Bird View Navigation System Document

Mengyu Chen
mengyuchenmat@gmail.com

2019

1. System Components Introduction

The Bird View Navigation System has many different class components, each of which is responsible for its unique functionality. Based on the importance of these components, they are divided into two types: Static Classes(Non-Destroyable) and Level-Specific Classes.

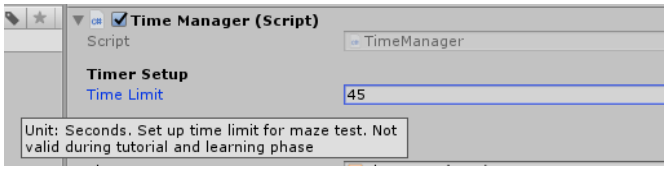
Static Classes are instantiated at the very beginning of the program, and will **persist during the entire experiment**, regardless of what mode the task is in and which particular navigation level has been loaded. They are unique (there is only one stance of each) and universal across the scenes that the functions kept in them can be called any time by any other classes because of their static nature. Currently, there are 10 static components:

- **Navi Manager** - the central manager object that decides the general logic of the experiment such as what to do next, what mode we are in, how many map levels left, etc. It contains the information about the general status of navigation tasks, and have public properties that can be retrieved for other objects. It makes calls on other static objects to make sure they are doing their jobs.
- **Level Launcher** - an asynchronous level loading manager that loads and unloads different navigation levels based on data provided by Navi Manager.
- **Arrow Manager** - this manager takes care of the instruction avatar that guides the user what to do between different levels, and indicates the next starting point that the user needs to move to.
- **Map Manager** - this manager controls the display of bird view map at every level. It manages the rendering of a second Camera which has a bird view.
- **Track Player** - a data recording manager that tracks the user position, rotation and other meta-data information, and writes into a .txt and .csv file for data analysis
- **UI Manager** - this manager takes care of the loading page UI objects and sends meta-data to Track Player
- **Time Manager** - this manager sets up a time limit for the navigation task and can be called by the Navi Manager to start counting down. Once the user is out of time, it will let Navi Manager to move onto the next phase.
- **Interaction Manager** - Interaction Manager is responsible for all user input interactions, mainly the VR controllers. Anything depends on controller input can get data from interaction manager.
- **Camera Marker** - a static instance placed on cameras so that other components can quickly find out which camera is active (desktop camera or VR camera) and get the camera transform data.
- **Fade Manager** - takes care of the fade transition effect.

Level Specific Classes are placed in each level and are instantiated only when the specific level has been loaded. They are usually environmental user collision detectors that need to be placed on many different objects. Currently, there are only a few of them:

- **Arrival Collision Check** - placed on a destination object with trigger type collider. Once user enters the collision trigger or use controller to click on the object it is attached to, it will call the Navi Manager to complete this level.
- **Arrow Collision Check** - placed on the instruction arrow avatar that checks if the user reaches the position required by the instruction. It is used on the default arrow avatar and also the learning phase arrows.

  ***Note: Every component are documented with explanations in the Unity Editor UI. You can hover the mouse above the components and read the explanation of each setup properties. (Example below. Tips will show upon mouse hover)**



2. Scene Structure Design

The Navigation System is designed based on an idea to be modularized and extensible. This scene structure is made as:

Total Scenes *= Launcher Scene* **0** *+ Tutorial Scene* ***1*** *+ Learning Scene* ***2*** *+ Test Levels ** ***N***

where Launcher scene is set to have index 0 in the Unity Build settings, VR Tutorial to be 1, Learning to be 2, and all the rest Test Levels are added after them.

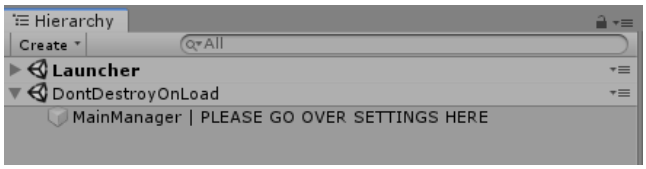
- **Launcher Scene**: the only starting scene of the program. It contains all the necessary infrastructure of the program, including VR/Desktop Camera, Map View Camera, Static Manager Objects, UI Panels, Event System, Empty Environment and Basic Lightings. Same as the static classes, this scene is to be loaded at the very beginning of the program and will **persist during the entire experiment**. All other scenes, including tutorial, learning and test levels will be loaded as Additive Scene on top of this Launcher scene. That is to say, one cannot enter other scenes without getting into the Launcher Scene because the camera objects are stored only in the Launcher scene.
- **Tutorial Scene:** a simple tutorial scene allows the user to get familiar with VR interface, and check if the height calibration and the VR interactions are working properly or not.

- **Learning Scene:** a scene that can be used as a pre-test guidance level for the user to learn the process of the experiment. Specific instructions are made along with a NaviTutorialManager to let the user have some test map-based navigation.
- **Test Scenes:** each test scene is stored as a scene to maximize customizability. That is, each scene can have its own spatial structure, object, and basic logic scripts. The build index for test levels starts from 3, so the level indices of test levels are ranged at:

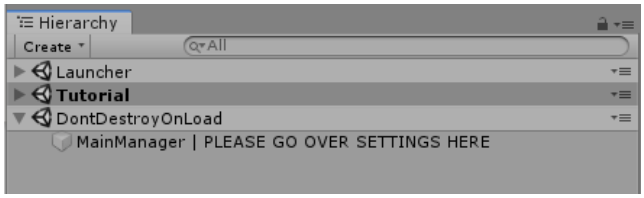
  Inclusive | | Exclusive

  [3, N + 3)

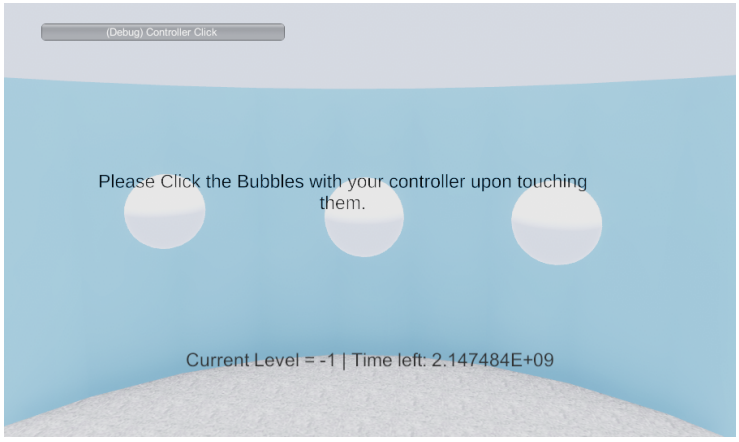
  Where N is the total number of test levels.
- **Scene Launch Order and Transition:** The program will start with the Launcher Scene, and then based on mode selection and user interaction, Navi Manager will choose which additive scene (Tutorial, Learning, or one of the Test Levels) to be loaded on top of Launcher Scene. (Image below shows the Launcher scene loaded upon running)



- Typically, it will start with the Tutorial scene. This scene will be loaded as an Additive Scene beneath the Launcher Scene. (See image below)



- The additive scene brings in the environment contents (such as ring wall, instruction, target and floor) , and the Launcher scene provides the static functions and VR infrastructures that allow the user to do the test.



- Since the navigation scenes are additive, once the task is finished, the additive scene will be removed and destroyed. The user will get back to the empty space in Launcher Scene, with **Instruction Avatar** guiding the user to the next phase.

- This loading and unloading additive scenes will iterate until the user has entered and finished all the required scenes.

3. Navigation Scene Components and Setup

***Example Navi Scene is also provided as Assets/Scenes/Level1, 2, 3**

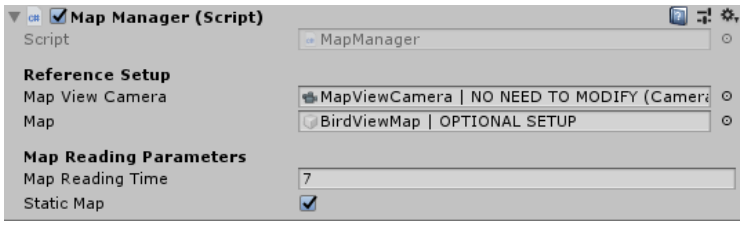
The Tutorial, Learning, and Test Scenes are all considered additive scenes since they will be dynamically loaded and unloaded during the experiment. In order for an additive scene to be properly functioning, several types of scene objects have been provided (**all prepared in Assets/Prefabs folder**):

- **Target**: target objects should be tagged as “Target”. It must be set to the layer of “RenderMap” so that it won’t be seen by the VR camera but only rendered by the Map View Camera on the map.
- **Starting Instruction**: Not required, but can be placed at where the user will start the level. A text panel is prepared and messages can be written on it.
- **Ring Wall**: An object that defines the size of the space. Its XZ scale is the radius in meter of how big the circle will be in VR.
- **Floor**: the floor should be tagged as “Floor”
- **Lightings**: lightings are optional but good to have. Currently, the example scene is using baked lighting for better rendering quality.
- **Finish Confirmation**: A script object to detect if user has clicked the trigger as a confirmation of finding the target. If so, it will end the level then.

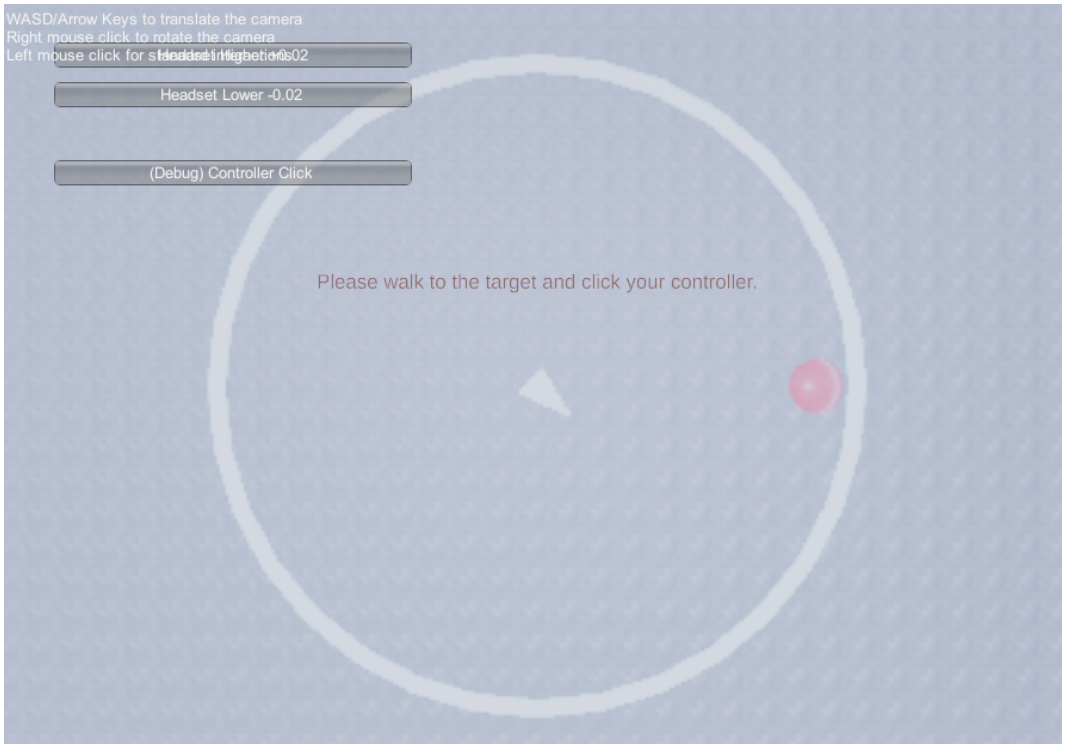
4. Bird View Map, Map Manager and Usage of Layers

The way how the BirdViewMap works is that:

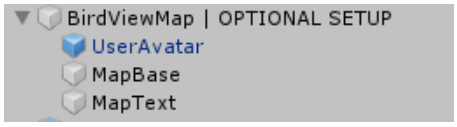
- We have a Map View Camera (a secondary camera that is different from the VR camera, see reference setup in MapManager below)



- This map view Camera is located on the top of everything and faces down. It captures the whole bird view scene.



- Then the Camera renders into a RenderTexture that is attached to a quad object (MapBase) in front of the user's VR camera.



- When the map is on, user will see what the MapViewCamera sees in their VR view
- The options "Map Reading Time" decides how many seconds the user will see the map at the beginning of each test level.

- Static Map: if static map is ticked, then while reading the map, user's rotation won't be reflected on the map. Otherwise, user can move their body and the user avatar on the map will also immediately move on the map
- When reading the map, user will notice that the target is only visible on the map, not in VR. That is because, the Target object is set to be in the Layer of "RenderMap". For the Map View Camera, it will **ONLY render** objects that are set to "RenderMap" and "RenderBoth". For the VR Camera, it will render **everything except the "RenderMap" layer**. A layer named "RenderVR" was also created for better organization of object visibility. (Below is the setting for UserAvatar object, the triangular shape in the map view but not visible in the VR view)

5. VR Calibration and Height Adjustment

A **Height Adjuster** component is made for manual and automatic height adjustment.

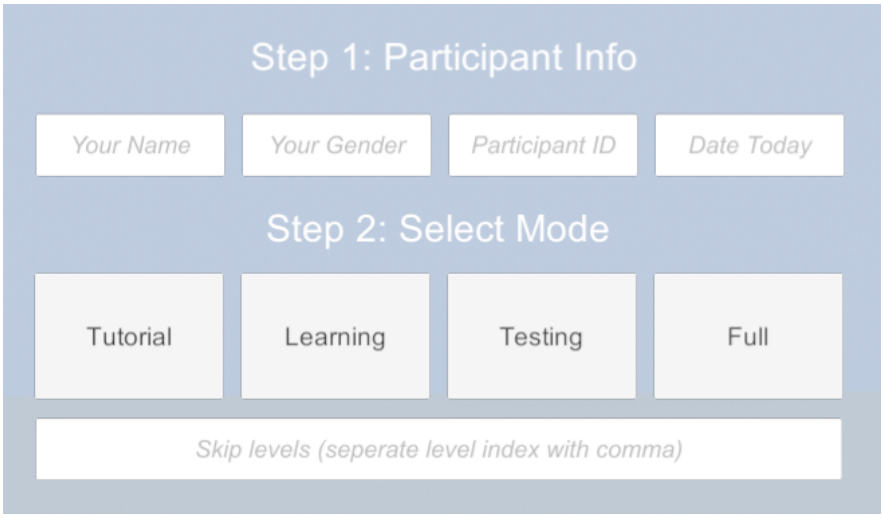
Manual Adjustment - During the experiment, one can any time press Up or Down arrow on the keyboard to adjust the user height in VR headset. There are UI buttons after running the program which also allows for height adjustment.

Automatic Adjustment - an offset value can be automatically assigned to adjust the height of VR view at the beginning of the program.

*__Note:__ For HTC Vive system, there is no need for Auto Adjustment. This feature is for Windows Mixed Reality system when calibration is not accurate. Based on current Samsung Odyssey calibration, this value is estimated at -0.9f, and can be changed in the GUI, and may not be necessary with different calibration method.

6. UI Management and Mode Selection

The UI Manager is made to collect participant information as well as the global settings of the navigation tasks. Upon launching the program, a UI component will pop out and collect user information. One can select the levels and mode to load the corresponding level contents. (Image below is the UI element after launching the program.)



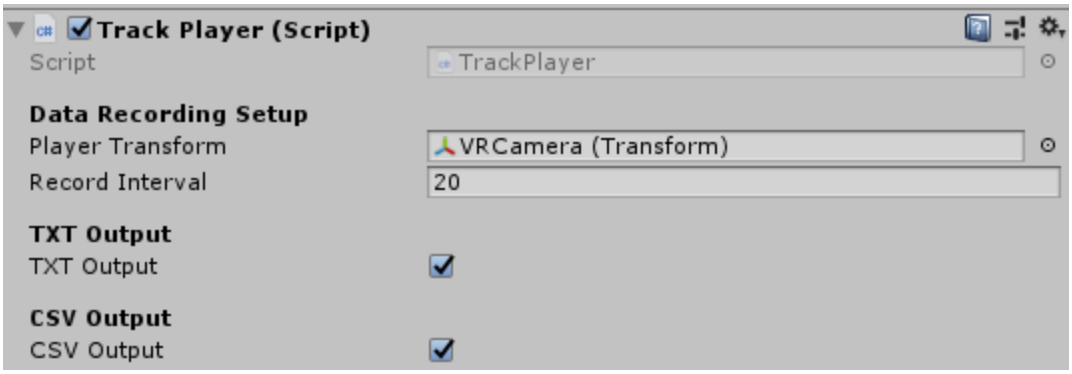
Available Navigation Modes:

- **Tutorial Mode**: it will go directly into the tutorial scene and quit upon finish.
- **Learning Mode**: it will go directly into the Learning scene and quit upon finish.
- **Testing Mode**: it will go directly into the Navigation Testing scenes. If there are many different test scenes, it will run the Non-Repetitive random algorithm to go through every navigation tests. This mode is usually used when tutorial and learning are not necessary to the user.
- **Full Mode**: it will start from tutorial scene, then move to learning scene, and randomly go through all the navigation test scenes.
- **Skip Levels**: in case the test couldn't finish property through all the levels, and one wants to skip certain levels to avoid repetition, simply put the level number separated by commas in the blank beneath.
    - For example, we have Level 1, 2, 3, 4, 5 and we would like to skip Level 2, Level 3, Level 5, then simply write **2, 3, 5** in the blank with commas in between and it will skip these levels, running only Level 1 and 4.

The secondary function of UI manager is to control the runtime UI elements to change the global settings and run some debug functions. Currently, there will be Headset height adjustment and a Simulator controller click in Debug Mode appearing as screen UI.

7. Data Recording Format

Track Player component captures the position and rotation of a given object



The Player object can be set in the editor via drag and drop (by default, it is the VR camera from SteamVR player object, which contains the VR headset position and rotation). **Record interval** refers to how many frames between each data recording activity. For example, if record interval is 20, then every 20 frames (about every 400 milliseconds or 0.4 second) there will be a line of data recording to the file, since typically the program is running at 50 frames / second. This interval can go up to 1, which means recording the data every frame (about every 20 millisecond).

Track Player component currently can output two file formats, one in .txt format and the other in .csv format. The text file stores data in a linear way and contains more debug or custom information. The CSV file only records necessary information for specific data analysis purpose and is formatted with columns and rows. Specific data recording format can be adjusted in Track Player class.

File location and name: files will be stored at ProjectDirectory/participantID_name_date.txt / .csv.

8. Tutorial A: Level Building Walkthrough

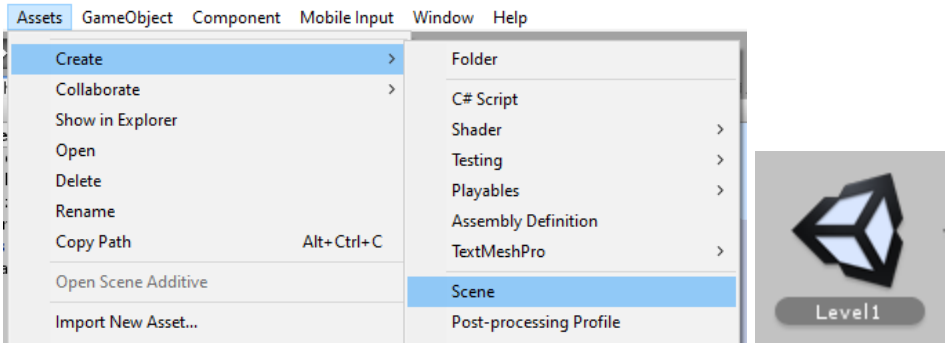
For this tutorial, it will walk you through how to create your custom Level.
*Make sure you pull the latest version of Bird View Navigation Project.

Idea of Level Building:

1. It has a defined area.
2. It has a target.

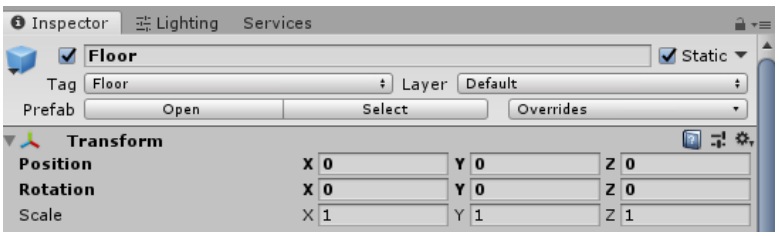
Steps:

a. Navigate to project folder /Assets/Scenes, on the menu use Assets/Create/Scene to create a new scene. Give it a name like “Level 1” or anything that you can easily identify its number. Open this scene. There will be only two objects in the Hierarchy list: 1) Main Camera object and 2) Direct Lighting in the scene.



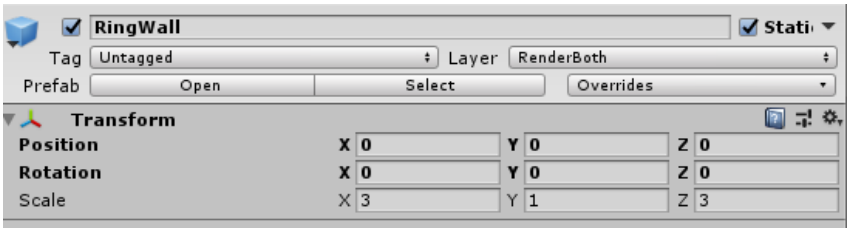
b. Delete the 1) Main Camera, since we will NOT NEED CAMERA in any additive scene. Instead, we will use the VR Camera (Player object) that has been already set up in the Launcher scene, which will be the base scene that provides many infrastructural objects for any additive scenes.

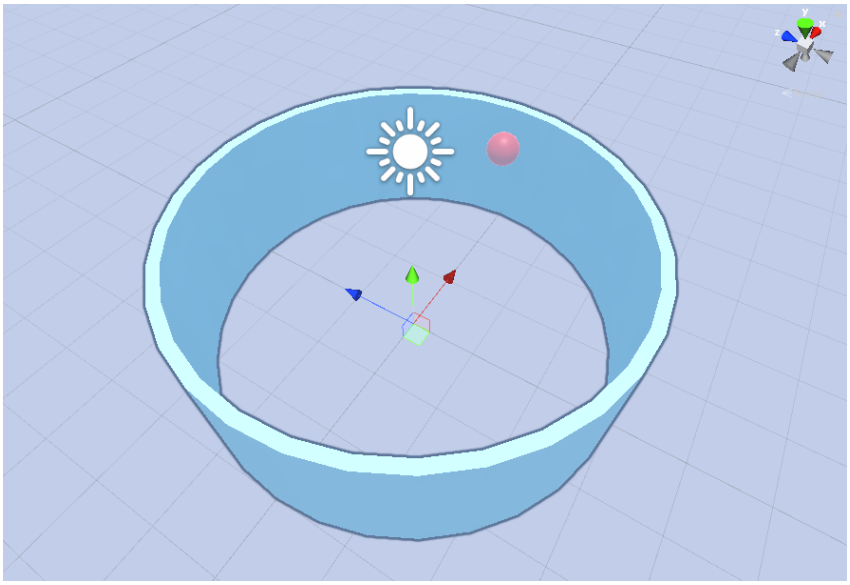
c. Go to folder, /Assets/Prefabs/LevelObjects, drag and drop “Floor” object into the Hierarchy list. Make sure Floor object is located at (0,0,0). You can change the scale of Floor object, depending on the space size you need. Note: The Floor object has a tag name as ‘Floor’.



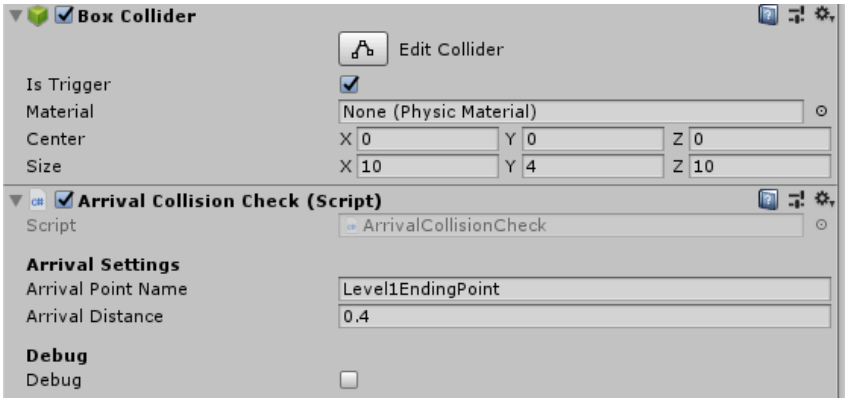
d. Also put a Ring Wall object into the scene. Adjust the scale of a ring wall to define and formulate an area of your walking area. Note that in Unity **1 unit = 1 meter** in real life. You can calculate how large the VR walking space you need.

Note: The Ring Wall is set to be on Layer ‘RenderBoth’, which will show in both VR Camera and Map View Camera.
(Image below shows the Ring Wall)



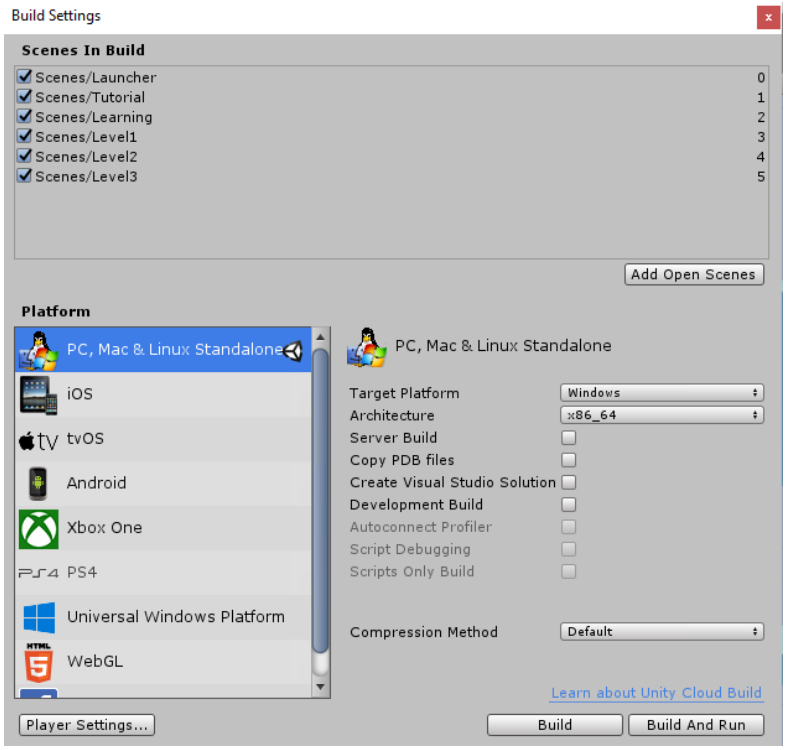


e. Place a Finish Confirmation Object into the scene. It is a script object that contains an “Arrival Collision Check”. Make sure it has a Box Collider with ‘Is Trigger’ option ticked.



f. Place a Target object in the space at where you want to set as the destination you would like the user to reach. Note that any 3D object can be set as a target, as long as its Tag is set to ‘Target’ and Layer set to ‘Render Map’. Reference the example prefab Target in the Assets/Prefabs folder.

g. Make sure you are in the 'Level1' scene you just made. On the menu, click **File -> Build Settings**. Assume that you use the provided Tutorial and Learning scenes, you should see **'Scenes in Build'** section where Scenes/Launcher, Scenes/Tutorial, and Scenes/Learning listed with index number 0, 1, 2 on the right side. Then click Add Open Scenes. (Image below shows that the scene you have just created has been added to the Build Settings, and has been assigned an index 3)



h. Repeat the steps above to create as many levels as you want. You can copy paste everything in the Level1 to a new scene and only modify the position of the target.

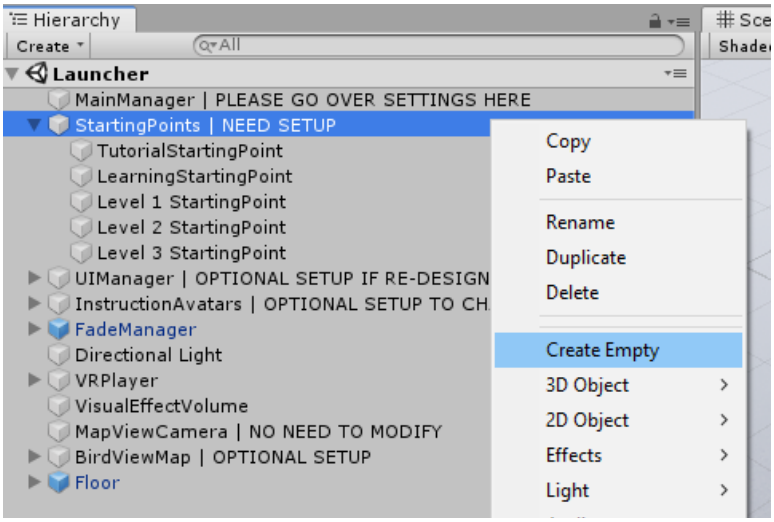
i. When you have all your custom levels ready, navigate to folder Assets/Scenes, double click 'Launcher' scene. Proceed to Tutorial B.

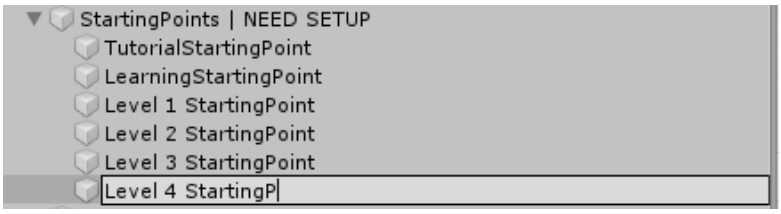
9. Tutorial B: Starting Points Setup Walkthrough

For this tutorial, you should set up the "Starting Points" for each level (the place where user will stand when they start each level). These starting points can be either always at (0,0,0) the origin point or at different custom positions in each level.

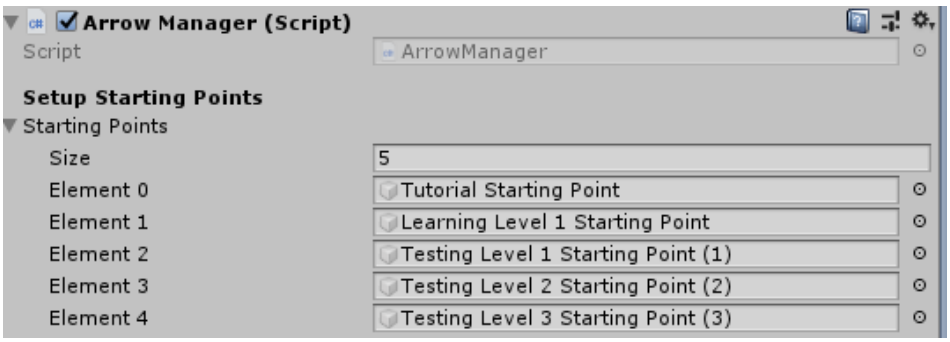
a. In Launcher scene's Hierarchy list, you should see the Starting Points object.

b. Right click Starting Points, choose Create Empty. After that, a new Game Object will appear. Rename to Level1 Starting Point (or anything you would like)





c. Move its position in the scene view to where you want the user to stand when they start the Level 1 level.
    i. Suggestion: since Level 1 is another scene outside of Launcher scene, you can drag and drop the Level 1 scene into Launcher scene's Hierarchy list, so that you can see Level 1 and Launcher scene at the same time.
    ii. Then you can move the Level1 Starting Point around with a reference to the Ring Wall.
d. Repeat creating starting point objects for each level you have created.
e. Click Main Manager object, on the inspector, there is an "Arrow Manager" component. Check the Starting Points array.

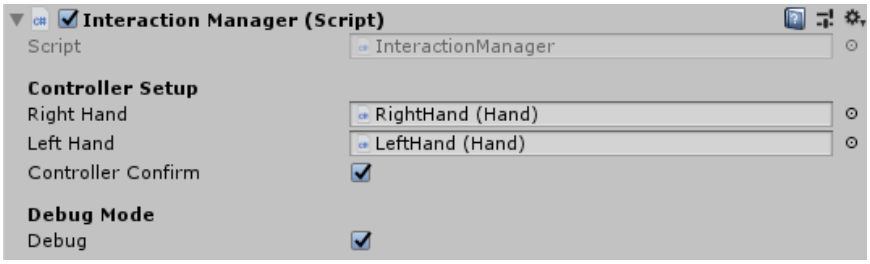


f. The size of Starting Points should be 2 + N, where N is the number of levels you have built. The first 2 starting points should be reserved for Tutorial scene and Learning scene.
g. Drag the newly created “Level N Starting Point” object to the elements blank in order.
h. You can refer to the example provided above, where Level 1 2 3’s starting points have been setup (in the example project, they are all set to (0,0,0) position).

10. Debug Mode in VR and Desktop

For the development purpose, two debug modes are provided: 1) VR debug mode and 2) Desktop debug mode.

1. **VR Debug Mode** - This mode provides a quick flying navigation without the need to walk in actual physical space. One can simply use the controller to fly around instead of walking in the space.
    a. Make sure the headset is connected properly. The setting of VR headset should be the same as the walking experiment.
    b. In the Launcher scene, find the “Main Manager” object, it has a component named as “Interaction Manager”.



    c. It has an option named Debug Mode. Make sure the Debug option is on.
    d. Run the project, then you should be able to use the **Grip button** on your controller to fly around. (Grip button may vary on different VR controllers).
        i. Note: Your flying direction is the pointing direction of your controller.
2. **Desktop Debug Mode** - This mode provides keyboard based navigation and mouse click for simply controller behavior simulation.
    a. Make sure your headset is **NOT CONNECTED**. By doing this, SteamVR will automatically use the Desktop Debug Camera. All other Camera related scripts will also use this Debug Camera.

b. In the Launcher scene, find the “NaviManager” object, it has a component named as “Interaction Manager”. Same as VR debug mode, **make sure the Debug option is ON.**
c. Use W A S D on keyboard to navigate. Right-Click and Hold on the mouse will let you rotate the camera view.
d. In Debug mode, an extra “(Debug) Controller Click” button will appear on the screen which you can click to simulate a controller click.
    i. For some interaction in the test, where you need to touch the object when clicking the controller, you can simply use keyboard to walk into the object as a “touch” interaction.