Overview

Onboarding Resource Application | ORA

Week 1: Main Menu + Screen Flow

What is ORA?

ORA is a mini-project built off last year's project AURA designed to give the development team hands-on experience with Unity, MRTK3, and NASA's tech stack. Throughout October, AR, Web, and AI developers will work through structured weekly tasks that ensure everyone gains a baseline understanding of the tools. ORA is intentionally small in scope but focused on collaboration, version control, and shared learning. The goal is not only to complete the project but also to build team readiness for larger-scale development later in the year.

Workflow Breakdown

Teams will consist of 6 to 7 members, with a structured base project given over 5 weeks to complete. Each week begins with individual tasks that allow every member to build their own understanding of the problem, followed by group challenges completed in teams of 2 to 3 using Unity and Live Share in VS Code. ORA Team leads are responsible for making sure members push their progress at the end of each week and for integrating all contributions into a single shared team branch.

ORA Teams

ORA-1	ORA-2	ORA-3	ORA-4	ORA-5
John Rogan	**Haotian Li**	**Kyran Park**	**Steven Fu**	**Tanvi Naik**
Eugene Xu	Andrew Yang	Anna Lin	Linus Wang	Amara Small
Darrin Du	Insu Jung	Jakelin Serrano	Dylan Li	Gloria Chang
Melinda Cordero Salinas	Ayush Madhav Kumar	Shaurya Pratap Singh	Sam Schneider	Hanan Husein
Aaron Sun	Mei Yi Yang	Evan Wang	Catherine Jin	Lalin Ozcan
Kenneath Zhang	Emma Zhang	Prisha Agnihotri	Amy Shi	Alex Liu
Alex Bartolozzi	Mahee Kavdia	Nathan Yap	Yatin Bichala	Jeffrey Zou

Individual Development (Base Tasks)

Each team member must complete the base task on their own. This ensures that everyone learns the fundamentals of Unity and contributes consistently to the project.
Weekly expectations for each member:

- At the beginning of each Wednesday meeting, every member must pull the latest version of their team's branch – `ora-<1/2/3/4/5>`
- Members will work on a copy of this main scene each week, scenes should be named clearly – `<feature>-<name>  ex: MainMenu_MollyM`
- Leads will introduce the base task for the week. These will be straightforward and follow set guidelines for learning.
    - Leads are expected to help through screen-sharing and live walkthroughs but should not directly code for members. Their role is to guide, clarify concepts, and support problem-solving...
- Once the base task is complete, members push their changes to their own branch following specific naming conventions – `ora-<1/2/3/4/5>-<name>`
- This stage is non-negotiable: each member must complete the base work before participating in group tasks, failure to do will be noted accordingly.

Group Collaboration (Group Tasks)

After everyone on the team has completed their individual task, members can begin working in groups of 2 to 3 people on the group tasks (after they push to individual branches). These challenges are designed to deepen understanding by exploring creative or advanced applications of the week's topic.
Structure of group tasks:

- Group tasks will extend the base functionality
- Group work is done using Unity on a single member's computer on a group branch scene pulled from one of the members' individual branches, ideally with Live Share to allow collaborative editing while avoiding merge conflicts.
- The group's completed scene should be pushed to a dedicated branch
    - `ora-<1/2/3/4/5>-week-<1/2/3/4>-group-<1/2/3>`
- Multiple group branches may exist each week per team, but only one group scene will be selected for integration into the main ORA team branch, so try to get as many bonus points as possible!!

Weekly Integration and Branching Strategy

The responsibility for integrating code falls on the ORA team leads. Their job is to review and merge work efficiently and accurately. It is essential that all members communicate any issues they encounter with Git or the integration process to prevent disruptions and maintain smooth collaboration across the teams.

Branching Structure

- Individual branches: one per member, each containing a personal scene copied from the team branch with that week's base task setup.
- Group branches: one per group, containing a collaborative base scene
- Team branch: the centralized team branch that contains a single, updated scene reflecting the latest baseplate and selected group task.

Integration Process

After each onboarding meeting (Sunday), leads review all group branches and choose one representative group scene to merge into the team's main scene. This merged scene will be the official version used to reflect team progress and includes all required base functionality and the selected group implementation. Leads are expected to complete merging before the next Wednesday meeting to ensure the baseplate is current and usable for the following week. Over time, the main scene will evolve to include the cumulative work of all prior weeks (base + group), serving as a record of the team's growth and contributions. Additionally, group tasks will have point values assigned based on completion and creativity, which will be used to determine a prize for the top three contributors. The *Merch Committee* will coordinate the prize planning throughout October.

Point Opportunities

Each week, members will have opportunities to earn points based on task completion, difficulty, and creativity. Point values will vary depending on the feature and specific task. Group tasks will have a completion point scale from 1 to 5, with additional creativity points usually awarded. Base tasks may also receive creativity or difficulty points. At the end of the month, points will be totaled individually to determine **prizes for the top 3 contributors**. The *Merch Committee* will coordinate prize planning throughout October.

Summary Workflow – NOT Week 1

At the start of the Main Wednesday week:

- Leads look at groups 1/2/3 and choose the best scene and pull that branch into the team branch:
  `ora-<?>-group-<?> → ora-<?>`
  - Takes the prefab created in the group folder and drags it into the team scene to then push before the meeting
- Everyone pulls the latest from their team branch and opens the current week:
  1. `ora-<?> → ora-<?>-<name>`
  2. `Assets > CLAWS > ORA > Week-<?> > Base-Tasks > name`
- Base tasks are assigned and must be completed individually.
- Members push completed work to individual branches:
  `ora-<?>-<name>`
- Leads pull the 1st persons completed base task into the team branch and push
  `ora-<?>-<name> → ora-<?>`

After individual completion:

- Members form small groups in their ORA teams to work on group challenges using Live Share. Switch from individual branch to group branch:
  `ora-<?>-<name> → ora-<?>-group-<?>`
  - Groups will be assigned by leads in completion order
- Each member must pull from the team branch into their *local* group branch:
  `ora-<?> → ora-<?>-group-<?>`
- Groups will only submit ***ONE*** scene for the group portion, so its up to members to decide whether to have 3 scenes and share them remotely or work on one person's laptop
- Each group will push whatever they finish and leads will score the points for the tasks for each person as group, but tallied individually based off the rubric listed for that week:
  Example:
  - Week 1– group 1: John 5/5, Ky 5/5, Steven 5/5
    group 2: Tanvi 4/5, Haotian 4/5
  - Week 2– group 1: Ky +4=9/10, Tanvi +4=8/10, Haotian +4=8/10
    group 2: John +3=8/10, Steven +3=8/10
  - It'll even out as the weeks go on since you can get bonus points for creativity on top of the 1 to 5 scale.

- **Groups cannot be the same each week**

Before the next week:

- Leads will open pull requests and merge selected group scene into the team branch and update the team's baseplate for the next round of work.

Git Dictionary	Week 1	GitHub	Slack Intro
Unity Tutorial	MRTK3 Docs	Unity Basics	MRTK3 Sample Repo

Git Dictionary

xGit Dictionary

- Pushing to GitHub (from working branch) for the first time:
  git push --set-upstream origin <branch-name>
- Pushing to GitHub subsequent times (from working branch):
  git push
- **Switching branches to a local branch:
  git checkout <branch_name> or git switch <branch_name>
- **Switching branches to a remote branch you don't have locally:
  git checkout -b <branch_name> origin/<branch_name>
- See what branches you have locally:
  git branch -l
- See what branches you have remotely:
  git branch -r
- See all branches:
  git branch -a
- Pulling from a remote branch into your branch:
  git pull origin/<branch_name>
- Deleting a branch (LOCALLY ONLY):
  git branch -d <branch_name>
- Save uncommitted changes temporarily:
  git stash
- Apply stashed changes:
  git stash apply
- Rebase your branch on another branch:
  git rebase <base_branch>
- Reset branch to a previous commit (soft and hard):
  git reset --soft <commit>
  git reset --hard <commit>
- Fetch remote updates:
  git fetch origin
- See condensed commit history:
  git reflog

- Initializing a new Git repository:
  git init
- Cloning a remote repository:
  git clone <repository-url>
- Staging files for commit:
  git add <file> or git add .
- Committing staged changes:
  git commit -m "Your commit message"
- Viewing commit history:
  git log
- Viewing repository status:
  git status

Week 1

Week 1: Main Menu & Screen Flow

Download Unity:

- Unity Tutorial
- Unity Basics
- Abbreviations & Definitions
- Coding Best Practices

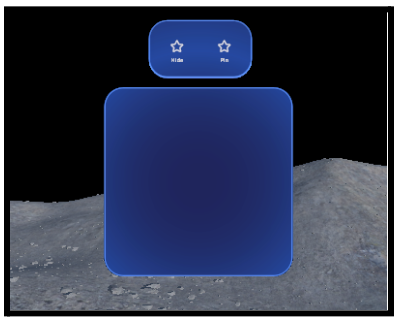
Open the ORA Repo and begin!

1. Leads will checkout their `ora-<?>` branch from main locally
2. Leads will push this branch to the remote repo
3. Members will fetch updates and switch to their `ora-<?>` team number branch
4. Members will checkout their `ora-<?>-<name>` branches from this team branch and begin working – this is the branch members will work individually all month
5. Go to Assets > CLAWS > Ora > Week-1 > Ora-<?> > Base-Task > Member-<?> and rename your folder and scene to your name + last initial

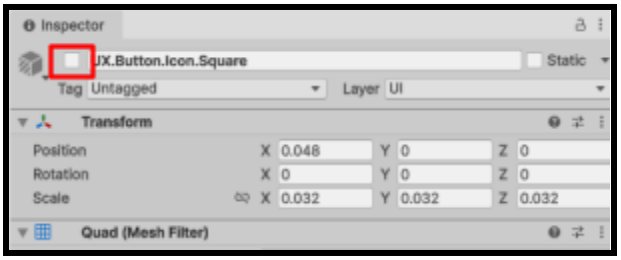
Base Task: Create a button menu to toggle between two screens

Objective: Create 2 backplates using SetActive and a public function to control the screens in a C# script and a menu with 2 buttons toggling them on and off

1. Drag 2 backplates and a button group prefab from the project window into the hierarchy view under the appropriate GameObject under Controller
    a. Assets > CLAWS > Prefabs
    b. 💡 These are custom prefabs with eyegaze control, you can find MRTK3 assets in the package folder under UX Components (non-canvas)
2. Edit the button group to only have 2 buttons and edit the shape accordingly:
    a. 💡Children inherent location from their parent so be mindful of positions in the inspector and make sure its always at (0, 0, 0)

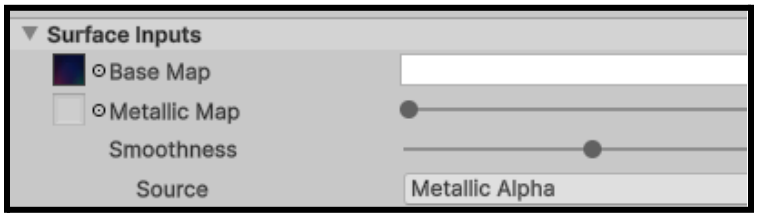


3. Creativity Points +1: Add your own button icons & text!
    a. Expand out the buttons child ‘CompressableButtonVisuals’ > IconAndText > char, sprite, or quad
    b. Feel free to edit any of the GameObjects by toggling / untoggling them in the inspector using scale, rotation text,, and materials to do so



        1. 💡The 3 dots allow you to create a variant of the material to then edit it

    c. Change the content of a material by adding a png/jpeg into the ‘Materials’ folder and dragging into the base map:



4. Create a script in the ‘Scripts’ folder called ‘MainMenu’
5. Add two GameObjects for your screens
    a. 💡Always Serialize your frontend and important GameObjects so you can see them in the inspector

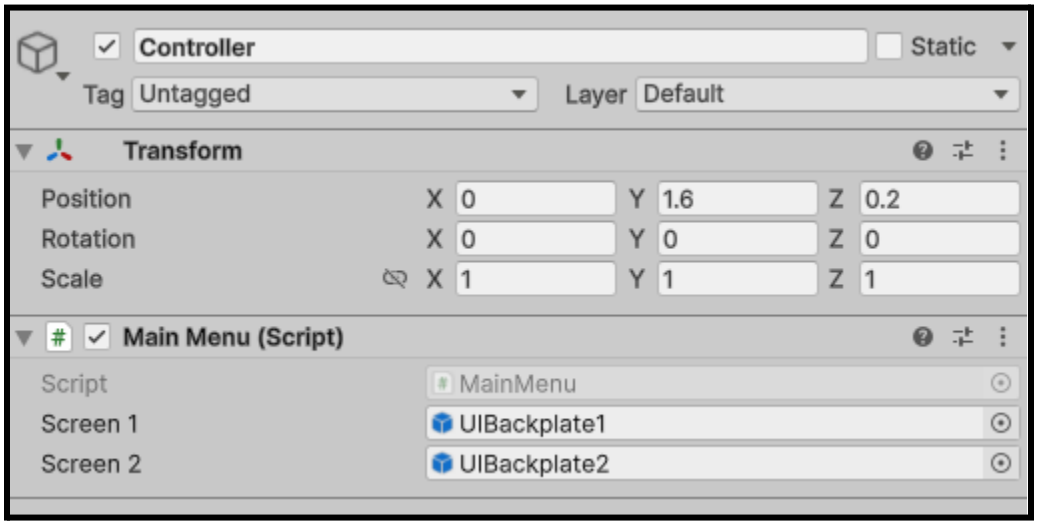
```
using UnityEngine;

0 references
public class MainMenu : MonoBehaviour
{
    3 references
    [SerializeField] private GameObject screen1;
    3 references
    [SerializeField] private GameObject screen2;
```

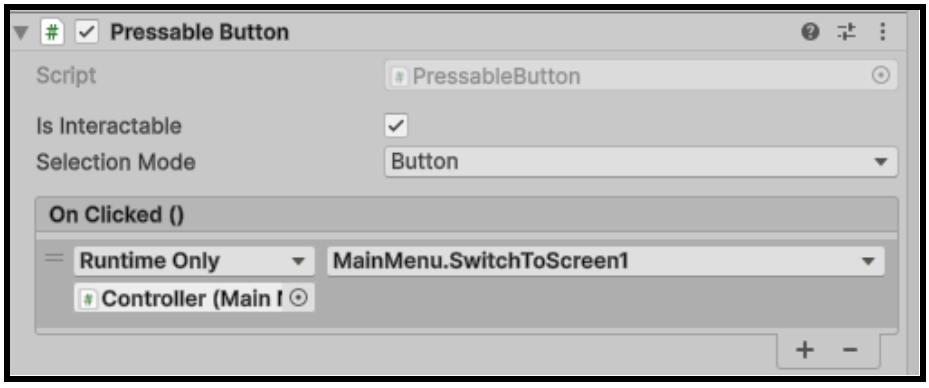
6. Set the 1st screen to true using SetActive in the Start function

   a. 💡The Start function is called automatically by Unity once, on the first frame when a script is enabled – used for initialization

7. Create two public void functions and remove the Update function
   a. Have one turn on screen 1 and one turn on screen 2
   b. 💡The Update function is called once per frame as long as the object and script are enabled – we don't use this function as much because of the performance costs
8. Assign the script to the Controller GameObject holding the menu children

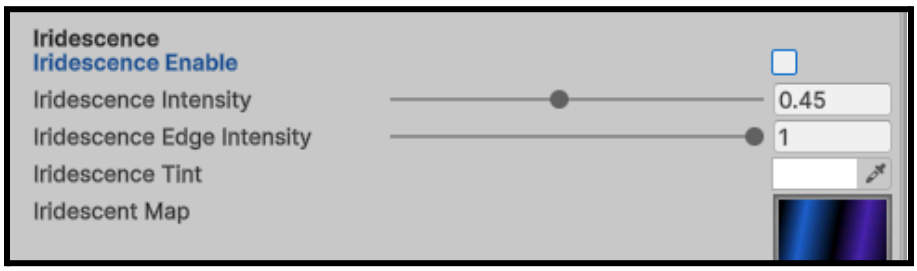


9. Go to the button menu GameObject and select a button child GameObject
10. Add a click event and drag the Controller GameObject into the empty slot and select the MainMenu script and function



11. Repeat the same for button 2, but selecting the correct function from the MainMenu scripts list
12. Creativity Points +1: After learning how to create material variants, choose a backplate to edit the child slate material of to differentiate it from the other

a. You can turn off the iridescence



b. Play around with the material settings in the inspector!

13. You can now play your scene!!
    a. 💡Be sure to reference the Unity Tutorial for how to navigate game view
14. You'll notice that when you use your right click to look around, that the button responds to eyegaze and won't let you select the button unless you're looking at it
    a. This is because of how our physical button works with eyegaze input!
    b. The onClick event gets registered as a left-click from a mouse when the users eyes are looking at the correct button
    c. 💡Reference the Eye Gaze Highlight doc for more info
    d. So to choose the correct screen make sure the frontplate highlight is lit up and then you can use your mouse to left-click or the spacebar and hand poke
        i. Difficulty Point +2: Add a gaze-pinch interactor type with the MRTK events in the inspector settings on the buttons so you can use the eyegaze feature with a spacebar / left-click pinch

****You have now finished the Base Task, push your code on your member branch****

Switching from Individual to Group work:

1. If you're the 1st one to finish the basetask, checkout a new branch from your member branch called `ora-<?>-group-1`
    a. For groups 2 and 3 follow the same instructions
2. Copy over your member folder from the base task with the scene, scripts folder, materials folder, and readme
    a. 💡You'll find a new readme in the group folder as well
3. Stage your changes and push to the remote repo on the group branch
4. The people in your group will need to pull that group branch from the remote to their local repos and switch to it
5. You will begin working concurrently in that branch but in your own member folder Assets > CLAWS > ORA > Week-1> ora-<?> > Group-Task > <name>

6. If you copied over your member folder, simply work in that folder
7. If you pulled the branch, copy the scene in the above members folder and paste it in a member-<?> folder in the group folder

Group Task | 1-5pts | 0pts | Part 1: Create a button menu to toggle between two screens
Objective: Create 2 backplates using the MRTK3 helper script ToggleCollection to control the screens in a C# script and a menu with 2 buttons toggling them on and off

1. In your new scene in Assets > CLAWS > ORA > ora-<?> > Group-Task > <name> select the Controller GameObject and add a component by searching for ToggleCollection.cs
2. Add two element toggles and drag the button GameObjects to the inspector window
3. Use the On Toggle Selected event to control the screen flow using the Controller GameObject like the base task, but with only one function for switching screens
    a. 💡You'll have to edit the MainMenu script – VS Code Live Share
4. Play the scene!
5. Change the README.md in the root folder to the names of your group members
6. Push your changes to GitHub on your group branches
7. Show the screens to your ORA lead to continue on to part 2 of the group task.

Group Task | 1-5pt | 1-3pts | Part 2: Create a welcome screen for your project
Objective: Create an opening screen that has your team name, a photo of your group, a name input box, user avatar colors, and id options

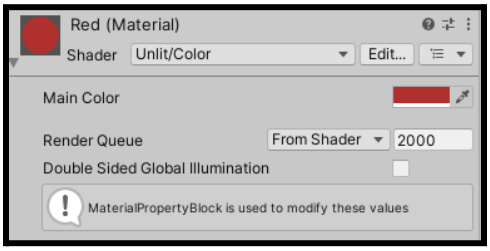
1. In the same scene as before in your member folders in the group folder, create a child GameObject of the Controller called welcomeScreen
2. Add a backplate prefab like in the base task
    a. 💡Use one with text on it!!
3. Add your team name and member names to the slate using the text mesh pros (TMPs) that are provided through the prefab, if not there could be errors
4. Add a quad or sprite to add a material with a photo of your team

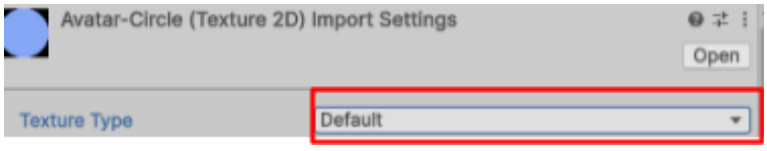
   a. Make the quad a child of the backplate so it inherits its positon
5. Add two buttons to the backplate with the text of 1 and the text of 2 to indicate which id the user is.
6. Create 2 new functions as well to assign the the 1 option or 2 option to the Astronaut user class (check out the Astronaut folder in Unity for more info)

```
0 references
public void astronautID_1()
{
    AstronautInstance.User.id = 1;
}

0 references
public void astronautID_2()
{
    AstronautInstance.User.id = 2;
}
```

7. Assign these functions in the inspector to the newly created buttons as previously
8. Create avatar colors using as many buttons as you would like and changing the sprites materials to solid colors (in the IconsAndText child)
9. A circle sprite is provided to you in the Group-Task folder and the material can be found by searching for unlit/color (feel free to upload your own png)
   a. 💡The unlit means that the shader does not respond to light





   b. 💡To change a png into a sprite select default -> sprite (2D and UI)
10. Create functions like the id buttons to log the color to the AstronautInstance.User.color variable
11. Add them to the buttons GameObjects in the inspector window like before

12. Create a button to open a keyboard on the Hololens as a child of the backplate
13. On the Controller GameObject add a component by searching for the KeyboardInput.cs script
14. Add a TMP GameObject (copy one from other sections) as a name field and drag the GameObject into the Controller inspector window and attach to the Keyboard Object
15. In your MainMenu script, create a new function called openMenu where it opens the menu screen previously made using SetActives of ONLY two GameObjects
16. Create a done button that you can attach your OpenMenu() function to

17. Play your scene! You should see the AstronautInstance script update as you select in the game view in the inspector view of the controller with the script
18. Show your lead when you're done or at the end of Sunday's meeting
19. Change the README.md in the root folder to the names of your group members
20. Push your changes to GitHub on your group branches, there shouldn't be merge conflicts, but if so alert someone in leadership and they can help
    a. 💡Make sure youre always working in your folder or else the files you touch may be edited by someone else

Next Steps

1. Leads will review each group branch and individual branch after the Sunday meeting and tally points for each group
    a. If you need more time to work after Sunday let your lead know
2. If you have any extenuating circumstances please let your team lead and subteam lead know so we can adjust expectations for onboarding