Documentation

Maze Task

An implicit measurement of trust

programmed by
Larissa Brübach
Informatik IX
Universität Würzburg
larissa.bruebach@uni-wuerzburg.de

Supervisor
Dr Alexandra Livia Georgescu
Department of Psychology at the Institute of Psychiatry, Psychology and Neuroscience
King's College London
alexandra.georgescu@kcl.ac.uk

Contents

1 Introduction

This document is the documentation of the maze task implementation by Larissa Brübach. The GitHub project can be found here: https://github.com/larissabruebach2020/AlexMaze. For access please contact Larissa Brübach or Alexandra Georgescu.

The experiment is based on the original maze taks by Joanna Hale [1].

[1] Hale, J., Payne, M. E., Taylor, K. M., Paoletti, D., & Hamilton, A. F. D. C. (2017). The virtual maze: A behavioural tool for measuring trust. The Quarterly Journal of Experimental Psychology, (just-accepted), 1-53.

2 General Information

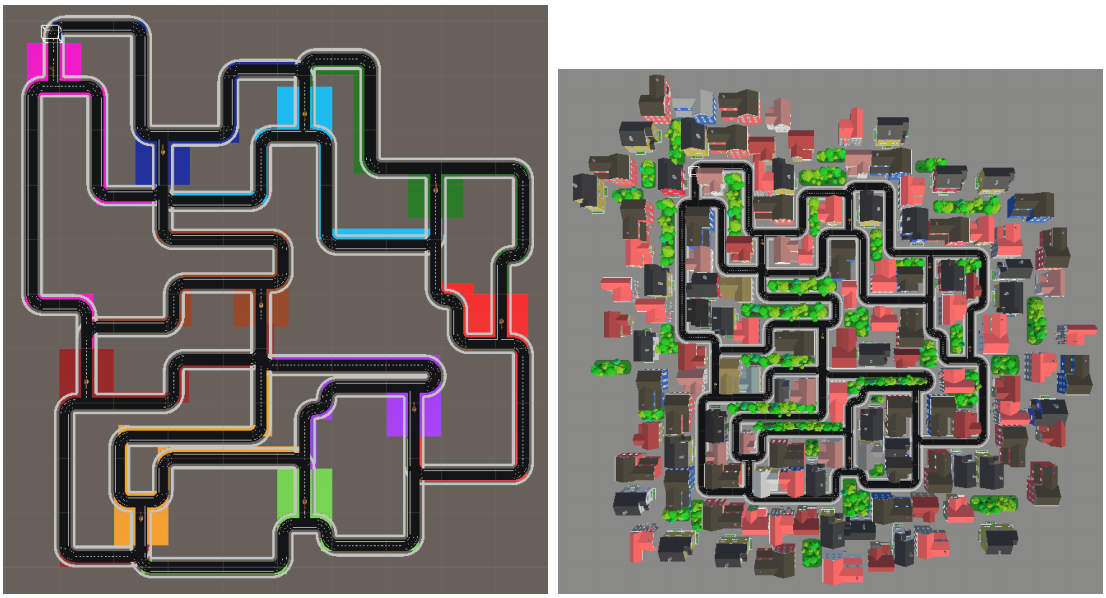
This application was implemented using Unity 2019.4.8f and the HTC Vive. It can also be used with the Oculus Rift S or in a desktop mode.

This is handled in the "Players" GameObject and the attached children and scripts.

For any questions regarding the implementation please contact Larissa Brübach.

3 General Structure

This maze consists of 10 rooms. They are all connected and there are no dead ends. Each room has an intersection that leads to two other rooms. They layout can be seen in figure 3.1. Each colored rectangle marks one room.



(a) Maze Layout (b) Maze Layout With Environment

Figure 3.1. Maze Layout

The different rooms are only loaded when they are needed. This is done to safe resources, as each room includes two animated characters. It also hides the other roads to make it harder for the participant to realize there is no exit.

4 Implementation Structure

This chapter describes the different C#-scripts used and the structure of the different scenes.

4.1 C#-scripts

4.1.1 Participant Name

The maze has a user interface (UI) that is visible only on the computer, not within in the glasses. It allows the tester to enter the subject ID (letters and numbers allowed) and the number of conditions (s)hewants the participant to go through (only numbers allowed). Note: to ensure the balancing of the conditions a multiple of 8 should be selected (8, 16, 24, etc.).

The tester has to enter those two valued and click "Start Experiment" in order for the participant to be able to start.

4.1.2 Scene Manager Script

This is the main script to manage the scenes and conditions.

At first it creates a list with all conditions. It is used to randomly select a new condition for each trial. When a condition was selected it is deleted from this list. When the list is empty, but the maximum number of trials (which is set in the editor) has not been reached, the list is refilled again.

When the application is first started it loads room number 1 and 2. Number 1 is always the first room. Number 2 is loaded because it leads to number 1 and gives the impression that room number 1 is connected. It also sets the agents position, rotation and audio file initially. This is afterwards done in "Load Next Room".

When the function "Start Trial" is called it randomly selects a condition for this room and stores the known logging values in "Maze Logging".

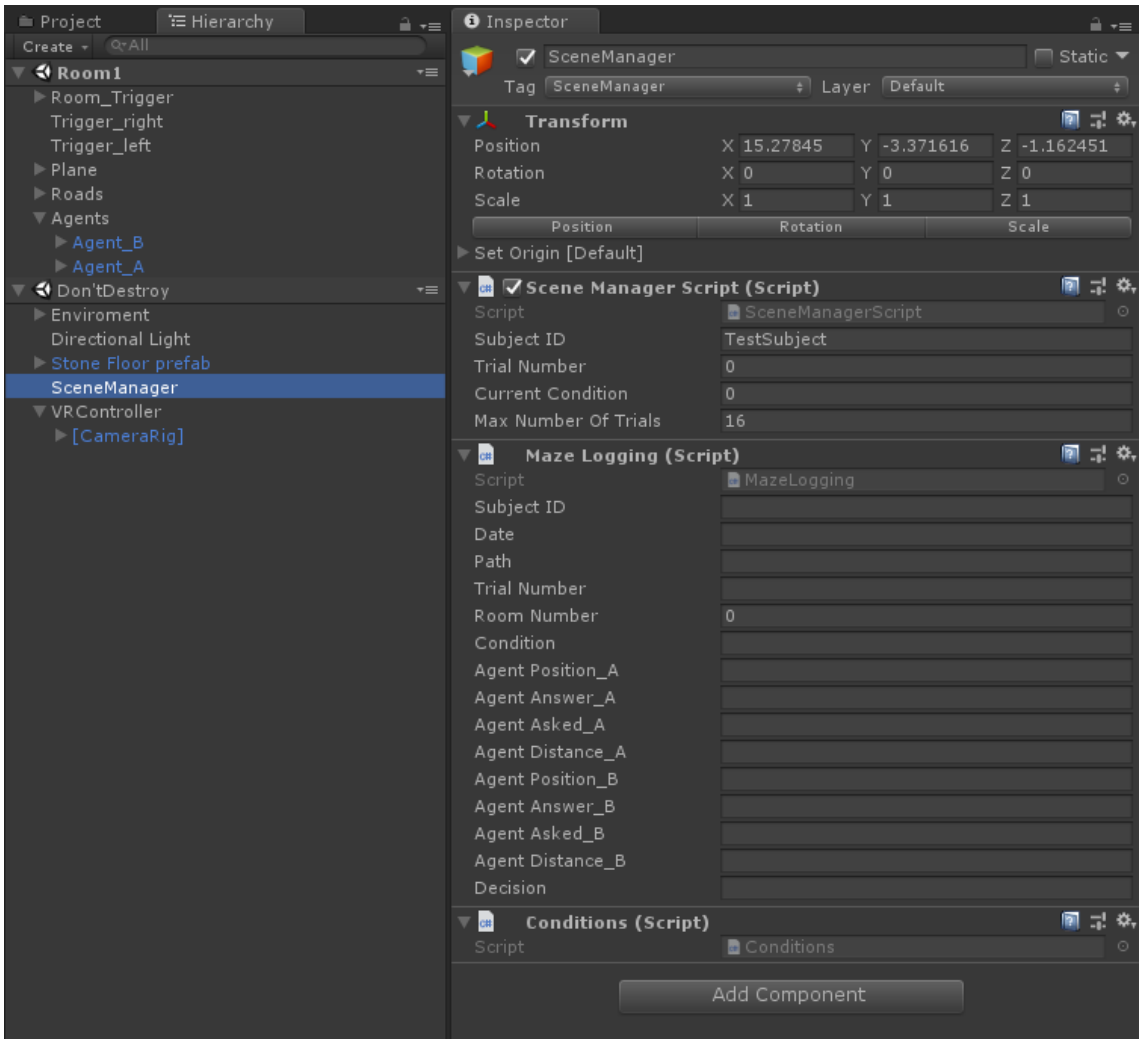


Figure 4.1. Scene Manager in Inspector

4.1.3 Entered Room

This script only sets the logging value for the room enter time when the user hits a trigger at the beginning of the room.

4.1.4 Made Decision

This script sets the logging value for the decision when the user hits a trigger after the intersection. It also ends a trial and writes all logging values of the trial into

the .csv file. Afterwards it resets all logging values to start the next trial.

4.1.5 Load Next Room

This script is responsible for loading the room for the next trial. It is activated when the user hits a trigger after the last intersection.

It gets the next room number from the maze logging, where this value has already been stored. It also gets the current condition from the Scene Manager Script. It then goes ahead and unloads all scenes expect the current one and the "don't destroy" scene. Afterwards it loads the new scene additive. It gets agent A and agent B from the new scene, using their tag. It assigns the position, rotation and audio file for each agent according to the current condition.

4.1.6 Maze Logging

This script is responsible for storing all values in the log file. The name of the log file consists of the subject ID and the date. The table 4.2 shows which values are stored for each trial, the meaning and where they are set (which script).

The log files are stored under Assets/LogFiles/. The file name is year-month-day_hour-minute-second_subjectID.csv.

Table 4.1. Maze Logging Values

Logging Value	Meaning	Set in
General Information		
Trial Number	ID which trial the participant is in	Scene Manager Script
Trial Start Time	at which time the participant started this trial (starts when the participant is done with the previous trial)	Scene Manager Script
Room Number	which room the participant uses for this trial	Scene Manager Script
Room Enter Time	at what time the user actually entered the room	Enter Room
Condition	which condition from the script "conditions" is used	Scene Manager Script
Agent A		
Agent Position A	position of the agent	Scene Manager Script
Agent Answer A	answer the agent gave	Scene Manager Script
Agent Asked A	whether the agent was asked by the participant	Ask Agent
Agent Distance A	the distance between the agent and the participant when asked (empty when the participant didn't ask this agent)	Ask Agent
Agent Time A	the time when the agent was asked (empty when the participant didn't ask this agent)	Ask Agent
Agent B		
Agent Position B	position of the agent	Scene Manager Script
Agent Answer B	answer the agent gave	Scene Manager Script
Agent Asked B	whether the agent was asked by the participant	Ask Agent
Agent Distance B	the distance between the agent and the participant when asked (empty when the participant didn't ask this agent)	Ask Agent
Agent Time B	the time when the agent was asked (empty when the participant didn't ask this agent)	Ask Agent

Table 4.2. Maze Logging Values

Logging Value	Meaning	Set in
Decision		
Decision	which way the participant went (right or left)	Made Decision
Decision Time	at what time the participant made the decision	Made Decision
Time Deltas		
Room Enter - Decision	time between the participant entering the room and making a decision	Maze Logging
Trial Start - Decision	time between the trial start and the participant making the decision	Maze Logging
Agent Asked A - Decision	time between asking agent A and the participant making the decision (empty if agent wasn't asked)	Maze Logging
Agent Asked B - Decision	time between asking agent B and the participant making the decision (empty if agent wasn't asked)	Maze Logging
Agent Asked A - Agent Asked B	time between asking the two agents (empty if only one agent was asked)	Maze Logging

4.1.7 Condition Model

This script defines the structure of all the needed variables for both agents for a condition. This includes:

- The audio file for agent A
- The animation name for agent A
- The position of agent A
- The rotation of agent A
- The audio file for agent B
- The animation name for agent B
- The position of agent B
- The rotation of agent B

4.1.8 Conditions

This script stores the actual values for each condition in the format of the condition model. The positions and rotations are fix values for both the right and the left position. They only need to be assigned by the condition model.

There are 8 conditions. They are made up of the following factors: 2 (agent answer: left or right) x 2 (agent conformity: same or different answer) x 2 (agent position: left or right).

4.1.9 Ask Agent

This script enables the user to talk to an agent. When the user is within the collider of the agent (s)hecan press the trigger button (HTC Vive and Oculus Rift S) or "E" when using the desktop mode to ask an agent. This triggers the animation for this agent for this condition as well as the corresponding audio file. The user can not ask again until the animation is done. When the answer animation and audio file are done the agent goes back to its idle movement.

4.1.10 Agent Look

The agents follow the participant with their eyes if (s)heis close enough. This is determined by the collider on the eyes. If the participant leaves the trigger the agent will look forward again. The look radius is clamped in order for the agent's eye movement to not look strange.

For the current agents the radius had to be clamped differently. This is why the agent needs to be specified in the inspector. See all configurations in 4.2.

4.2 XR Setup

This folder contains all scripts necessary to run the application with any VR headset and in a desktop mode. This part was developed by Florian Kern under the ViLeArn project at the University of Wuerzburg and is copyright protected. For more information visit https://vilearn.hci.uni-wuerzburg.de/.

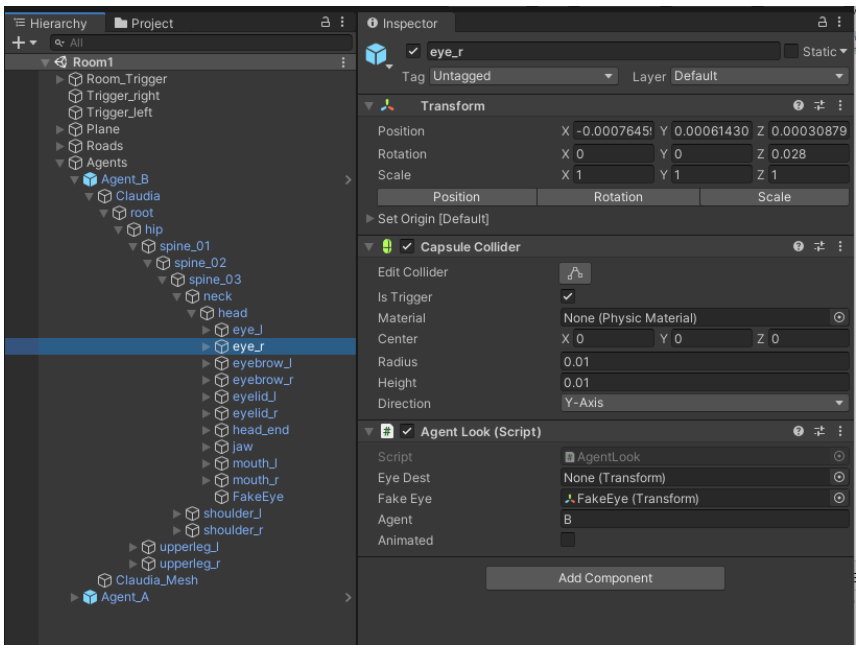


Figure 4.2. Agent Look in Inspector

4.2.1 Game State

This script saves the game state and determines whether VR or the desktop mode is used and which buttons will be used for the interaction.

4.2.2 Mouse Look

This scripts makes it possible to make the camera follow the mouse courser.

4.2.3 XR Controller

Here the logic for the height of the participant and walking is handled.

4.2.4 XR Controller Input

This script handles the input from the controller.

4.3 Scenes

4.3.1 Don't Destroy

This scene contains all elements that are never unloaded and must always be present. This includes:

- The environment (floor, trees and houses)
- The lighting
- The Players (which include the VR and the desktop mode)
- The Scene Manager (which has the Scene Manager Script and the Maze Logging attached to it)

4.3.2 Level Scenes

All level scenes include the same elements.

- The roads
- The agents
- Different trigger

The roads are just a part of the environment. They are not included in the "Don't Destroy" Scene, because multiple roads can lead to the same room. If all roads were displayed at all times the user would see roads (s)heis not supposed to take. Only showing the current roads also helps with the illusion that there are many rooms and not just the same rooms over and over again.

There are two agents in each scene (A and B) which are childed to an Agents game object (4.4). This game object has a tag which shows the room that these agents belong to. Each agent has an animator (to play animations) (4.5, 4.6), an audio source (to play its answer audio file), a capsule collider (used as a trigger) and the Ask Agent script attached (4.3). The capsule collider is used to determine whether the participant is close enough to the agent to talk to him.

There are 5 trigger in each room.

- Decision Right (4.7)

- Decision Left
- Entered Room (4.8)
- Trigger Right (4.9)
- Trigger Left

Decision Right an Decision Left are used to detect when the user has made a decision and which one it it. They have the "Made Decision" script attached to them and run them when the user enters this trigger. The script also has a variable attached that shows which room this road leads to. This variable was set in the editor for each Trigger in each Room. It must not be changed! These triggers also determine the start of a new trial.

Entered Room has the "Entered Room" script attached to it. It loges the time the participant entered this room.

Trigger Right and Trigger Left are positioned halfway to the next room. They have the "Load Next Room" script attached to them. The next room is only loaded when the participant is far enough away to not notice that the previous room is unloaded, but not close enough to notice that the next room isn't loaded yet.

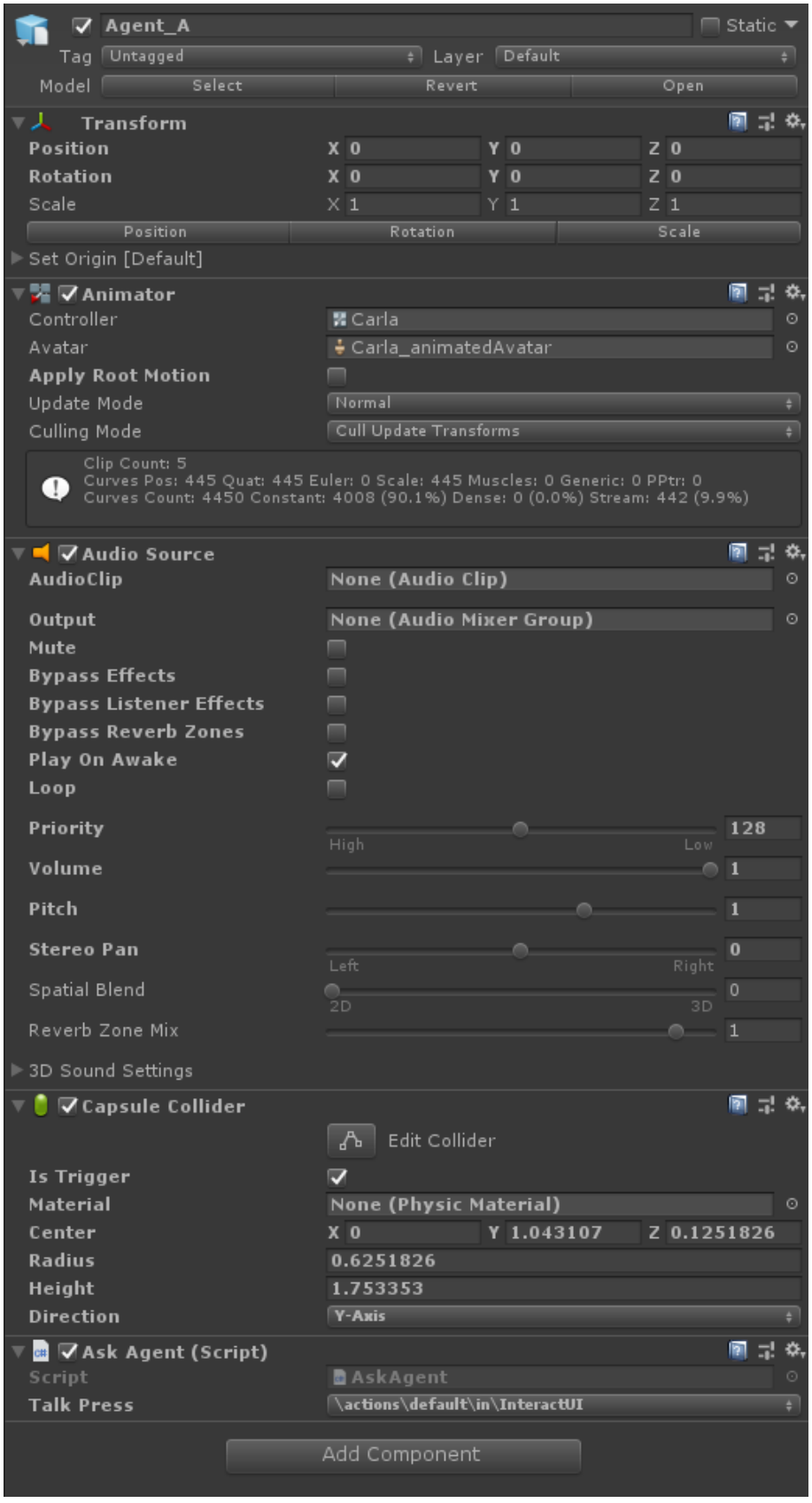


Figure 4.3. Agent in Inspector

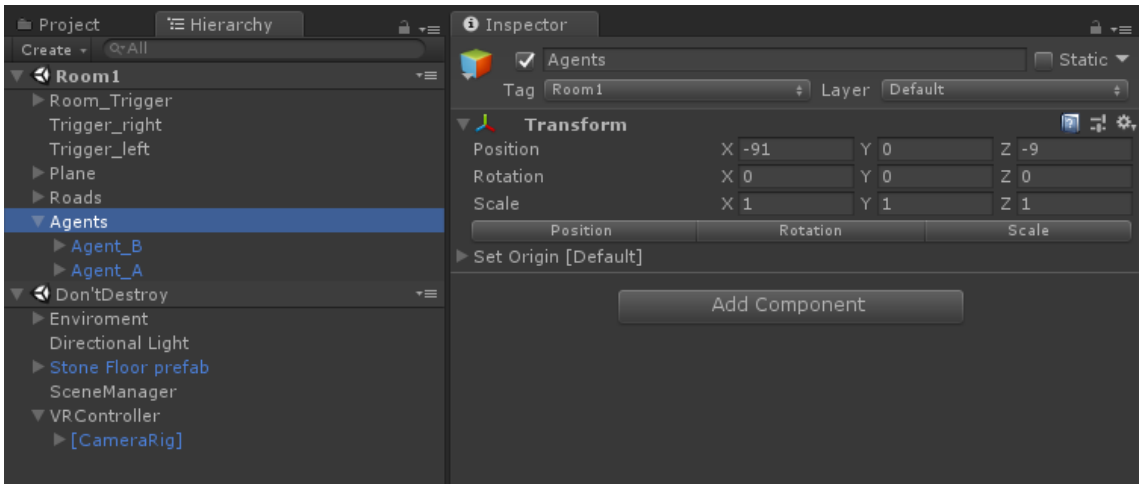


Figure 4.4. Agents game object

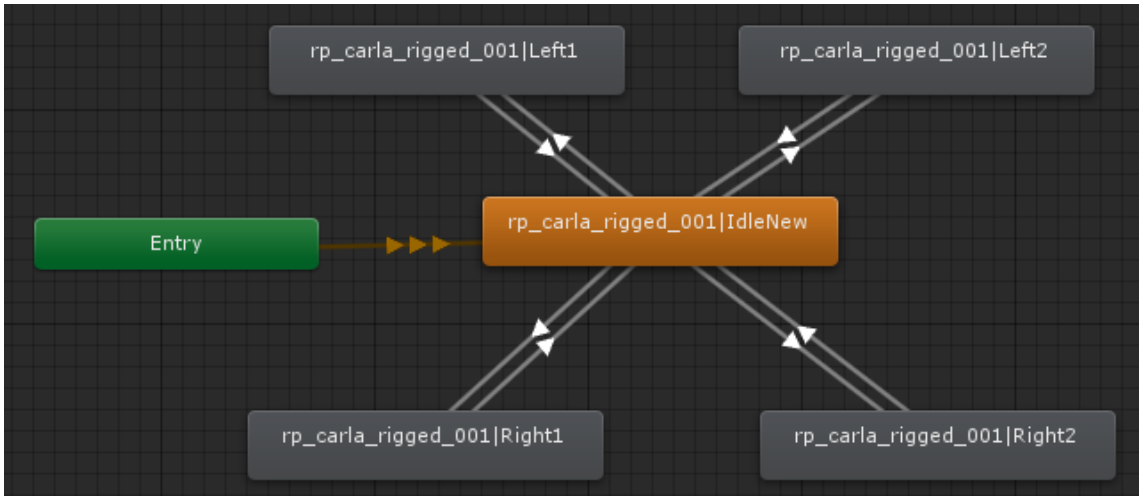


Figure 4.5. Animator

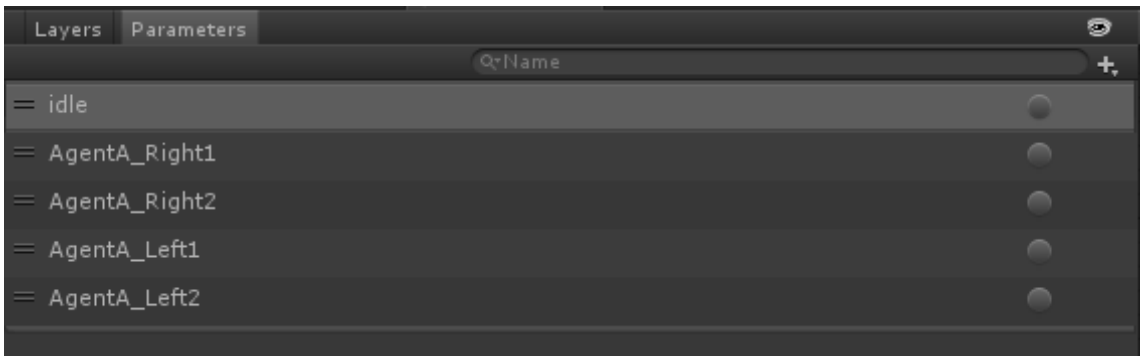


Figure 4.6. Animator Parameters

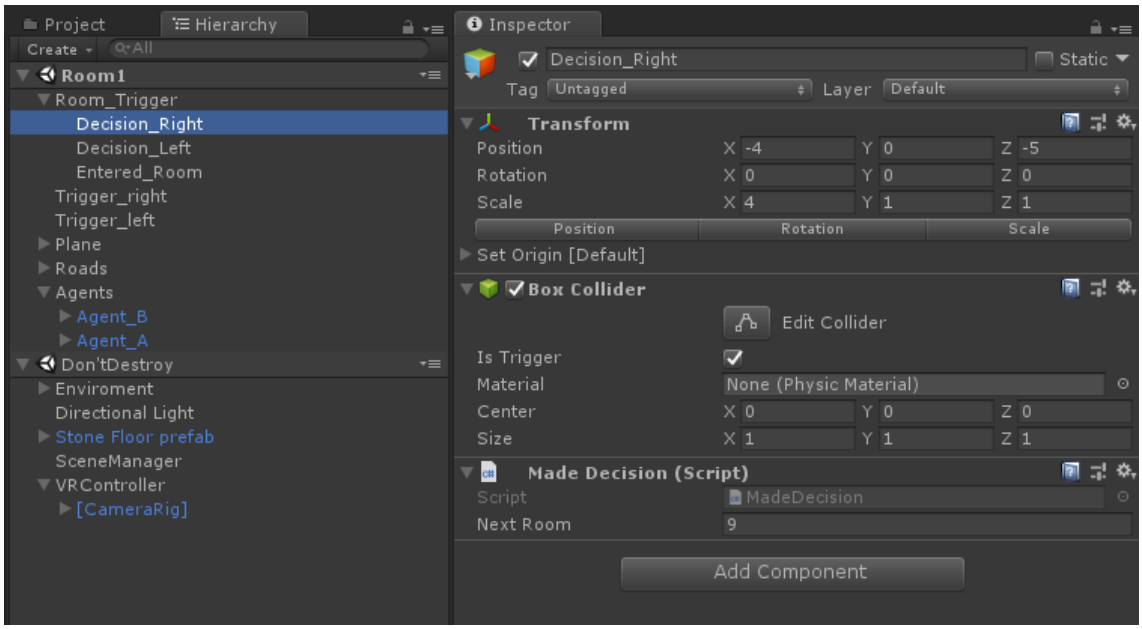


Figure 4.7. Decision Right or Left

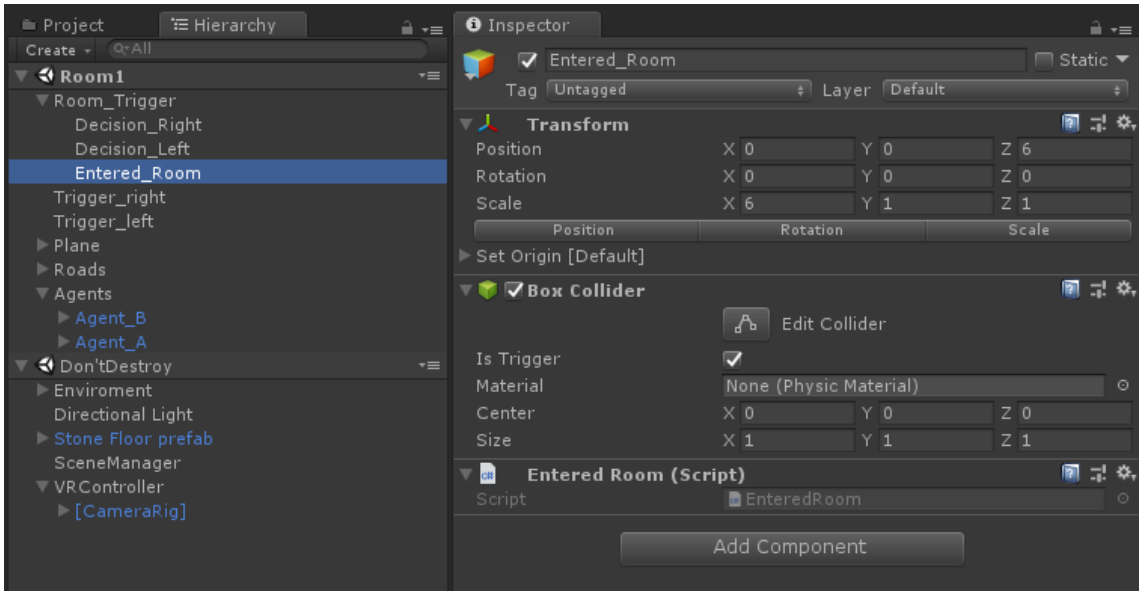


Figure 4.8. Entered Room

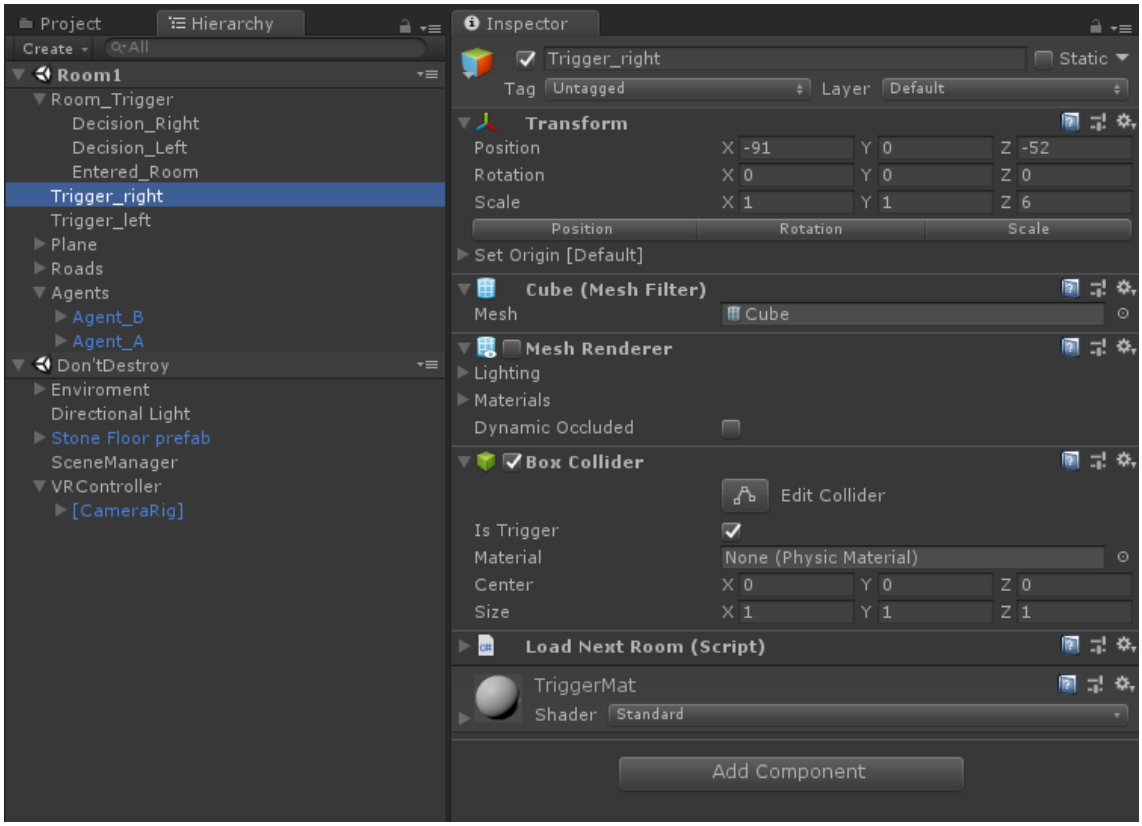


Figure 4.9. Trigger Right or Left

5 Replace an avatar

The programm will use the avatar prefabs which are assigned in the SceneManager script (see 5.1). The avatars used are in the folder Assets/Resources/MazeTask/Agents. A new avatar needs to be in the .fbx format. This also has to include the animations. Currently 5 animations are used: idle, right1, right2, left1, left2.

1. Add the new avatar to the Unity folder.
2. Open a level scene.
3. Drag the new agent from the project window onto the "agents game object" (within Unity) to child it to that game object.
4. Rename the agent to either "Agent_A" or "Agent_B".
5. Set the transform position to 0, 0, 0.
6. Create an animator for the agent (see 4.5 and 4.6 for reference and https://www.youtube.com/watch?v=JeZkctmoBPw for help). Make sure you attache the triggers to the animation transition.
7. Add the necessary components (see 4.3)
   - Animator (add your animator from the previous step)
   - Audio Source (nothing has to be changed here)
   - Capsule Collider (adjust size)
   - Ask Agent
8. Add the components for eye movement (see 4.2)
   - Create a sphere under the head with the same position as one of the eyes and size (0.0001, 0.0001, 0001)
   - Deactivate the mesh renderer and name it "Fake Eye"
   - Add a capsule collider to both "real" eyes and mark it as trigger
   - Add the Agent Look script to both eyes and assign the fake eye as the eye destination

9. Create a Prefab of this avatar
10. Delete the avatar from the scene
11. Assign the prefab to the SceneManager script (see 5.1)

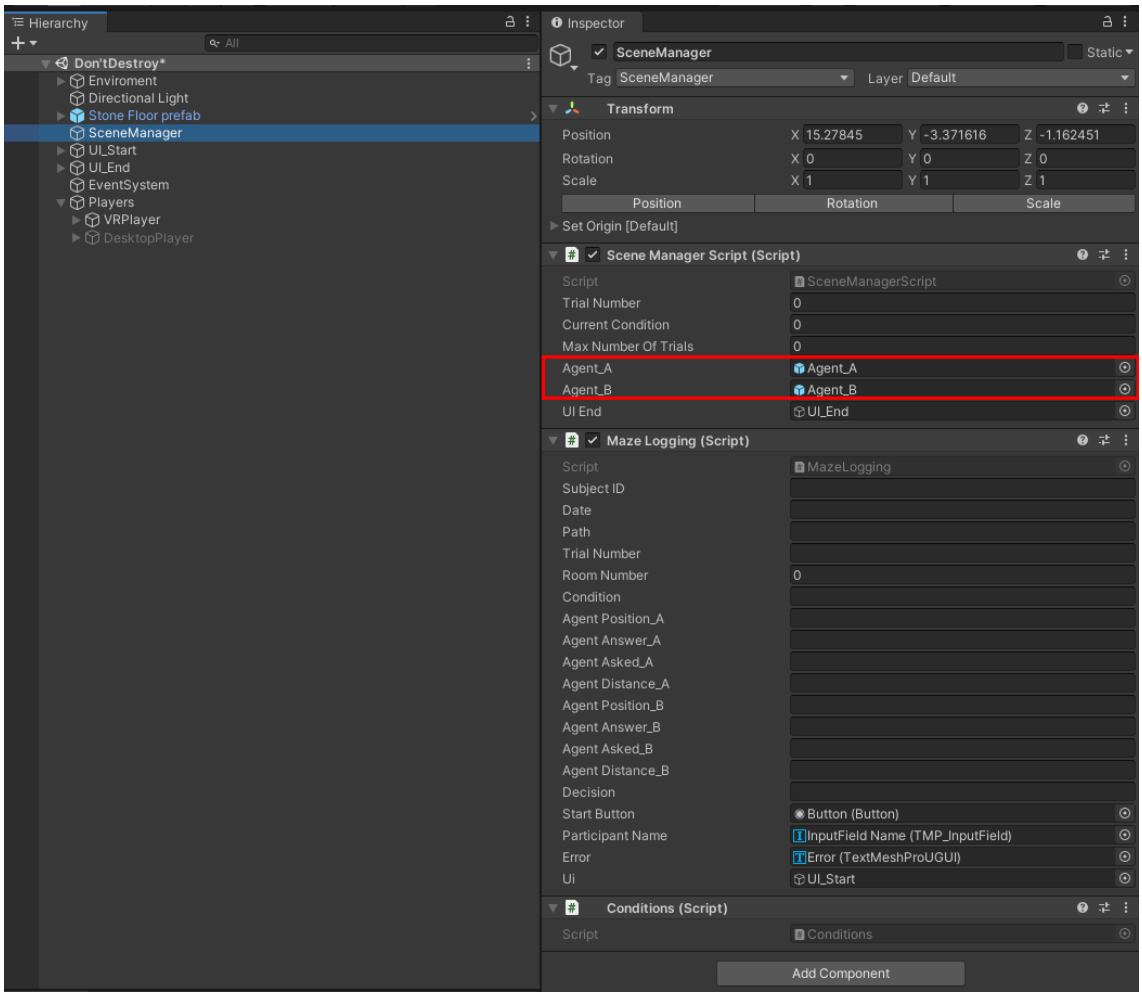


Figure 5.1. Assign Avatar Prefab to Scene Manager script

6 Start the application

The application can be used either with an VR headset or using the desktop mode with mouse and keyboard. To change between the two modes go to the "Players" GameObject. Here you can find the child GameObjects "VRPlayer" and "DesktopPlayer" (see 6.1). For the VR mode activate the "VR Player" and deactivate the "DesktopPlayer". For the desktop mode activate the "DesktopPlayer" and deactivate the "VR Player".

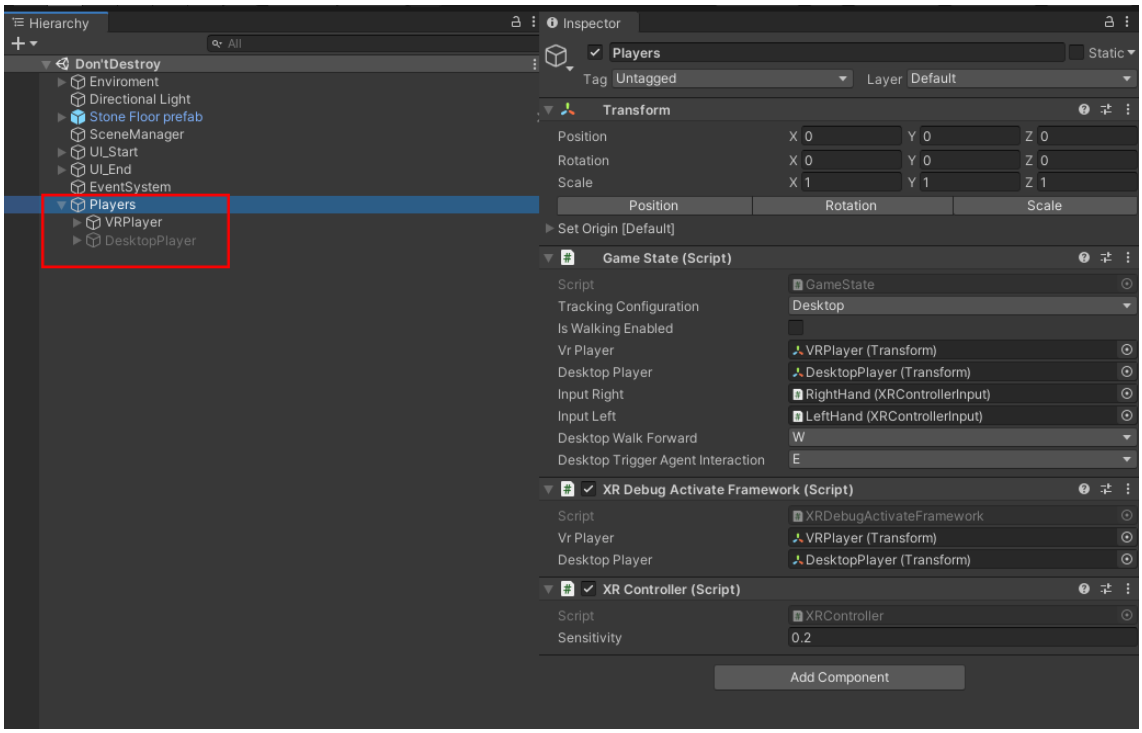


Figure 6.1. VRPlayer and DesktopPlayer

There are two ways to start the application. Either way you have to first start Steam VR and one of the controllers of the used headset.

Now you can either:

- Use the Unity Editor

- Start the Maze.exe

To use the Unity Editor open the project with Unity 2019.4.8f. Open the scene "Don't Destroy". Afterwards you can start the application by pressing the play button in the editor.

To start the Maze.exe simply navigate to the folder "Maze Application" in the root folder and copy it to your desktop. Open the folder and double click on the Maze.exe. The application will launch automatically.

Now you have to enter the participant ID and trial number and press "Start Experiment".

VR Mode: The participant can now walk through the maze. To do so (s)he has to press the touchpad (HTC Vive) or move the thumbsstick forward (Oculus Rift S). (S)He will the move into the direction (s)heis looking at. To talk to an agent (s)he has to be close enough and then press the trigger with his/her index finger (HTC Vive and Oculus Rift S).

Desktop Mode: In this mode the camera is controlled by the mouse courser. You can walk by pressing "W" and the camera will move in the look direction of the camera. To talk to an agent press "E".

When the maximum number of trial is reached the participant sees an overlay and won't be able to walk anymore. This is the end of the application. When using the Unity Editor press the play button again. The Maze.exe application can be closed by pressing Alt + F4 on your keyboard.

Note: When you are using the Unity Editor you have to open any other window than Unity and then switch back to Unity before the application is closed. Otherwise no logfile will be created. This is a weird behavior from Unity that I cannot change.