UnityVR User Manual

Michael Schreier

December 25, 2015

Contents

Outline

The neuronal agents and their virtual environment (VR) are simulated on a distributed and specialized device. The agents have all main abilities of a human, they are capable to execute simple actions like moving or jumping, to move their eyes and their heads and to show emotional facial expressions. Agents learn their behavior autonomously based on their actions and the resulting sensory consequences in the environment. For this purpose, the VR-engine contains a rudimentary action- and physic-engine. Small movements (like stretching the arm) are animated by the VR-engine, while the neuronal model rather controls high-level action choices like grasping a certain object.

Unity3D is used as VR-Engine for UnityVR. Unity3D was chosen as it is an easy to use cross-platform game or VR engine. This document is the user manual. If you look for further development informations refer the Development Manual (*[Repository directory]/unityVR/doc/VR-doc*).

- Chapter 1 starts with a short instruction of Unity3D and its IDE. It also explains the importing of the project UnityVR. It will not cover details or features of Unity3D. If you want to learn more about Unity3D, please visit the Unity3D homepage (`http://www.unity3d.com`).
- Chapter 2 shows the UnityVR workflow by example and describes all essential parts of it. We will create a simple environment and the connection to the neuronal network simulator ANNarchy. All scipts/classes of UnityVR are written in the C# language whereby the ANNarchy client is written in C++. Because of that, some basic knowlege of both languages is required.
- Chapter 3 gives a more detail overview of UnityVR. For a full overview refer to the development documentation (doc/VR-doc/html/index.html).
- Chapter 4 covers further infomation and tips about useful features of Unity3D and the design workflow.

1 Basics of Unity

Unity3D is a game engine with a build-in IDE. It is suited for easy developing of 3D games. This chapter will give a short overview over the IDE and its features. For further details refer the Unity3D documentation (`http://unity3d.com/learn/documentation`).

1.1 Installation

You can obtain Unity3D from

- Official homepage (`http://unity3d.com/unity/download`)
- For older Unity3D versions go to `http://unity3d.com/unity/download/archive`

You could get a 30 day trail for free, but some parts of UnityVR (RenderTextures) require a Unity3D professional license. It is advised to use Unity-Version 4.2.0, provided under unityVR_xxx/install/UnitySetup-4.2.0.exe

1.2 Elements of the IDE

This section will give a short overview about Unity3D. Fig. 1.1 shows the IDE of Unity3D 4.2.0f.

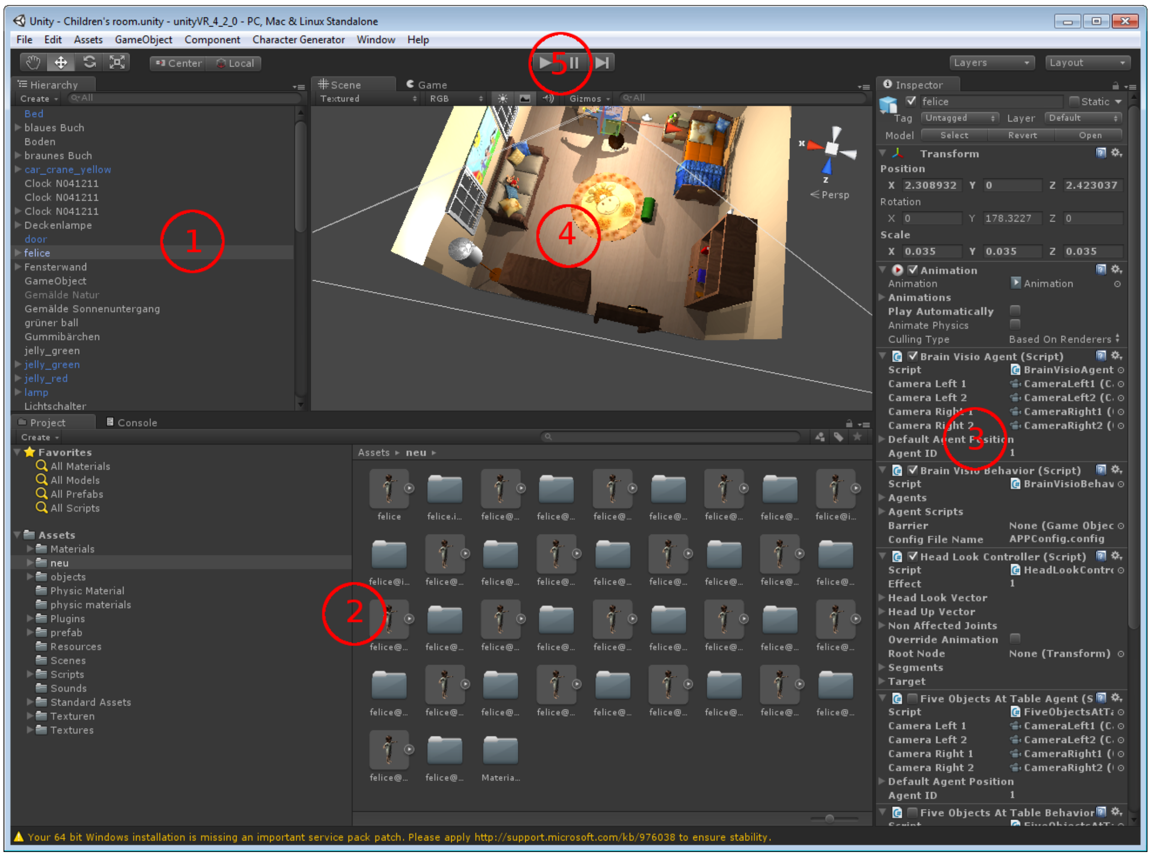
(1) **Scene Graph**
Tree View of all currently present objects in the active scene.

(2) **Project View**
Shows the directory structure and all files that are contained in the *Assets* folder.

(3) **Object Inspector**
Shows details and attached components/scripts of the currently selected object.

(4) **Scene**
This is the development view of the actual scene. Here you can add objects to the scene or modify objects.

(5) **Start/Pause/Stop**
Starts/pauses or stops the game engine for playback in the editor.



felice — Static
Tag: Untagged — Layer: Default
Model: Select | Revert | Open

Transform
Position: X 2.308932, Y 0, Z 2.423037
Rotation: X 0, Y 178.3227, Z 0
Scale: X 0.035, Y 0.035, Z 0.035

Animation
Animation: Animation
Animations
Play Automatically: ☐
Animate Physics: ☐
Culling Type: Based On Renderers

Brain Visio Agent (Script)
Script: BrainVisioAgent
Camera Left 1: CameraLeft1 (C
Camera Left 2: CameraLeft2 (C
Camera Right 1: CameraRight1 (
Camera Right 2: CameraRight2 (
Default Agent Position
Agent ID: 1

Brain Visio Behavior (Script)
Script: BrainVisioBehav
Agents
Agent Scripts
Barrier: None (Game Objec
Config File Name: APPConfig.config

Head Look Controller (Script)
Script: HeadLookContr
Effect: 1
Head Look Vector
Head Up Vector
Non Affected Joints
Override Animation: ☐
Root Node: None (Transform)
Segments
Target

Five Objects At Table Agent (S
Script: FiveObjectsAtT
Camera Left 1: CameraLeft1 (C
Camera Left 2: CameraLeft2 (C
Camera Right 1: CameraRight1 (
Camera Right 2: CameraRight2 (
Default Agent Position
Agent ID: 1

Five Objects At Table Behavior

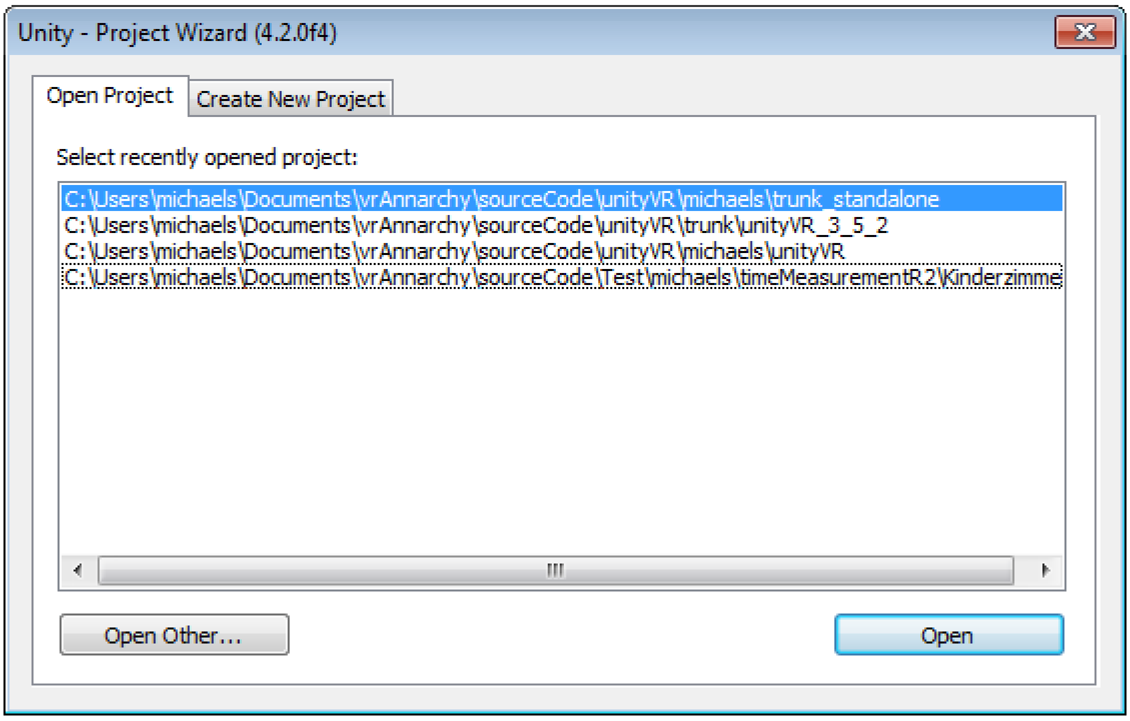
Your 64 bit Windows installation is missing an important service pack patch. Please apply http://support.microsoft.com/kb/976038 to ensure stability.

Figure 1.1: Unity3D IDE (Version 4.2.0f)

1.3 First Time Importing the Project

1. Checkout the latest UnityVR Release from via HG from `ssh://ai.informatik.tu-chemnitz.de//work/mercurial/vrNeuroStable`.
   The recommended HG tool for Windows is TortoiseHG (`http://tortoisehg.bitbucket.org`).
2. Click on *"Open Project"* in the *"File"* menu. The Project Wizard will popup (Fig. 1.2).
3. Now click on the *"Open other..."* button and choose the directory which contains the project. This will start a new instance of Unity3D and the import status window appears (Fig. 1.3). This process can take up to 30 minutes depending on the computer and project size.

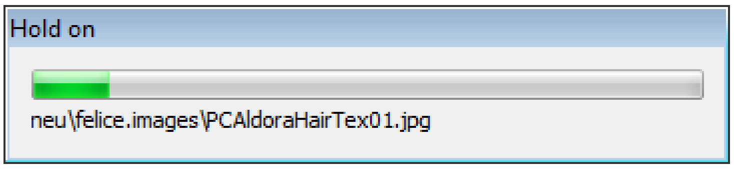
Note: Make sure to run Unity with a proper GPU.



Select recently opened project:

C:\Users\michaels\Documents\vrAnnarchy\sourceCode\unityVR\michaels\trunk_standalone
C:\Users\michaels\Documents\vrAnnarchy\sourceCode\unityVR\trunk\unityVR_3_5_2
C:\Users\michaels\Documents\vrAnnarchy\sourceCode\unityVR\michaels\unityVR
C:\Users\michaels\Documents\vrAnnarchy\sourceCode\Test\michaels\timeMeasurementR2\Kinderzimme

Figure 1.2: The Project Wizard shows recently opened projects. The *"Open other..."* button lets you open another already existing Project.



neu\felice.images\PCAldoraHairTex01.jpg

Figure 1.3: This windows shows the actual import progress. Depending on the PC and the Project, it can take a long time.

2 Creating New Scene by Example

This chapter describes the creation of a minimal scene in Unity3D step by step. We begin with an empty scene where we are going to create a simple environment with an agent with all essential scripts and the connection with ANNarchy.

2.1 UnityVR Design / Code

These following steps describe the necessary procedures in Unity3D to create a new environment with the agent Felice and are done inside the UnityVR project. Make sure you import of UnityVR into Unity3D properly as descripted in 1.3.

2.1.1 Creating the Environment

1. Click the *File* menu and choose *New Scene*.
2. Click the *File* menu and choose *Save Scene*. Save it into the *Assets/Scene* Folder and choose a descriptive filename.

Now we have empty scene (Fig. 2.1). In the following step we want to create a floor, a table, lights and the agent.

3. We need a floor, on which the agent can walk. Click the *GameObject* menu, sub menu *Create Other* and choose *Plane*. A *plane* has an already attached mesh collider, which is needed, so that the agent does not fall through the floor.
4. Go to the *Texture* folder in the Project View and attach a texture on the new created plane via drag and drop (Fig. 2.2).
5. Go into *objects/common* folder, select the *table* object and drag it into the scene. We have to scale the table down, to make it smaller. The easiest way to do that is to ajust the scale values in the Object Inspector. I chose the value 0.05 for *X*, *Y* and *Z* (Fig. 2.3). Optionally, you could add a *BoxCollider* and a *Rigidbody* to the table to enable collision detection and physics respectively.
6. We create a light source. Click the *GameObject* menu, sub menu *Create Other* and choose *Point Light*. Drag it to a spot in the scene and change its values in the Object Inspector as you wish.
7. Adjust the values of the *Main Camera* so that the environment is visible in the Camera View.

Our VR environment is finished. Now we start with the controlling scripts.

8. Create an empty object (*GameObject->Create Empty*). Name it *MasterObject*.

Now we attach a derived *BehaviorScript* to the *MasterObject* created in the last step. The *BehaviorScript* is the main control script in UnityVR. It is responsible for controlling the environment and initialization of the agents. The behavior to messages from the ANNarchy client can be changed by overriding functions in a derived class of *BehaviorScript*. Even if you do not want to change the behavior, it is not intended to use the base class *BehaviorScript* directly. As a convention, you should always create a derived class from *BehaviorScript*. Also the *BehaviorScript.AgentInitalization* function needs information about the type of the *AgentScript* which we will create later.

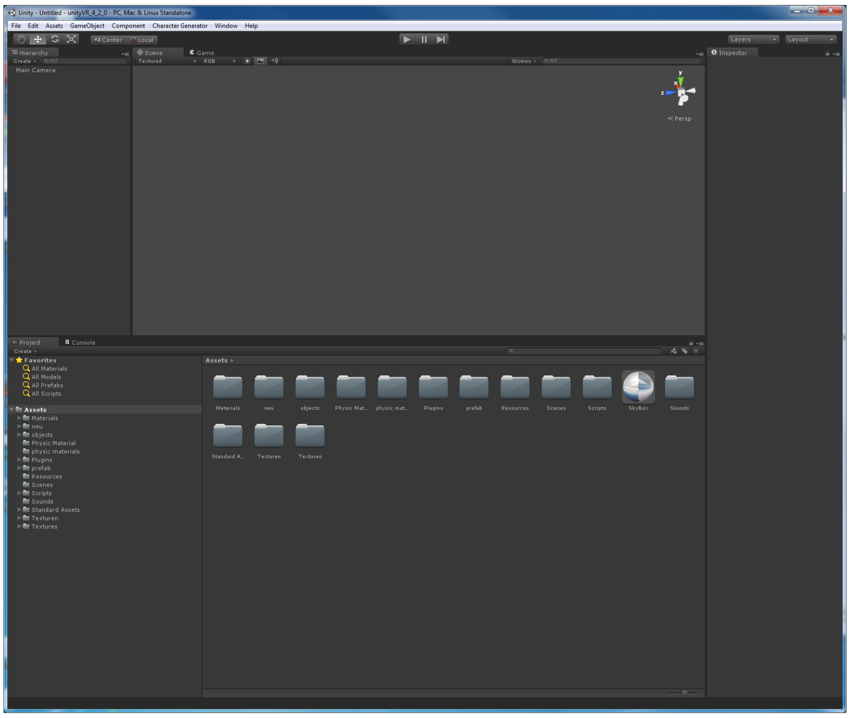
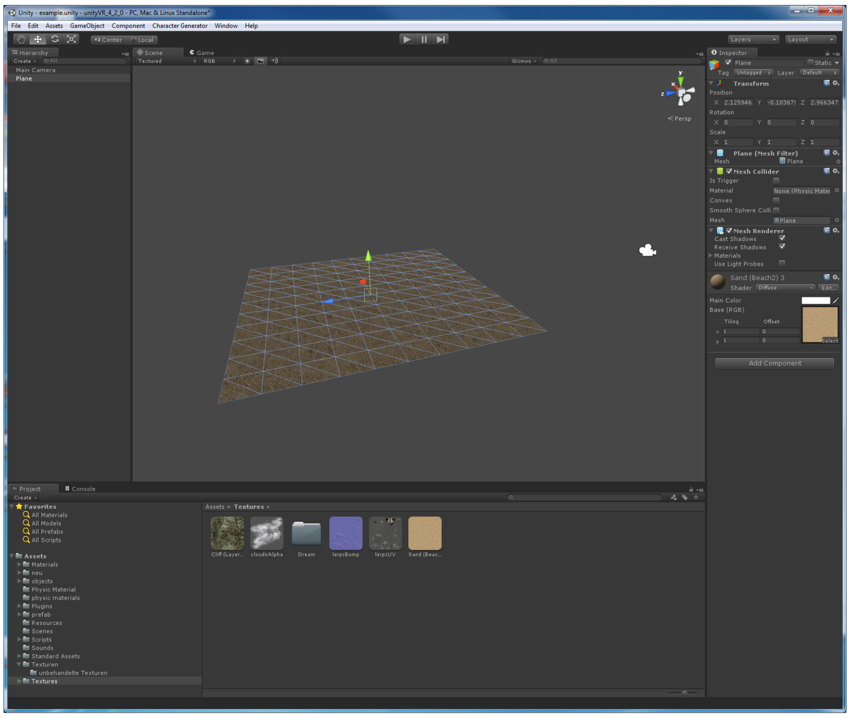


Figure 2.1: View of a new created scene



Inspector
Plane — Tag: Untagged — Layer: Default
Transform
Position X 2.125946 Y -0.10367 Z 2.966347
Rotation X 0 Y 0 Z 0
Scale X 1 Y 1 Z 1
Plane (Mesh Filter) — Mesh: Plane
Mesh Collider — Is Trigger: ☐ — Material: None (Physic Mater — Convex: ☐ — Smooth Sphere Colli: ☐ — Mesh: Plane
Mesh Renderer — Cast Shadows: ✓ — Receive Shadows: ✓ — Materials — Use Light Probes: ☐
Sand (Beach2) 3 — Shader: Diffuse
Main Color
Base (RGB) — Tiling x 1, y 1 — Offset x 0, y 0

Figure 2.2: Editor View with the instantiated floor plane

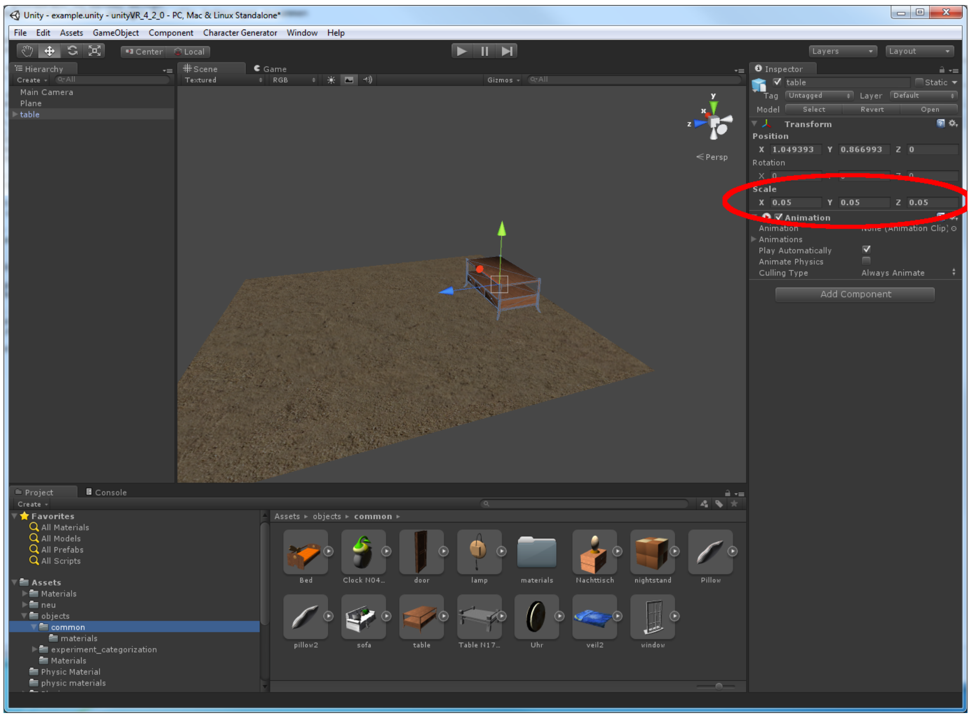


table — Static
Tag: Untagged — Layer: Default
Model: Select, Revert, Open
Transform
Position: X 1.049393, Y 0.866993, Z 0
Rotation: X 0, Y 0, Z 0
Scale: X 0.05, Y 0.05, Z 0.05
Animation
Animation: None (Animation Clip)
Animations
Play Automatically: ✓
Animate Physics: ☐
Culling Type: Always Animate

Figure 2.3: Scene with table. The red ellipse shows the scale options in the Object Inspector.

9. We create the new class *ExampleBehaviorScript*. In the example of a *BehaviorScript* below we will override the *AgentInitalization* and *ProcessTrialReset* function.

```
using System;
using System.Collections;
using SimpleNetwork;
using UnityEngine;
public class ExampleBehaviorScript : BehaviourScript
{
    protected override void AgentInitalization()
    {
        getAllAgentObjects<ExampleAgentScript>();
        getAndInitAllAgentScripts<ExampleAgentScript>();
    }

    protected override void ProcessTrialReset()
    {
       Debug.Log("Trial Reset received");
       base.ProcessTrialReset();
    }
}
```

10. Attach the *ExampleBehaviorScript* to the *MasterObject*.

2.1.2 Creating the Agent

1. Now drag the agent *Felice* (file *./animation/Felice_grasp.fbx*) into the scene. Change the scale values appropriate e.g. 0.02 for *X*, *Y* and *Z*.
2. Change the tag of Felice to "agent". So the initializing function of *ExampleBehaviorScript* is able to find Felice.
3. Disable *Play Automatically* in the animation settings of Felice.
4. The most importent script, that must be attached, is an *AgentScript*. It is the main control script of an agent instance. *AgentScript* controls the responses of the agent to messages from the client. Like the *BehaviorScript*, the *AgentScript* is intended to be derived. The following code shows the *ExampleAgentScript* which must be attached to the agent.

```
using System;
using System.Collections;
using SimpleNetwork;
using UnityEngine;

public class ExampleAgentScript : AgentScript
{
  protected override void ProcessMsgAgentGraspID( MsgAgentGraspID msg )
  {
            Debug.Log("AgentGraspID received");
  }

}
```

4. There are serveral other components from Unity3D and the *Assets/Scripts* folder that must be attached directly to our agent *Felice* to get it working:
   - *Component -> Physics -> Character Controller*[1]
     - Set the values for *Center* to [0, 51, 0], *Radius* to 14 and *Height* to 100. The agent will be inside the collider of the *Character Controller*.
   - *Controller Collider* script (responsible for collision detection)
   - *HeadLookController* script (head rotation of the agent)
   - Add 4 cameras to the agent object, two for each eye.[2]
     - Name them *CameraLeft1*, *CameraLeft2*, *CameraRight1* and *CameraRight2*.
     - Deactivate the *Audio Listener* in the editor for all four cameras.
     - Adjust the postition of the cameras so that they match with the eyes of the agent.
       - ∗ *CameraLeft1* and *CameraLeft2* for the left eye at position [-4.2, 84.5, 8]
       - ∗ *CameraRight1* and *CameraRight2* for the right eye at position [1.5, 84.5, 8]

[1]http://docs.unity3d.com/Documentation/Components/class-CharacterController.html
[2]One camera (no. 1) renders the Eye View to be displayed in the running VR. The other one (no. 2) renders the Eye View for the connected client.

2.1.3 APPConfig.config

The config file of UnityVR is *APPConfig.config*. It is located in the root directory of UnityVR. For a description of all parameters look at 3.7. For this example your *APPConfig.config* should look like the following listing.

```
<?xml version="1.0" encoding="UTF-8"?>
<cConfiguration xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="
    http://www.w3.org/2001/XMLSchema">
  <LocalPort>1337</LocalPort>
  <SyncMode>false</SyncMode>
  <SimulationTimePerFrame>0.1</SimulationTimePerFrame>
  <ImageResolutionWidth>1024</ImageResolutionWidth>
  <SendGridPosition>true</SendGridPosition>
  <MovementSpeed>3</MovementSpeed>
  <CameraDisplayWidth>512</CameraDisplayWidth>
  <FovHorizontal>120</FovHorizontal>
  <FovVertical>90</FovVertical>
</cConfiguration>
```

2.2 ANNarchy/C++ Client

The environment and the agent have seperate TCP sockets to communicate with the client. The IP address is the IP of the computer where Unity3D is executed. The ports are chosen by the user in the *APPconfig.config* file with the *LocalPort* property. The environment has the port *LocalPort* whereby the agent has the port number *LocalPort+agentID+1*, whereby *agentID* is a property of *AgentScript*.

The following code shows an example *main.cpp* for the client. You find this file as *example-main.cpp* in the same folder as this document.[3] You need to change the IP address in line 16 to match with to PC where UnityVR is running. To compile and run it follow the steps in 3.9.

```
#include <dirent.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <iostream>
#include "ANNarchy/annarProtoMain.h"
#include "ANNarchy/annarException.h"
#include "NeuronalField/Network.h"

using namespace std;

int main(int argc, char *argv[])
{
  try
  {
    char srv_addr[16] =
    { "xxx.xxx.xxx.xxx" }; //ip address

    // local- and VR-port
    unsigned int remotePort = 1337;

    //number of the agent
    unsigned int agentNo = 0;
```

[3]doc/user-doc/example-main.cpp

```
//create image output directory
char imgDir[] = "imgFinaltest_R3";
DIR *dir = opendir(imgDir);
if (NULL == dir)
{
  printf("Image output directory doesn't exist => Create it!: %s\n", imgDir);
  if (mkdir(imgDir, S_IRWXU | S_IRWXG) == -1)
  {
    printf("Make directory failed! => exiting!\n");
    exit(1);
  }
}
else
{
  closedir(dir);
}

printf("init connections.\n");
AnnarProtoMain * mainObj = new AnnarProtoMain(srv_addr, remotePort, agentNo,
    false);
AnnarProtoSend * sender = mainObj->getSender();
AnnarProtoReceive * receiver = mainObj->getReceiver();

printf("Threads starting.\n");
mainObj->start();
printf("Connections are init. and threads are started.\n");

//Variables for the received data
AnnarImage *rightImage = NULL;
AnnarImage *leftImage = NULL;
int actionState = 0; // action execution state
int id, curID; // actionIDs
bool retValue; // returned by the received functions
string param; // optional parameter for menu item

//send reset signal
sender->sendTrialReset(0);

for(int i = 0; i < 360; i++)
{
  sleep(1);
  //send a movement command
  id = sender->sendAgentMovement(i, 10);
  actionState = 0;

  //wait until movement command was executed
  while (curID != id || actionState != 1)
  {
    retValue = receiver->getActionExecState(curID, actionState);
  }

  //save eye images
```

```
      char name[256];
      sprintf(name, "%s/leftImage_%04i.png", imgDir, t);
      leftImage = new AnnarImage(name);
      sprintf(name, "%s/rightImage_%04i.png", imgDir, t);
      rightImage = new AnnarImage(name);

      if (receiver->getImageData(rightImage, leftImage))
      {
        leftImage->save();
        rightImage->save();
      }

      delete leftImage;
      delete rightImage;
    }
  }
}
```

See also *agentBrain/networkInterfaceCPP/Tests/src/main.cpp* for more examples.

3 UnityVR User Reference

3.1 Download UnityVR

Download the Unity-Project via Mercurial from : `ssh://ai.informatik.tu-chemnitz.de//work/mercurial/vrNeuroStable`or from the AI-software repository (UnityVR Revision xxxxxx).

You can donwload Unity from thier website (`https://unity3d.com/get-unity`). We highly recommend the use of the same Unity version as the existing project was created with - currently 4.2.0. Unity archives: `https://unity3d.com/get-unity/download/archive`

The main-config file is the unityVR_xxx/APPConfig.config (should be copied from install/) and the documentation can be found under unityVR_xxx/doc.

3.2 Project Folder Structure

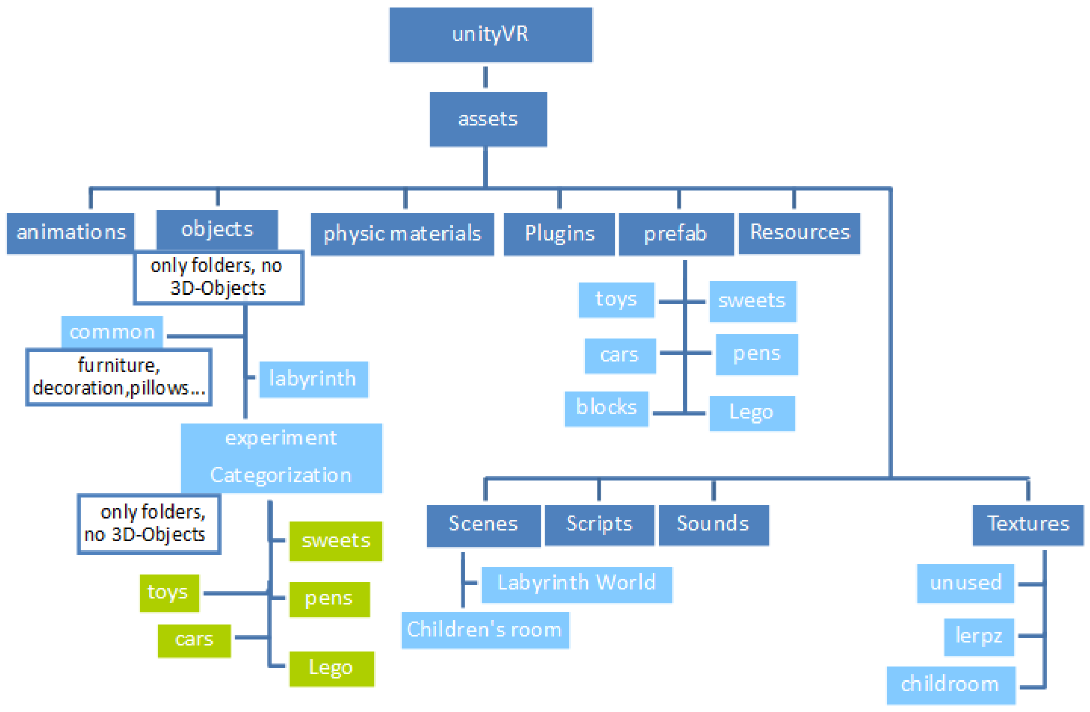


Figure 3.1: Folder structur

Directory	Description
Assets	This is the main folder of the Project View in the IDE
animations	Contains the agents and animations
objects	Contains all used objects
objects/common	Basic and environment objects
objects/labyrith	Specialized objects only used in the Labyrinth scene
objects/experiment categorization	Specialized objects only used in the categorization experiment
physic materials	Demonstration of physic materials like rubber balls
Plugins	Additional .NET assemblies like SimpleNet.dll
prefab	Prefabs (see chapter 4.2)
Resources	Contains objects that are loaded at runtime via *Resources.Load(...)*
Scenes	Contains all scene files
Scripts	Contains all scripts
Sound	Reserved for future usage
Textures	Created by Unity3D for textures

3.3 Classes

Figure 3.2 shows the class diagramm of UnityVR. The two main classes of UnityVR are *AgentScript* and *BehaviorScript*. The *AgentScript* is used to customize the behavior of an agent, while the *BehaviorScript* is responsible for customizing the behavior of the environment.

3.3.1 AgentScript

AgentScript is the main control script of an agent instance. It must be attached directly to the agent. This script is intended to be derived and the behavior of the agent can be modified by the virtual functions with *"Process"* prefix which modify the processing of actions (see 3.4.2). The following code shows all functions, which can be overridden, to create custom behavior.

```
using System;
using System.Collections;
using SimpleNetwork;
using UnityEngine;

public class ExampleAgentScript : AgentScript
{
  protected virtual void ProcessMsgAgentGraspID(MsgAgentGraspID msg) { }

  protected virtual void ProcessMsgAgentGraspPos(MsgAgentGraspPos msg) { }

  protected virtual void ProcessMsgGraspRelease(MsgAgentGraspRelease msg) { }

  protected virtual void ProcessMsgAgentEyeFixation (MsgAgentEyeFixation msg) { }

  protected virtual void ProcessMsgAgentEyeMovement (MsgAgentEyemovement msg) { }

  protected virtual void ProcessMsgAgentPointID (MsgAgentPointID msg) { }

  protected virtual void ProcessMsgAgentPointPos (MsgAgentPointPos msg) { }

  protected virtual void ProcessMsgAgentTurn( MsgAgentTurn msg ) { }
```

```
    protected virtual void ProcessMsgAgentInteractionID (
      MsgAgentInteractionID msg) { }

    protected virtual void ProcessMsgAgentInteractionPos (
      MsgAgentInteractionPos msg) { }

    protected virtual void ProcessMsgAgentMovement (MsgAgentMovement msg) { }

    protected virtual void ProcessMsgAgentMoveTo(MsgAgentMoveTo msg) { }

    protected virtual void ProcessMsgAgentCancelMoveTo(MsgAgentCancelMoveTo msg) { }

    protected virtual void ProcessMsgAnnarNetwork(AnnarNetwork msg) { }

    protected virtual void ProcessMsgAnnarUpdateRates(AnnarNetwork msg) { }
  }
```

3.3.2 BehaviorScript

BehaviorScript is the main control script for the environment. It stores all references to agent objects and all attached *AgentScript*s from the scene. It also initializes the *AgentScript*s. All agent objects must have the "agent" tag, otherwise *BehaviorScript.AgentInitalization* can not find the agents automatically. A minimal example of a *BehaviorScript* which initializes the agents looks like this:

```
using System;
using System.Collections;
using SimpleNetwork;
using UnityEngine;
public class ExampleBehaviorScript : BehaviourScript
{
    protected override void AgentInitalization()
    {
        getAllAgentObjects<ExampleAgentScript>();
        getAndInitAllAgentScripts<ExampleAgentScript>();
    }
}
```

The following listing shows all possible functions which can be overridden. Mark that these functions are within the class.

```
    protected virtual void ProcessMsgEnvironmentReset (MsgEnvironmentReset msg)
    {
      //custom EnvironmentReset behavior
    }

    protected virtual void ProcessMsgTrialReset (MsgTrialReset msg)
    {
        //custom TrialReset behavior
    }
```

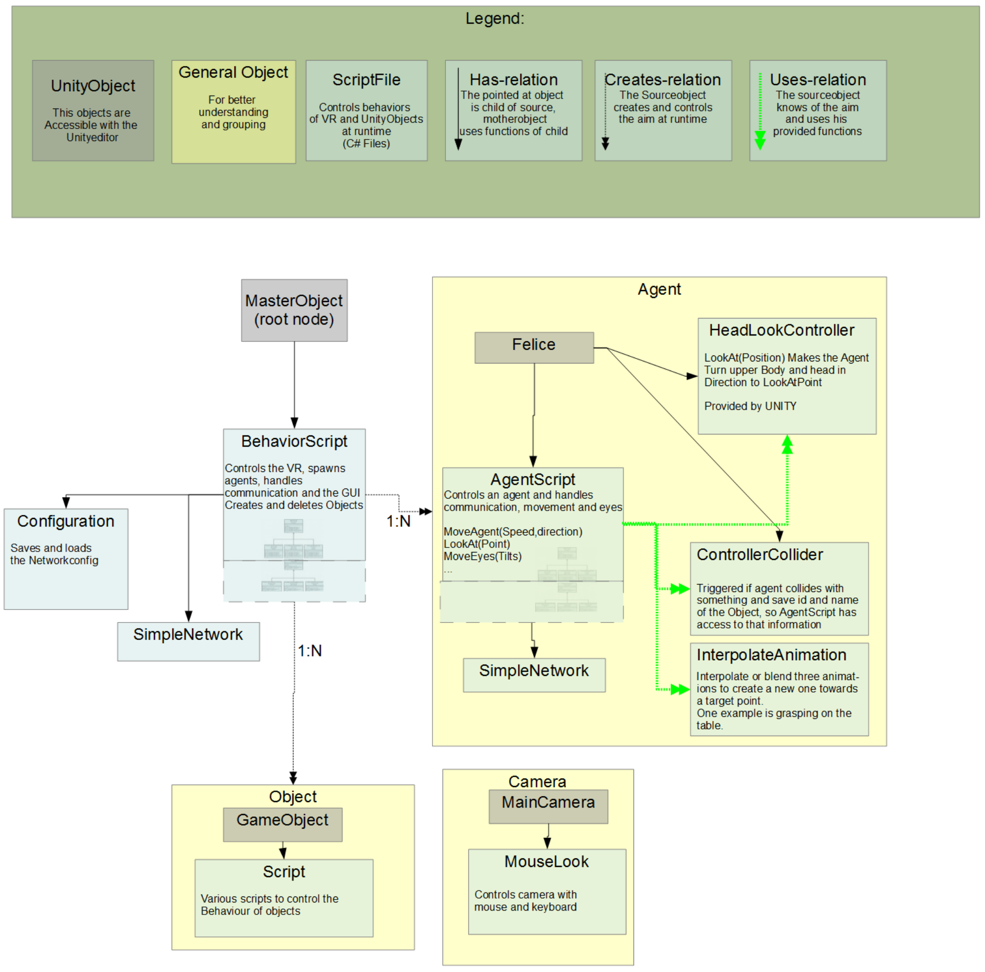

Legend:
UnityObject
This objects are Accessible with the Unityeditor
General Object
For better understanding and grouping
ScriptFile
Controls behaviors of VR and UnityObjects at runtime (C# Files)
Has-relation
The pointed at object is child of source, motherobject uses functions of child
Creates-relation
The Sourceobject creates and controls the aim at runtime
Uses-relation
The sourceobject knows of the aim and uses his provided functions
MasterObject (root node)
BehaviorScript
Controls the VR, spawns agents, handles communication and the GUI Creates and deletes Objects
Configuration
Saves and loads the Networkconfig
SimpleNetwork
1:N
1:N
Agent
Felice
HeadLookController
LookAt(Position) Makes the Agent Turn upper Body and head in Direction to LookAtPoint
Provided by UNITY
AgentScript
Controls an agent and handles communication, movement and eyes
MoveAgent(Speed,direction)
LookAt(Point)
MoveEyes(Tilts)
...
ControllerCollider
Triggered if agent collides with something and save id and name of the Object, so AgentScript has access to that information
InterpolateAnimation
Interpolate or blend three animations to create a new one towards a target point. One example is grasping on the table.
SimpleNetwork
Object
GameObject
Script
Various scripts to control the Behaviour of objects
Camera
MainCamera
MouseLook
Controls camera with mouse and keyboard

Figure 3.2: UnityVR Class diagram. AgentScript and BehaviourScript are intended to be derived, so that each agent and each environment has its own behavior.

3.4 Agents

At the moment (17. December 2015) there are two agents included in UnityVR, Lerpz and Felice. Lerpz was the first agent which was included and has only some limited animation capabilities. Felice has a full set of grasp animations. The feature to grasp an object (described in 3.6.1 and 3.6.2) is only usable with Felice.

3.4.1 Sensors

The agent has the following sensors similar to a human.

- Collision detection
- Eye cameras
- Agent position (intended for debugging)

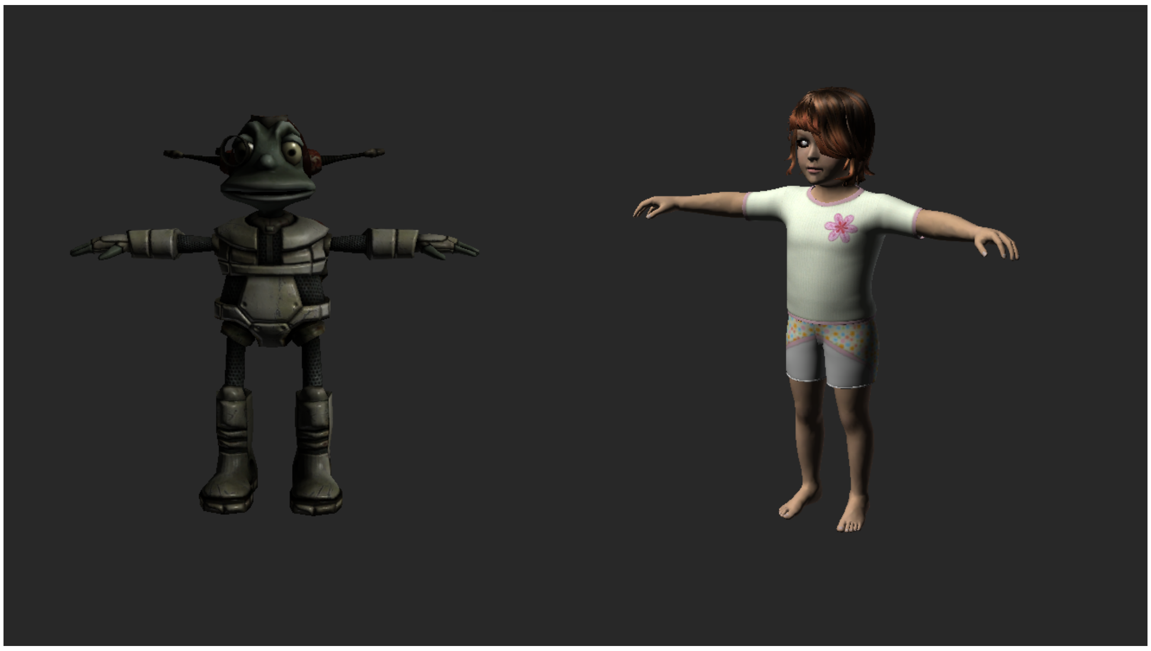


Figure 3.3: Lerpz (left) and Felice (right), the two agents which are included in UnityVR.

3.4.2 Actions

The agent can execute the following actions.

- Execute a movement of the agent
- Rotation of the eyes
- Fixation of the eyes
- Grasping to a position with the right hand
- Grasping to a certain object with the right hand
- Point at an object with the right hand
- Point to a position with the right hand
- Interact with an object
- Interact with a position

3.5 Animations

All informations about using and creating animations are described in:

- unityVR/doc/animation-3Dobjects/daz/animationDocu

For a full overview of the animation capabilities of Unity3D refer to `http://docs.unity3d.com/Documentation/Manual/CreatingGameplay.html`. With Unity3D 4.0, a new animation system was introduced. Since the project started with Unity3D 3.5, UnityVR uses the *Legacy animation system*.

3.6 Animate actions

Animations are used to illustrate the execution of actions. They are necessary for grasping, pointing and interaction with objects.

3.6.1 Grasp to a Specific Position

Felice is able to grasp to custom position. The following code shows an example where the *GraspToPos* function can be used to grasp to a specific position in World Space coordinates.

```
InterpolateAnimations interpolateAnimations;

void Start ()
{
  interpolateAnimations = new InterpolateAnimations(this, "Assets/animations/
      feliceTarget.csv");
  ArmAnimationList = ArmAnimationList.Union(interpolateAnimations.animationArray.
      ToList()).ToList();
}

void GraspToPos(float x, float y, float z)
{
  interpolateAnimations.animate(x,y,z);
}
```

3.6.2 Grab an Object

To get *Felice* to grab an object with her right hand, you need to

- Attach the *HandCollider* script to the right hand of *Felice*. You find the right hand via Felice → Genesis → hip → abdomen → abdomen2 → chest → rCollar → rShldr → rForeArm → rHand
- Attach a *SphereCollider*
  - Set *Position* to [2,-1,1]
  - Set *Radius* to 4
  - Turn *IsTrigger* off
- Attach a *Rigidbody* to the right hand
  - Turn on the *Freeze Position* and *Freeze Rotation* of all coordinates
  - Turn *UseGravity* off
- Change the tag of the right hand to "agentHand"
- Attach a collider and Rigidbody on the target object
- Change the tag of the object to "usable"

Now, when the right hand collides with an tagged object, the object is attached to the hand.

Releasing of an grasped object can be done by the following function:

```
void ReleaseObject()
{
  if(rightGraspedObject != null)
  {
    //change parent to global one, this releases the object
    rightGraspedObject.transform.parent = null;
    rightGraspedObject.rigidbody.useGravity = true;    // enable gravity
    rightGraspedObject.rigidbody.isKinematic = false; // enable pyhsics
    //mark object as released
    rightGraspedObject = null;
  }
}
```

3.7 Config file — APPConfig.config

The APPConfig.config is located in the project root folder.

```
<?xml version="1.0" encoding="UTF-8"?>
<cConfiguration xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="
    http://www.w3.org/2001/XMLSchema">
  <LocalPort>1337</LocalPort>
  <SyncMode>false</SyncMode>
  <SimulationTimePerFrame>0.1</SimulationTimePerFrame>
  <ImageResolutionWidth>1024</ImageResolutionWidth>
  <SendGridPosition>true</SendGridPosition>
  <MovementSpeed>3</MovementSpeed>
  <CameraDisplayWidth>512</CameraDisplayWidth>
  <FovHorizontal>120</FovHorizontal>
  <FovVertical>90</FovVertical>
</cConfiguration>
```

Option	Description
LocalPort	The local port for the TCP connection for the ANNarchy client
SyncMode	Switch between the synchronous/asynchronous mode (see 3.8)
SimulationTimePerFrame	The simulation times in seconds. Precisely, the time between two send time points of the images. In synchron mode, the sending is always executed in the first frame, therefore it is the simulated time of 2 frames. In asychron mode, we wait at least "SimulationTimePerFrame" seconds.
ImageResolutionWidth	Width in pixel for the rendered eye camera images in the scene
ImageResolutionHight	(obsolet) Height in pixel for the rendered eye camera images in the scene
SendGridPosition	If turned on, the Agent sends its position data in every frame to the Client.
MovementSpeed	Movement speed of the agent
CameraDisplayRes	Width & Height of the rendered eye images which will be sent to the client.
FovHorizontal	Field of View in degree of the horizontal
FovVertical	Field of View in degree of the vertical

3.8 Synchron Mode

For a full overview refer *unityVR/doc/VR-doc/synchronDescription.pdf.*

The UnityVR could operate in two modes: synchronous and asynchronous. This can be changed in the configuration file with the *SyncMode* parameter (synchronous mode = true).
The purpose of the synchronous mode is that the agent and the VR use the same time scales. To ensure this, the simulation time scale is independent from the calculation time of the agent and the VR. Such equal time scales are essential to simulate a scientific experiment. The simulation time is sliced into periods and the VR and agent are synchronized at the end of each periode.
The synchronization occurs in the following order. First, the agent receives its input and a *MsgStartSync* message indicating that the agent simulation could start. Then, the agent has enough calculation time to process input information as the VR is paused until the agent has processed all data and has sent all motor commands back. This is indicated by an *MsgStopSync* message sent by the agent to the VR. The synchronization protocol is split over two frames of the VR.

- In the first frame, the VR sends the *MsgStartSync* message and all sensor data, like images and location, to the agent. The VR continues to the second frame. Now the agent should process the data and send motor actions back to the VR. The last messages, which the agent must send back, is *MsgStopSync*.
- The VR remains in the second frame and processes all motor messages from the agent until it received the *MsgStopSync*.

The time scale is controlled by the value of *SimulationTimePerFrame* which can be changed in the configuration file. It determines the interval of sending images. In asynchronous mode, it is the interval in real time, hence the images are sent after this time period has passed. In synchronous mode, *SimulationTimePerFrame* determines the simulation time period. As the images are sent always in Frame 1, it is how much time is simulated over 2 frames in an experiment. For the case that the *MsgStopSync* is lost, there is a timeout implemented in the second frame.

3.9 Connection with ANNarchy

UnityVR can be connected to the ANNarchy library via network (tcp/ip and protobuf). To do so, you must install the following tools:

1. If you want to develop/modify the virtual reality, we advise you to use Visual Studio 2010. A development project of the virtual reality could be started via unityVR/unityVR_xxx/unityVR_xxx.sln. If it does not exist, it will be generated by opening a script/asset in Unity. The main project description can be found under unityVR/doc/VR-doc/html/index.html .
2. Protobuf-C++ library (`http://code.google.com/apis/protocolbuffers/docs/overview.html`) is a fast and lightweight serializer, intended to use for network transfer. It can normally be found in the repositories of your linux distribution. For ubuntu 10.04, the necessary packages are libprotobuf-dev and protobuf-compiler. Do NOT install protobuf-c-compiler or libprotobuf-c0-dev, these are pure c version. To avoid troubles, install protobuf-2.5.0.
3. If you choose ANNarchy 4: Consult *agentBrain/networkAnnarchy4/python/readme.txt* in addition. Currently, the following steps should allow you to connect and execute e.g. *agentBrain/networkAnnarchy4/python/test_cases.py*
   a) Execute *agentBrain/networkAnnarchy4/recompileProtoc.sh* .
   b) Import the project in the *agentBrain/networkAnnarchy4/* directory into Eclipse and build it.
   c) Copy *agentBrain/networkAnnarchy4/Debug/libann4VrInterfaceCpp.so* to *agentBrain/networkAnnarchy4/libs/* .
   d) Enter the IP address to the VR at the end of test_cases.py and run this file via python command while the VR is running.
4. If you choose ANNarchy 2.3:

a) Compile your ANNarchy 2.3 application with network support. For this, execute recompileProtoc.sh and afterwards you must simply add PROTOC=1 after the scons command (see above), eg. > scons GUI=0 DEBUG=1 PROTOC = 1 OR call the installscript with - -protoc: > python script/installANNarchy.py - -protoc .

b) An example could be found in: *agentBrain/networkInterfaceCPP/Tests*. For this, you should use eclipse+cdt as IDE, since the directory contains an eclipse-project file (.project). You can start the example by first starting the unity-project, then click at run (the arrow) and finally compile & start the agent (c++ project).

4 Further Information

This chapter covers some more infomation and tips about working with Unity3D, the UnityVR and 3D objects.

4.1 Duplicate an Object of the Project

In the context menu of an item in the Unity3D Project View, there is no option to duplicate it. The hidden function to do this, is to select the item and press Strg-D.

4.2 Prefabs

Prefabs are types of assets. They are used to create a reusable *GameObject* which can be used in other scenes and to create instances at runtime. If you change properties in a prefab, it is automatically applied to all instances of that prefab.

4.2.1 Creating a Prefab

1. Create your desired object in the scene editor. Attach all components and scripts.
2. Create an empty prefab in the Project View (right click) and drag the object from the Scene View onto the empty prefab in the Project View.

4.2.2 Instantinate Prefabs at Runtime

The prefab must be located in the *Resources* folder, otherwise Unity3D can not find it at runtime.

As an example, the following code loads the *Lerpz* agent into the scene which is located as a prefab in the *Resources* folder:

```
Instantiate(Resources.Load("Lerpz") as UnityEngine.Object, new Vector3( 0 , 0, 0),
     Quaternion.identity);
```

For more information about the *Instantiate* function look at `http://docs.unity3d.com/Documentation/ScriptReference/Object.Instantiate.html`

4.3 Multiple Agents

In UnityVR it is also possible to put more than one agent into the scene. Each agent has its own TCP socket to communicate with its client. There are two ways to achieve this.

1. The first method is to create all the agents inside the Unity3D editor (like in 2.1.2) and change the *AgentID* of all attached *AgentScript*s to ascending integers (0, 1, 2, 3, etc.).

2. The second method is used in the Labyrith scene. It is also an example of using prefabs. Lerpz (the agent) is stored as a prefab in the *Assets/Resources*. It is than instantinated at runtime. The following code is taken from *Asserts/Scripts/LabyrithBehaviorScript.cs*. It spawns three instances of Lerpz at runtime.

```
protected override void AgentInitalization()
{
  const int numberOfAgents = 3;
  agents = new GameObject[numberOfAgents];

  for(int i = 0; i < numberOfAgents; i++)
  {
    //Position for agent x_i = (0,-20,20,-40,40,-60,60,....)
    float pos = Mathf.Pow( -1,i ) * 20 * Mathf.Ceil(i/2f);

    agents[i] = Instantiate(Resources.Load("Lerpz") as UnityEngine.Object, new
        Vector3( pos , 0, 0), Quaternion.identity) as GameObject;

    agents[i].GetComponent<LabyrinthAgentScript>().AgentID = i;

    //Get the Agent ready
    agents[i].GetComponent<LabyrinthAgentScript>().DefaultAgentPosition = new
        Vector3( pos , 0, 0);
  }


  // find and init all agent SCRIPTS  in the scene containing a specific script
  getAndInitAllAgentScripts<LabyrinthAgentScript>();

  //prevent that several cameras are rendered at top of each other: allow only
      first agent to display camera
  for(int i = 1; i < numberOfAgents; i++)
    agentScripts[i].setDisplayCameraVR(false);

}
```

4.4 Working with 3D Objects

4.4.1 3D Libary

In this library you will find objects which are not included in our UnityVR project. Our 3D library includes many Objects in different categories:

1. Accessories (common, pillows, veils, ...)
2. Blocks (Lego, cubes)
3. Cars
4. ChildVersion1 (this is our first version of the child felice)
5. Felice (the first animations of felice)
6. Neuron
7. Pens
8. Sweets

9. Toys

The pictures (.jpg) show you the preview of the objects. So you do not have to open every object. The files have the extensions .3DS, .blend and .fbx.
The .fbx format is the best format for exporting and importing objects. This format is the only one which supports the import of animations into Unity3D. For example, the maya (and 3DS) format do not support animations. Additionally, these formats should not even be used for simple 3D objects as they require the maya software for opening these objects.

4.4.2 Textures in Max 3D

In Max 3D you can change or rename the materials and textures of your objects. Also, you can increase or decrease them. The following three links will show how to work with textures and materials in Max 3D and how to export them.

- `http://www.youtube.com/watch?v=zLMv1KUy8ZQ`
- `http://www.youtube.com/watch?v=5lSANYBzQAI`
- `http://www.youtube.com/watch?v=VNaXrWcbQfw`