This method hides the current UI page and unhides the next page in the list of pages until the last page is reached. When the last UI page is reached, the UI Controller will switch to the Hotspot category of UI and hide the Introduction UI. This script also shows the Menu UI once the player enters the second introduction UI screen.

Text

Description automatically generatedGraphical user interface

Description automatically generated

1. **Hotspot**

The **Hotspot UI** are the second screens the user sees when running the project. These screens are essentially the setup before the user can launch the AR scenes and timeline. The player needs to be in the right location and have the phone in the right direction so the AR can spawn in an accurate location. The **UI Controller Hotspot Scene** script controls the logic for the Hotspot UI. The user will see three screens when looking through this part of the project.

Graphical user interface, text, application

Description automatically generated

1st Screen- Going to the Proper Location:

The player will see a red arrow, instructions, and reference images of the location they need to go to get to the proper location. The user can press the **Where?** button to hide and unhide the reference images of the location the user needs to go to. The player also can follow the direction the red arrow is pointing to go in the proper direction of the location. When the player enters the Hotspot, the second screen for the hotspot UI will be shown. If the player exits the Hotspot, they will return to this screen. There is a Boolean called **hotSpotActivated** which keeps track if the player is in the Hotspot and controls what screen is shown.

2nd Screen- Compass (Looking in the right direction):

The player will get a new set of written directions and a compass object with a red marker for the next screen. The compass keeps track of the orientation in degrees of the player and contains a uT meter. There is a **Compass game object** inside the Hotspot UI which contains a script called **IsCompassSetUpCorrectly.** This makes sure the uT meter is at a proper level and that the user if facing in the right direction. The player will go to the final screen once the IsCompassSetUpCorrectly value is true.

A screenshot of a weather map

Description automatically generated with low confidence

There are three children of the Compass game object: another game object called **Compass, uTMeter, and Target Direction**. The Compass and uTMeter game objects were all taken from the compass plugin. However, the Target Direction game object does not use this plugin and determines what direction the user needs to face for the compass to be set up correctly. You can set the values of where the player needs to be facing as shown below.

Graphical user interface, text, application

Description automatically generated

The compass was implemented using this plugin: <https://assetstore.unity.com/packages/tools/integration/compass-81806>

3rd Screen- Set Up Complete:

The third screen confirms that the AR project is properly set up. The place where the instructions were for the past two screens now says the AR experience is calibrated and the user is instructed to press the **Launch AR** button. This button will hide the Hotpost UI and shows the **Timeline UI** which begins the core experience of the app.

A picture containing text

Description automatically generated

1. **Menu**

The **Menu UI** is activated after the very first button on the Introduction UI is pressed and always stays activated. The logic for the Menu UI is all found under the **Menu Controller** script attached to the Menu game object. All the elements under this game object are the UI components for the menu.

A screenshot of a computer

Description automatically generated with medium confidence

To use the menu, you press on the icon with the three white horizontal lines which runs the **ShowMenuPopUp()** function in the Menu Controller script. Like the image shown below, you will get a menu pop up with **an Exit, Our Team, and Quit Button**. The Exit button is a grey X logo and will run a function called **HideMenuPopUp()** which shows the player only the three white horizontal lines again. The Our Team button will lead to an **acknowledgments page** that is not yet finished. When in the acknowledgments page, you can press the return button which is a grey arrow on the top left of the screen like shown below. The Quit button will exit the application but this only works when building the app and not in the editor.

Text

Description automatically generatedGraphical user interface, text, application, chat or text message

Description automatically generatedGraphical user interface

Description automatically generated with medium confidence

1. **Timeline**

Timeline Overview:

The timeline is a historical timeline that allows users to view different historical events surrounding the Pickrick restaurant. The user clicks on the image associated with each event to launch the event. The types of events are organized into two categories: **info events and AR events**. Info events shows the users historical events through text, videos, and images. AR events show the users historical events using AR, in addition to things the info events show.

A picture containing text, piano

Description automatically generated

Game Object Hierarchy:

The timeline UI contains a timeline object which is an empty game object with a Page script to connect it to the Page Controller. This parent game object contains a scroll area which contains a Scroll Rect, which is a UI element in Unity which allows the player to scroll through UI. Some settings in the Scroll Rect are specified which controls how the user can scroll through the timeline. For example, the user can only scroll horizontally. Within this ScrollArea is the **Timeline Container** object which contains all the actual **Timeline Events**.

Text

Description automatically generated

Timeline Event Manager:

The Timeline Container game object contains a script called **Timeline Event Manager** which stores and controls all the Timeline Events. For organization purposes, all the timeline events are children of the Timeline Container. The Timeline Event Manager script has a list called **Event List** which contains all the timeline events. Each Timeline Event contains a **Timeline Event Script** which allows it to be added to the Event List. The Change Timeline Event Animation speed variable controls how fast a timeline event will snap to the center of the timeline. The Selection Marker is a reference to the red rectangle that marks which event is currently viewing.

Graphical user interface, text

Description automatically generated

Timeline Events:

The Timeline Events as mentioned before all contain a Timeline Event script. The Timeline Event game objects also contain an image that acts as a button where you can add the picture that associated with each timeline event. This image is also a button which runs the **PressOnEvent()** function within the timeline script that launches the historical event and informs the Timeline Event Manager that the event was changed. The Timeline Event script also contains a Boolean called isArEvent which determines if the AR logo should be shown for each timeline event. Timeline events also have child game objects called **Event Name and Event Date** which are text UI objects that can be edited for each event.

Graphical user interface, text, application

Description automatically generated

Prefabs:

Instead of making a timeline from scratch, you can look under Assets -> Prefabs -> UI to find a prefab called **Timeline** which creates an empty timeline with no events. You can then use the **Timeline Event** prefab to create a timeline event game object. You only need to edit the UI elements and add your new timeline event to the Event List in Timeline Event Manager to get the timeline to work properly and show the information you want.

1. **Event\_InfoPage**

Each Timeline Event either links to an info event or an AR event. To make an info event, you need to create a **Event\_InfoPage.** This type of page shows the name and date of a historical event, text information about this event, and either a video or image associated with this event.

Creating an Info Page:

To create an info page, drag either the **Event\_InfoPage\_Image** or **Event\_InfoPage\_Video** prefab found under Assets -> Prefabs -> UI into the scene under the UI object. If you want your historical event to include an image, drag Event\_InfoPage\_Image, and if you want your historical event to include a video, drag Event\_InfoPage\_Video.

Notice that these prefabs include a Page script and **Event\_InfoPage** script. The Event\_InfoPage script controls how this Page you just created turns on and off. Create a unique PageType and change the type to this new page type you created to set up this event correctly.

Graphical user interface, text, application

Description automatically generated

Editing the Info Page:

When you add one of the info page prefabs, you will see something like the image below. You can edit what the event looks like by changing the text in the **Event Name, Event Date, and Event Description** game objects. If you select the image prefab, you can simply drag the image you want by finding the Image component under the **Image** game object. If you want to add a video, you need to create a new **Video Renderer Texture** and place it under Assets - > UI -> Videos. Then drag this texture under the **Texture** variable in the **Raw Image** component inside the **Video Renderer** game object. At this same texture in the **Target Texture** variable in the **Video Player Component** inside the **Video** game object. Lastly, add the actual video in this same component in the **Video Clip** variable.

A picture containing graphical user interface

Description automatically generated Graphical user interface, text

Description automatically generated

Connecting the Info Page to Timeline:

The last thing you need to do to set up a timeline event with an info page is reference this page to the timeline event. To do this, assign the **Timeline Event variable** in the Timeline event script to the info page game object you created. This will connect the timeline event to your info page, and the info page will show on screen when you touch the associated timeline event.

Graphical user interface, application

Description automatically generated

1. **Event\_MoreInfo**

When you create an AR Event on the timeline, you also create a More Info Page which allows the user to read about the AR event they are looking at. The user clicks on a button that says More Info to view the information on this page and clicks a back arrow to hide this information. Just like the Info Page, the More Info page shows the name and date of a historical event, text information about this event, and either a video or image associated with this event.

Graphical user interface, application, Teams

Description automatically generatedA screenshot of a computer

Description automatically generated with medium confidence

Creating and Editing a More Info Page:

Creating a more info page is very similar to creating a normal info page. Drag either an **Event\_MoreInfo\_Image** or **Event\_MoreInfo\_Video** prefab found under Assets -> Prefabs -> UI and drag this game object under the UI game object. Create a new page type and add it to this object just like you do for a normal Info Page.

Editing the more info page works the same way as you would edit the info page, so refer to the previous page for this.

Setting Up Buttons and Connecting AR Event to More Info Page:

Each More Info Page needs to be connected to an actual AR Event. For organization, all AR Events are found under the **ARObjects** game object. The parent game object for all AR Events contains a **Event\_AR** script, so find the appropriate game object with this script to connect to your more event page. To connect your More Event Page with the AR object, drag the More Info Page into the More Info Page variable in the Event\_AR script.

Graphical user interface

Description automatically generated with medium confidence

Under the **MoreInfo** and **Back** game objects in the more info page, there are two buttons you need to assign functions to. On the OnClick Event for these two buttons, reference the AR Object that you just assigned this more info page to. Add the **ShowMoreInfo()** function to the more button and the **HideMoreInfo()** function to the back button.

Graphical user interface, text, application

Description automatically generated

Graphical user interface, text, application

Description automatically generated

Graphical user interface, text, application

Description automatically generated

Navigation:

The **Navigation** game object in the more info page allows you to target an object in the AR scene you want the player to look at. If the user is left of a target object, a right arrow will show, and if the user is right of a target object, a left arrow will show. If the user is looking at this target object, no arrow will be shown, and audio will play. When the player is not looking at this target, the audio will stop playing.

Under the Navigation game object, find the **LeftRightNavigation** script. This script controls all the logic described above. Go to the AR Object associated with this page to find an object to target and assign the Target variable to this game object in the LeftRight Navigation script. Next, assign the AR Camera variable with the camera in the scene. If you want to add the audio feature described above, check the Play Sound Boolean. Lastly, assign the Audio Controller object and pick the sound you want to play. The audio logic is described later in the documentation in the Audio Manger Section.

Graphical user interface, text, application

Description automatically generated

**ARObjects:**

This game object is where all the models and animations are located for AR Timeline Events. For each of the AR Scenes, there is one parent object containing all the models and animations. There are also some lighting related game objects found under this.

1. **AR Objects (Parent Object)**

This parent object contains two scripts: the **PlaceObjectInFrontOfCamera** script and **GroundHeight** script. The PlaceObjectInFrontOfCamera script places the AR Objects in the proper location and runs right when the player has the phone in the right direction according to the compass. GroundHeight is a script that came with the ARLocation Plugin that orients a game object on the real world ground.

Graphical user interface

Description automatically generated

1. **AR Events/Connecting AR Events to Timeline**

For each AR Event, there is a single parent object that contains an **Event\_AR** script. This script allows this AR Event to attach with the timeline. Go to the timeline event you want associated with this AR Event and assign the Timeline Event variable inside the Timeline Event script to the Event\_AR script. Now this AR Event object and all the children on this AR Event containing the models and animations will turn on when the associated timeline event is pressed.

Graphical user interface, text, application

Description automatically generatedGraphical user interface, text, application

Description automatically generated

1. **Animations**

The animated objects in the AR scenes are animated using Unity’s Sequencing Timeline. If you look at all the children object under an AR Event, you’ll find a game object labeled as the timeline. This timeline contains a **Playable Director** script which launches the Unity timeline. More information about how this worked can be found here (<https://docs.unity3d.com/Packages/com.unity.timeline@1.2/manual/index.html>) or simply by Googling “Unity Sequencing Timeline.”

A computer screen capture

Description automatically generated with medium confidence

**AudioManager:**

The **Audio Manger** game object contains all the audio clips used for this project in a way very similar to the UI Controller. The **Audio Controller** script specifically contains all these audio clips and organizes them into a list of tracks. Each track contains an audio source and that a list of audio tracks. Note that this project only contains one audio track because only one audio file plays at a time. If you wanted multiple audio files playing at once, you would need to create more audio tracks.

Graphical user interface, application

Description automatically generated

You add audio clips to each track by adding the audio clips to a list called Audio. First, each audio clip needs an **AudioType.** Just like the PageType in the UI Controller, the Audio Type is an Enum that assigns an audio clip to a name. You should create a new AudioType for every new audio clip, and you can add a new AudioType by editing the script under **Assets->Scripts->Audio->AudioType**. Once you create a unique audio type, add the audio clip and audio type to a new element in the Audio array like the above images displays.

To control audio clips playing, stopping, and restarting, you do this in a script by referencing the AudioController script and calling functions within this script. You play an audio clip by using the **PlayAudio()** function, stop an audio clip by using the **StopAudio()** function, and restart audio using the **RestartAudio()** function. These functions have three parameters. The first parameter you put the actual audio type you want to play, the second parameter is a boolean where you can decide if you want the audio to fade, and the third parameter is a float that allows you to assign a delay to the audio. The image below is an example of how these functions are used.

Text

Description automatically generated

**Reporter:**

This Game Object allows you to check the console when you are testing the app on a mobile device. This uses a Unity package called Log Viewer. (<https://assetstore.unity.com/packages/tools/integration/log-viewer-12047>) This game object was created by going to Reporter and pressing create like in the image below. When testing the project on a mobile device, **quickly move your finger in a circular motion to see the debug log.**

Graphical user interface, text, application

Description automatically generated