Julien Bruneau

[Email address]

Abstract

This is a detail documentation of the crowd experimental platform CrowdMP.

Crowd Experimental platform : CrowdMP

Table of Contents

[1. Overview 2](#_Toc4569218)

[A. Introduction 2](#_Toc4569219)

[B. Content 2](#_Toc4569220)

[2. How to use the platform 5](#_Toc4569221)

[A. Step by step guide to create your own basic experiment 5](#_Toc4569222)

[3. Base assets 8](#_Toc4569223)

[A. Scripts 8](#_Toc4569224)

[i. Organization 8](#_Toc4569225)

[ii. Manager 8](#_Toc4569226)

[iii. Loader 8](#_Toc4569227)

[iv. Agents 9](#_Toc4569228)

[v. Recorder 10](#_Toc4569229)

[vi. Tools 10](#_Toc4569230)

[vii. Workflow 11](#_Toc4569231)

[B. Configuration files 12](#_Toc4569232)

[4. Add-Ons 13](#_Toc4569233)

[A. Fove 13](#_Toc4569234)

[i. Using the Fove as a simple HMD 13](#_Toc4569235)

[B. MiddleVR 14](#_Toc4569236)

# Overview

## Introduction

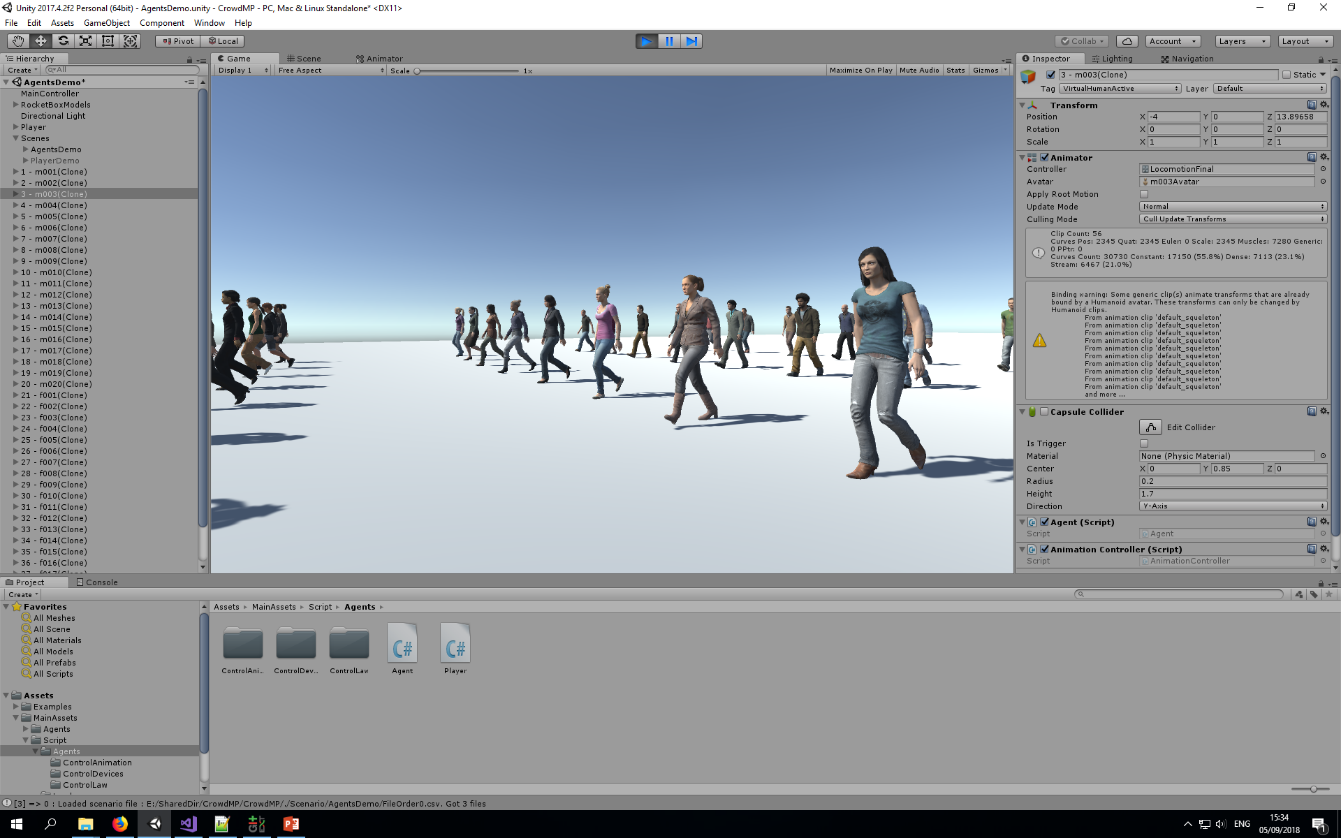


Figure 1: Unity view of the platform

CrowdMP is an experimentation platform. It is developed in Unity and his main purpose is to ease the implementation of experiments that require any kind of crowds. It provides a base to immerse participants in homemade environments populated with custom virtual crowds. It also provides many tools/plugins to customize as much as possible the experiment so it can be used for a wide variety of experiments that use virtual crowds. To this end, it offers many different features that can be useful when running experiments such as user controls, trials management, data recording, instructions presentation, etc. It also offers features specific to running experiments with crowds such as a variety of character models, animations and behaviors. Finally, it has been developed to ease the addition of new features in order to be adaptable to a wide range of experiments.

This document is a handbook explaining how to use the platform to create customized experiments. It starts with a guide to create new simple experiments step by step followed by a few practical examples. Then a complete list of the different configuration and trial parameters and their functionalities is provided. Finally, an in-depth technical documentation of the platform is given explaining the different mechanism and how to customize them by adding plugins. A few examples of how to create your own plugin/custom feature are also provided.

## Content

The content of CrowdMP is separated in two blocks. A first block is the base assets. The base assets contains the core of the platform that is shared amongst all users and should not be modify. The second block contains every plugins added by current or previous users to satisfy their specific needs and customize their experiment. The figure below show this separation at the code level:

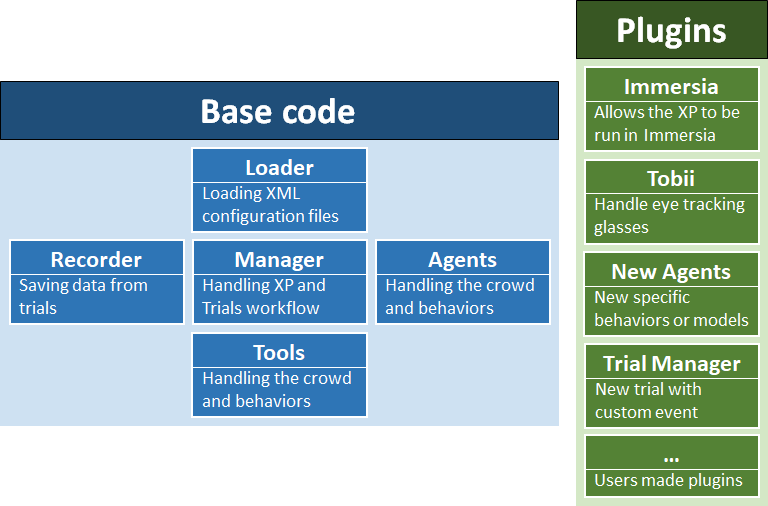


Figure 2: Code organization of the platform

The base assets provide several object to create simple experiments with different kind of crowds. Some of these objects are mandatory, other are optional and their use will depend on the experiment design. Here is a list of these objects (represented on Figure 3):

* A MainManager (Mandatory) => MainAssets/MainManager.prefab
  + A transition screen (Optional)
  + An ending screen (Optional)
* An instruction room (Optional) => MainAsset/Instructions/InstructionRoom.prefab
  + A room the size of Immersia (Optional)
  + A list of example panel instructions (Optional)
* Several agent models (Optional)
  + Several different controller and behaviors (Optional)
  + Animations (Optional)
* A few player models (Mandatory – at least one)
  + Several different controller and behaviors (Optional)
* A few example stages with
  + A choice of *TrialManager* (Mandatory)
  + A choice of *Recorder* (Optional)



Figure 3: Sample of objects provided by the platform

# How to use the platform

## Step by step guide to create your own basic experiment

Basic experiments that require no plugins can easily be created using CrowdMP. It is done in six steps (The following example scene is in the demo folder of the platform under the name “tuto”):

|  |  |
| --- | --- |
| **Step 1**  Mandatory: Create an empty new scene and add the MainManager prefab.  Optional: Modify or Delete the Transition/End panel to suit your requirement  Update: New public variable to select config file “ConfigPath” in MainManager |  |
| **Step 2**  Mandatory: Create your own stage and Tag the parent object with “Stage” Mandatory: Add a *TrialManager*  Optional: Add Walls and Pillars to the *TrialManager* for collision avoidance Optional: Add a *Recorder* |  |
| **Step 3**  Mandatory: Choose as many Player as you need Optional: Choose as many models for the agent as you need |  |
| **Step 4**  Optional: Add the instruction room prefab Optional: Add/Replace the instruction panel with your own instructions and sort them in the Instruction Trial Script List |  |
| **Step 5**  Mandatory: Generate Trial files for each trial (see format and parameters in section …) | <?xml version=**"1.0"** encoding=**"utf-8"**?>  <Trial xmlns:xsi=**"http://www.w3.org/2001/XMLSchema-instance"** xmlns:xsd=**"http://www.w3.org/2001/XMLSchema"**>  <scene meshName=**"Kerlann"**>  <Position x=**"0"** y=**"0"** z=**"0"** />  <Rotation x=**"0"** y=**"0"** z=**"0"** />  <recordingFile>**Output\_{USER}\_{ITT}.csv**</recordingFile>  <endingConditions>  <condition parameter=**"time"** test=**"greater"** value=**"60"** />  </endingConditions>  </scene>  <player>  <TrialRegularPlayer mesh=**"RegPlayer"** >  <Position x=**"0"** y=**"0"** z=**"0"** />  <Rotation x=**"0"** y=**"0"** z=**"0"** />  <controlLaw>  <LawControllerSpeedAngle speedCurrent=**"0"** speedDefault=**"1.33"** speedVariation=**"0.5"** angularSpeed=**"30"** accelerationMax=**"0.8"** timeBeforeControl=**"0"** />  </controlLaw>  </TrialRegularPlayer>  </player>  <SavedDataList>  <data>  <DataUnitySpatial />  </data>  </SavedDataList>  <agents>  <agent>  <TrialRegularAgent mesh=**"f014"** animationOffset=**"0.223"**>  <Position x=**"0"** y=**"5"** z=**"0"** />  <Rotation x=**"0"** y=**"0"** z=**"90"** />  <controlLaw>  <LawMoveStraight speedCurrent=**"1.33"** speedDefault=**"1.33"** accelerationMax=**"0.8"** />  </controlLaw>  <controlSim>  <RVOconfig SimulationID=**"0"** neighborDist=**"5"** maxNeighbors=**"20"** timeHorizon=**"2"** timeHorizonObst=**"0"** radius=**"0.33"** maxSpeed=**"2"** />  </controlSim>  </TrialRegularAgent>  </agent>  </agents>  </Trial> |
| **Step 6**  Mandatory: Generate the Config file (see format and parameters in section …)  Mandatory: Create a sorted list of the Trial Files in a csv file for each users (corresponding to the sourceFileExperiement in the config file) | <?xml version=**"1.0"** encoding=**"utf-8"**?>  <Config xmlns:xsi=**"http://www.w3.org/2001/XMLSchema-instance"** xmlns:xsd=**"http://www.w3.org/2001/XMLSchema"**>  <experience currentUser=**"0"** startingTrial=**"0"**>  <sourceFileExperiment>**./Scenario/FileOrder{USER}.csv**</sourceFileExperiment>  <userHeight>**1.7**</userHeight>  </experience>  <Recording dataSeparator=**","** decimalSeparator=**"."**>  <folder>**./Output/**</folder>  </Recording>  <log debugLvl=**"0"** />  <AddOns>  <AddOn>  <ConfigTobii isUsed=**"false"** Interface=**"fe80::8c3a:5558:db25:1436"** glassesIP=**"fe80::76fe:48ff:fe25:2368"** discoverPort=**"13006"** dataPort=**"49152"** />  </AddOn>  <AddOn>  <ConfigVicon isUsed=**"false"** />  </AddOn>  </AddOns>  </Config> |

# Base assets

## Scripts

### Organization

The base code is divided into 5 parts as represented by the pictures below.

**Base code**

**Agents**

Handling the crowd and behaviors

**Loader**

Loading XML configuration files

**Manager**

Handling XP and Trials workflow

**Recorder**

Saving data from trials

**Tools**

A variety of useful functions

### Manager

The Manager block contains all the class controlling the experiment and trials flow. It is the central block as it links all the other blocks together to run the experiment. Here is a list of the classes found in the Manager block:

* *MainManager.cs:* Loads trial in the right order with transition/ending screens.
  + *ManagerCustomEditor:* Customize unity editor to add the button "Create Template Config Files" in the MainManager script.
* *TrialManagers.cs:* Interface for manager controlling the proceeding of a Trial.
  + *InstructionsTrial.cs:* Trial manager allowing participants to navigate through a list of instructions.
  + *RegularTrial.cs:* Trial manager allowing participants and agents to navigate according to their controlLaw/Simulation.
  + (…)
* *PluginManager.cs:* Interface that should be implemented by most plugin that required to be setup at the beginning of the XP.

### Loader

The Loader block contains all the classes that handle the loading of parameters from XML files. Most XML serializable classes are in the Loader block but some are place with the object they parametrize. For example the XML serializable class containing parameters for an agents is in the same place as the agent class. Here is a list of the classes found in the Loader block:

* *LoaderXML.cs:* Read/Write classes from/to XML files.
* *CustomXmlSerializer.cs:* Class to encapsulate Interfaces/Abstract Class implementation and allow easy XML serialization/deserialization of them.
* *LoaderConfig.cs:* Load the experiment parameters.
* *LoaderXP.cs:* Load the current trial parameters.
* *IConfigExtra.cs:* Interface for plugin parameters.

### Agents

The Agents block contains all the classes that define and control an agent. Agent is the general term to define a moving and interactive character, whether it is control by a participant or not and whatever is model (Human, robot, abstract, etc.). Here is a list of the classes found in the Agents block:

* *Agent.cs:* Abstract class for any agent of any type.
  + *Player.cs:* Abstract class for player agents.
    - *RegularPlayer.cs:* Implementation for a player without body, using a control law to navigate through the virtual environment and can be override by a crowd simulator.
  + *RegularAgent.cs:* Implementation for a simple agent following a control law and a crowd simulator. Can navigate through environment and avoid collision.
* *ControlLaw.cs:* Interface for classes computing the goal of an agent (That can be override by a crowd simulator).
  + *LawControllerSpeedAngle.cs:* Implementation using a device to influence speed and direction of movement.
  + *LawControllerSpeedSideStep.cs:* Implementation using a device to influence speed of forward and side movement, no rotation is done.
  + *LawFileData.cs:* Implementation that load a trajectory from a csv file (Format: time,x,y) to control the agent
  + *LawGoals.cs:* Implementation that defines a list of goals and uses unity built-in navmesh feature to find a path to the goals in order and can looped through the goals.
  + *LawMoveStraight.cs:* Implementation that defines a straight and infinite movement, no rotation is done.
  + *LawStanding.cs:* Implementation that does not move the agent but play a standing animation from different types. (0 : No animation,1 : idle, 2 : talking, 3 : applause, 4 : look around)
  + *LawWaypoints.cs:* Implementation that defines a list of waypoint that agent has to reach in order and that can be looped.
* *Simulations.cs:* Class that manages all the running crowd simulators and reports the results to the concerning agents.
  + *ControlSim.cs:* Interface for classes controlling a crowd simulator.
    - *SimRVO.cs:* Implementation to control the RVO simulator.
      * *RVOConfig.cs:* XML serializable class containing RVO parameters.
  + *Obstacles.cs:* Class that define and list the different obstacles of a trial stage so it can be inserted to the crowd simulators.
* *AnimationController.cs:* control the animation of the virtual human models (**ToDo**: create interface for other models).

### Recorder

The Recorder block contains all the classes that handle the recording of the data studied during the experimentation. Here is a list of the classes found in the Recorder block:

* *Recorder.cs:* Interface for classes handling data recording
  + *RegularRecorder.cs:* Implementation that save Spatial data and device state at each step of the experiement
  + *ConfigurableRecorder.cs:* Implementation that can be configure with a list of data to save.
* *RecorderData.cs:* Interface that define a type of data and how to record it for the ConfigurableRecorder data list.
  + *DataUnitySpatial.cs:* Implementation for the position and rotation of player and agents
  + *DataDeviceState.cs:* Implementation for the state of the player’s device

### Tools

The Tools block contains a variety of useful tools. Some of these tools are enclose in a custom class so that it can easily be adapted to different platform (Screen/Immersia/HMD/android). Here is a list of the classes found in the Tools block:

* *ToolsCamRecorder.cs:* Tool to record pictures of the screen at specific Framerate to create video.
* *ToolsDebug.cs:* Handle all the debug messages from error messages to simple logging.
* *ToolsGeneral.*cs: Handle small useful functions.
* *ToolsInput.cs:* Handle the master input used by the experiment runner during the experiment.
* *ToolsOutput.cs:* Handle the creation and writing of output files.
* *ToolsTime.cs:* Handle the different time used during an experiment.

### Workflow

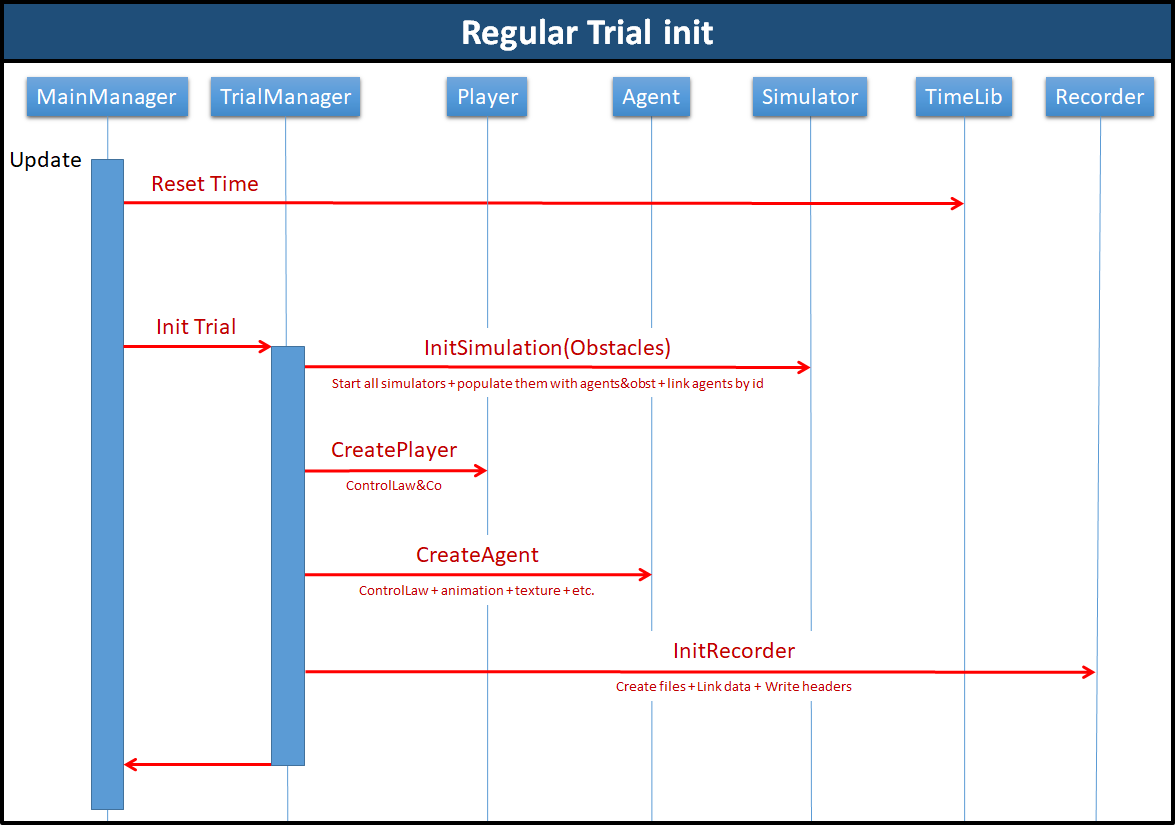


Figure 4: Workflow during trial initialization

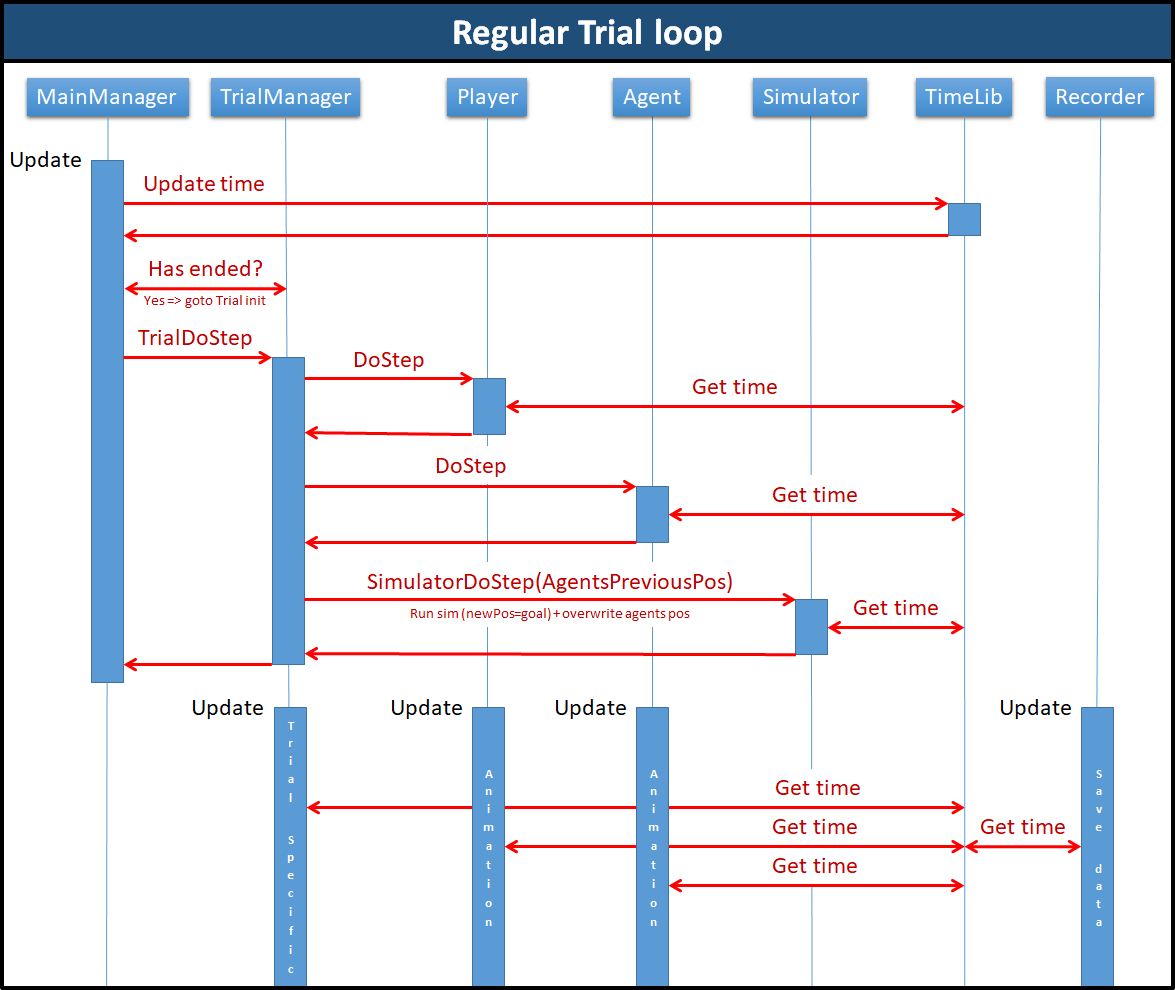


Figure 5: Workflow during a regular trial main loop

## Configuration files

### Files structures

# Add-Ons

## Fove

The Fove add-on enables the experiment to be run in the Fove HMD. It also offers tools to exploit the gaze of a participant to use it in a trial and/or to record it for future analysis.

### Using the Fove as a simple HMD

To enable the Fove plugin to work in your experiment, 3 steps are mandatory:

|  |  |
| --- | --- |
| **Step 1**  Mandatory: Add the Fove Rig to the player at the same level as the Main Camera OR use one of the Fove player prefab. Both contains the required FoveInterface script. |  |
| **Step 2**  Mandatory: Add FoveManager to MainManager.PluginManager and link the FoveRig to it. |  |
| **Step 3**  Mandatory: Active the FovePlugin in the config.xml file. (Blue line in example) | <?xml version=**"1.0"** encoding=**"utf-8"**?>  <Config xmlns:xsi=**"http://www.w3.org/2001/XMLSchema-instance"** xmlns:xsd=**"http://www.w3.org/2001/XMLSchema"**>  <experience currentUser=**"0"** startingTrial=**"0"**>  <sourceFileExperiment>  **./Scenario/Rennes/FileOrder{USER}.csv**  </sourceFileExperiment>  <userHeight>**1.7**</userHeight>  </experience>  <Recording dataSeparator=**","** decimalSeparator=**"."**>  <folder>**./Output/**</folder>  </Recording>  <AddOns>  <AddOn>  <ConfigFove isUsed=**"true"** eyeDirectorIsRight=**"true"**/>  </AddOn>  </AddOns>  </Config> |

## MiddleVR

The MiddleVR plugin allows an experiment to run using a MiddleVR configuration. Its main purpose it to run the experiment in Immersia or ImmerMove but it can be used for other devices supported by MiddleVR. Unlike other add-on, the MiddleVR plugin cannot be activated/deactivated through the config file but needs to be activated during when compiling the executable. To enable MiddleVR plugin to work in your experiment, 3 steps are mandatory:

|  |  |
| --- | --- |
| **Step 1**  Mandatory: Add the VRManager prefab to the scene.  Optional (if only one player): Change VR System Center Node variable to your Player and deactivate the VRPlayerDummy game object  Optional: Choose a camera for the Template Camera variable |  |
| **Step 2**  Mandatory: Add **MIDDLEVR** as a scripting define symbols in player settings (can be found in the File -> build setting) |  |
| **Step 3**  Mandatory: Make sur you have a config.vrx file in the project directory (MiddleVR config file) |  |