# Tips (Unity Development)

* Do user testing as soon as possible to finalize the scripts to avoid more works in the future, especially during the retrospective.
* Player-computer interactions are really important and should be considered in the early phase.
* Search for free assets from the Unity Asset Store.
* Use terrain instead of plane for the landscapes of the scene.
* Trees can be added.
* The texture can be changed.
* The level of the landscape can be adjusted.
* Discuss the outline of the level or the ideas with your Unity Developers before developing the actual level in Unity.
* Check with your Unity Developers if the level going to be developed is feasible.
* Name and sort the objects used.
* Name the objects before duplicating them to avoid further inconvenience on renaming the duplicated objects.
* Sort and put the objects for the same model under a parent object. For example, all the objects used to make a chair model under the parent “Chair.”
* Make prefabs package after designing the models so that it could be exported to other Unity projects.
* When using cylinder objects, make sure to change the capsule collider to mesh collider.
* Unity does not allow customization of geometry shapes such as making a hole, so look up for alternatives 3D modelling programs like Autodesk Maya or Blender. Otherwise, scripts are needed for those types of customization.

# Tips (VR Development)

* In VR, scale can be strange to design around without a Vive setup on hand. It is reasonable to assume that an average person is anywhere from around 1-1.5 units tall in Unity.
* Audio cues are 100% necessary for any level of player for level awareness. Prior to actual development, find free audio assets that all match a general theme and design around each component of the level to have some sort of sound.
* Providing the player with information in VR is a multi-faceted problem. Attaching a menu floating to the player’s face is disorienting, and there is no way to guarantee that a player will notice a still text box in the level. We decided to implement a still text window in level, but we added more feasible interactions with the window. Our window makes noise on spawning, the player must look at and use the controller to cycle through messages, and the windows can optionally be animated to move. We also ensured that the level design aided visibility, making windows spawn where the player must go in a level.
* Moving objects, like our gate in Level 3, can be very disorienting in VR. Be careful when adding in dynamically moving objects with many visual components to it.
* Developing without a headset is reasonable for level design, but impossible for unit testing and script design. Ensure that access to a Vive or Rift setup is always available at every hour.
* Preventing the player from moving their head through a wall in VR must be handled. We simply added fast damage from head collisions.
* The pause menu, if not bonded to the player’s controller, must be visible and the player must know that it exists if it is spawned. We implemented a distance check when spawning the pause menu to ensure that the player can’t spawn the pause menu outside of a wall where they can’t interact with it.
* When a scene loads, there is no guarantee that the player will be where you want them to be. Room sizes are variable, so if a level’s starting room is restricted to a certain size, the scene needs to ensure that the player doesn’t spawn in a wall. In our hub level, we decided to disable everything in the level until the player walks into a collider located in the middle of the room, which is tied to the exact middle of the actual play area.

# **GitHub Overview**

## Semester 1

Operation: Git! Hazy.

Status: Disaster.

Initial idea was to use an already tested a .gitignore file in conjunction with git LFS for binary/media files. Git LFS is a git extension that replaces files in your repo that you specify in a .gitattributes file, with pointers. The pointers reference the files somewhere else, like a remote server (in our case github.com hosted them, just like the regular repo files).

When everything is working correctly, a git checkout <branchname> is performed and git LFS will see the pointers and fetch down the specified files. With a roughly 60 MB repo and no errors a git checkout <branchname> operation would take around 7 minutes to fetch the files down from the pointer information (the high bandwidth of the university made no difference). Which was, in my opinion, a completely unacceptable operating time. In our repo, during the already excruciatingly long checkout process, git LFS invariably errored out attempting to download files — we never found out exactly what the issue was with the broken pointers; but the git LFS errors didn’t stop Unity from working so we worked through it.

Our other issue was with the Unity file structure in conjunction with the .gitignore. Our root git folder structure was:

/project-name/

.gitignore

.gitattributes

readme.md

Where all the unity related files were in the /project-name/ folder. This is not how the .gitignore we were using was pathed to be used. But since none of us had used unity before we didn’t know all the files we were committing were supposed to be excluded. Still, since commits were primarily done from the one workstation in Shorr, the repo was thought to be in a primarily working state.

The merge results from a merge around mid-first-semester put the repo in a non-working state that thankfully a member of our team (Brian) was able to negotiate back into working order. But we didn’t realize the .gitignore issues would still persist for future merges. Near the end of the first semester a merge was done from another members branch (Isaac) that worked fine from the git perspective; however from the unity perspective it was bricked. We finished out the semester from flash drive backups.

## Semester 2

Operation: Git! Focused.

Status: Success.

Step 1: Fix the Repo.

From our pristine flash drive backup diligently guarded over the break by the ever vigilant Dat. An orphaned git node was created from our repo and the flash drive files were tracked from it. The orphan was to make sure all the bad commit history of the unity files did not continue to cause us problems for the duration of the project. This however broke the commit hashes for release in github, but the breaking change was well worth it.

All the Unity project files were moved out of the /project-name/ folder into the root because we discovered that is what the .gitignore file was expecting. This fixed our commit issues with the Unity specific IDE files that should not have been in the repo to begin with. Success! We additionally thought to flag the Unity library directory after learning Unity would rebuild it from scratch if it was not present when opening a project. But after fixing the .gitignore location we realized it was already ignored.

We decided to table Git LFS (you can affectively disable it by removing the .gitattributes file from the repo). Our project was not heavy with assets ( that we wanted to track ), so we added a special folder /Assets/AssetPacks/ that could be placed into the Unity project to browse asset packs, but not inflate the git repo to ridiculous sizes. With the team enforced rule that if an asset was used from the asset pack, it was moved to a tracked folder /Assets/TrackedAssetPacks/. This removed the pointer git LFS issues, and reduced our repo size considerably.

Since the git issues were so time consuming in the beginning of the semester. It was decided to task a team member (Isaac) to effectively become a merge gate-keeper to handle all merges from team members and resolve any conflicts that came up. Having each team member who was working on tracked files work from their own fork ( the original idea ), turned out to not be necessary with the system of one merge master.

# Tips (GIT)

* We found that segmenting our workflow rather than segmenting our project.
* Not using entire asset packs in the tracked assets
* Using two or three Git branches
  + A master branch for releases
  + A development branch for normal changes
  + Sometimes create a branch for non-working builds and work that may not be kept
* Don’t modify any library files directly, make a copy then modify
* Use the GitHub provided Unity .gitignore
* Consider using Unity Smart Merge