05/01/2019

ÉCOLE NATIONALE SUPÉRIEUR
D'ÉLECTROTECHNIQUE, D'ÉLECTRONIQUE, D'INFORMATIQUE
D'HYDRAULIQUE ET DES TÉLÉCOMMUNICATIONS
INSTITUT NATIONAL POLYTECHNIQUE

INSTITUT TOULOUSAIN D'OSTÉOPATHIE

Technical Guide: Osteo - Android

Projet Long 2019:
Cervical pathology detection thanks to virtual reality

Members : Txomin Itoiz, Clément Brunie, Thibaud Ishacian, Lucien Haurat, Clément Guillaumin, Edmond Boulet-Gilly
Project Manager: Edmond Boulet-Gilly
Industrial Supervisor: Jean-François Coiffin
Client: Denis Ducommun

Version 1.0

Introduction 2

Prerequisites and target platform 2

Compile to .apk 3

General Architecture 3

Main Menu 4

Scene 4

Test parameters 5

Profile 6

Files 7

VR Environment 8

Scene 8

Target 8

CountDownText 9

Player 9

ExitObject 9

Notes 10

Introduction

This paper is the technical guide to the Osteo Android application, developped during the 2019 “Projet Long” by the members named hereinabove. The application allows to launch a virtual reality test with a target moving horizontally, and to record the head movement of the user. The head movements are then saved locally and can be sent by email, the idea being that this data should be used by another piece of software to detect cervical pathologies.

The VR environment consists in following the target, moving in a semi-circle to the right and to the left at a constant speed, pausing at each end of its course.

Prerequisites and target platform

The development and the compilation have been done on Unity 2018.3.2f1, and as the target platform is Android, it is necessary to install Android Studio to use the Android SDK. Unity should detect the latter in order to work correctly.

The application also needs some third-party packages (included in the provided source code):

- *GoogleVR* package for unity
- *Textmesh Pro* (included in most of Unity distributions)

Compile to .apk

Once the prerequisites are met, it is possible to open the project with the Unity Launcher, selecting the “VRApp” directory.

After the project is open, go to File > Build Settings, and select Android. You will the have the choice to only compile to a .apk file, or to compile it and run it on the phone connected to the PC. In all cases, it is possible to install manually the .apk file after the compilation.

General Architecture

The application is split into 2 scenes: one scene for the main menu “MainMenu”, and one scene for the VR environment “App”. The scripts responsible for changing scene from the menu to the VR environment are *SceneLoader* and *VREnabler*, and the ones responsible for the VR environment to the main menu are *LoadSceneVR* and *VRDisabler*.

Moreover, some arguments are passed from one scene to another. The scripts responsible for that action are located in /Asset/ParameterPasser

The *MenuScript* and *VRScript* directories contain respectively the scripts dedicated to the menu, and the ones dedicated to the VR environment.

The *Resources* directory contains the Prefab (Unity prefabricated entities), necessary to the manu. It is the only directory which is imperative you don’t mix with the others, the *FilePanel* script relies on its location.

Main Menu

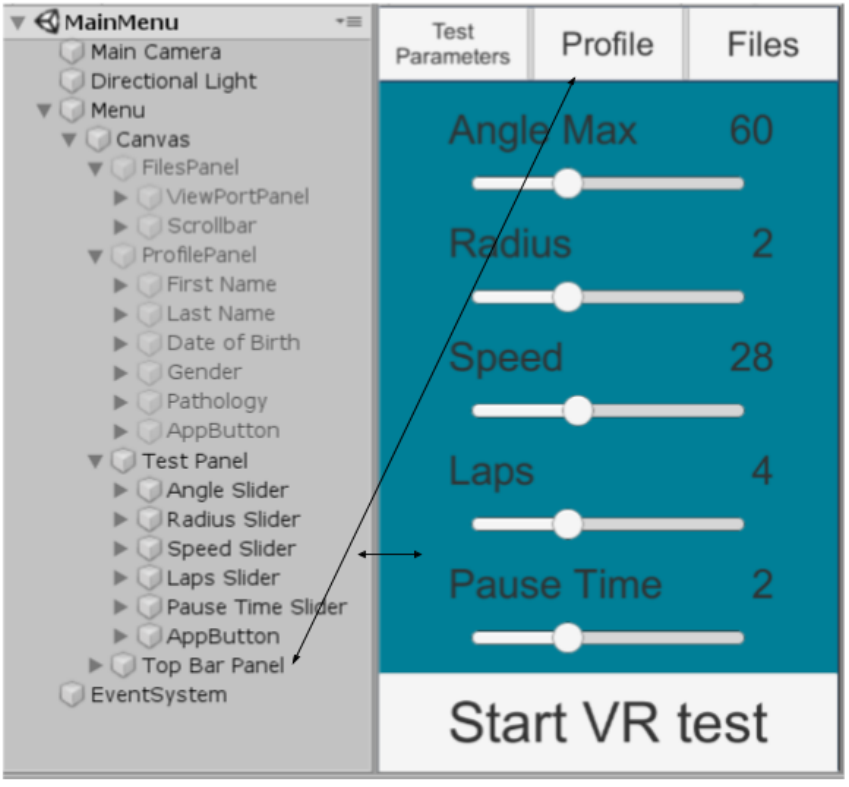
Scene

The main menu leads to 3 submenus thanks to the 3 tabs at the top, then to start the test with the button on the bottom: “Start VR Test”.

The main menu scene is composed of a *Menu* entity, parent of a *Canvas*, the tabs at the top *Top Bar Panel*, and the 3 submenus *Test Panel*, *ProfilePanel*, and *FilesPanel*, related respectively to the *Test Parameters*, *Profile*, and *Files* tabs.

The *Top Bar Panel* buttons activates and deactivates ent menu entities with Unity events specific to each button.

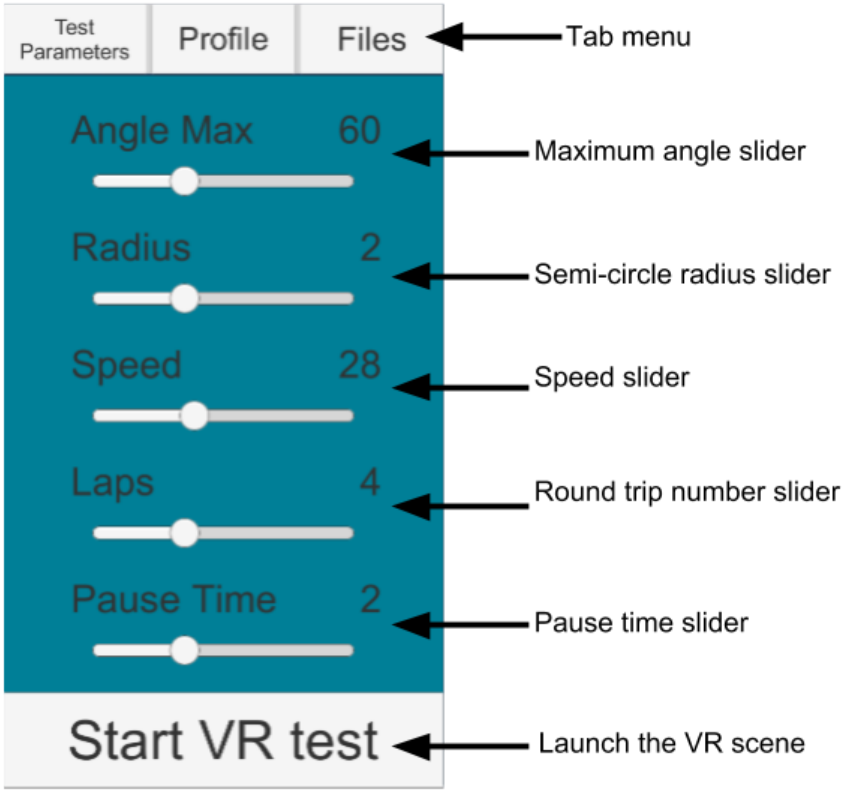
Only the first 2 submenus have an *AppButton*, which allows to switch to the second scene.



Test parameters

The first tab, “Test Parameters”, is composed of multiple sliders, each with its own specific script:

- Angle Max: *AngleMaxPasser* writes the given value to “*AngleMax*”
- Radius: *RadiusPasser* writes the given value to “*Radius*”
- Speed: *SpeedPasser* writes the given value to “*Speed*”. The displayed value isn’t the same passed to the other scene, because of angular speed conversion, and that the slider entity doesn’t support well enough decimal numbers.
- Laps: *MaximumLapsPasser* writes the given value to “*MaximumLaps*”
- Pause Time: *PauseTimePasser* writes the given value to “*Pause Time*”

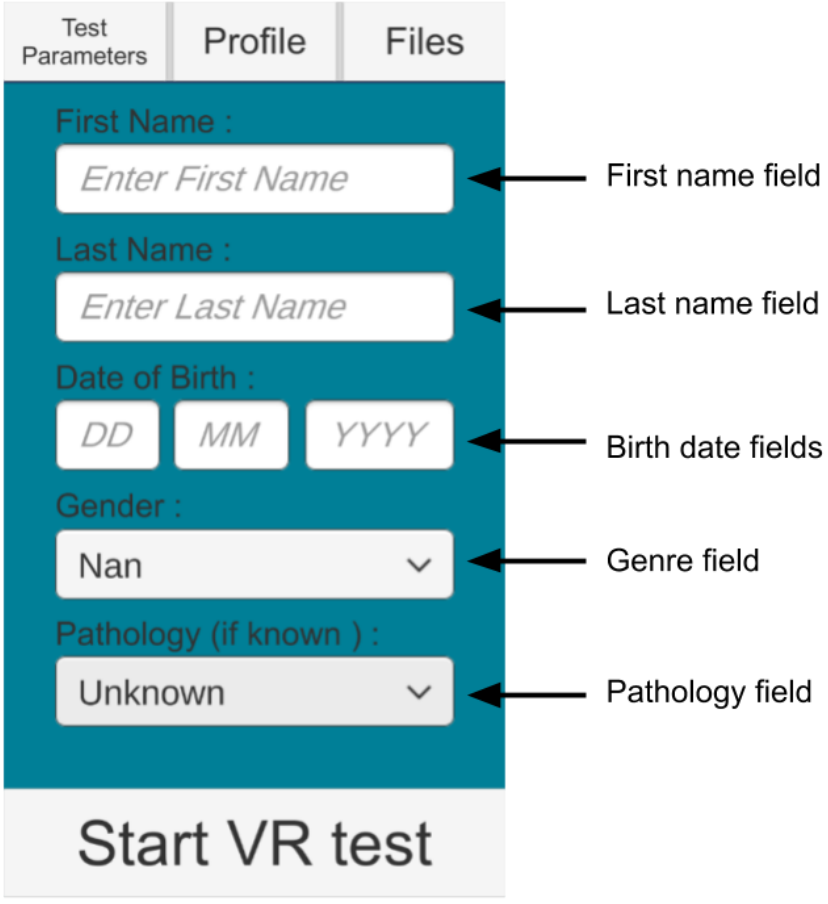


Profile

The second tab asks the user about personal information, and all are passed as a string with the *StringPasser* script:

- The first name writes the given value into "*FirstName*"
- The last name writes the given value into "LastName"
- The birth date writes the given value into "*DayBirth*", "*MonthBirth*" et "*YearBirth*"
- Genderwrites the given value into "*Gender*"
- Pathology(if known) writes the given value into "*Phatology*"

Because the genre and the pathology are not fields but drop-down menus, they need the *DropDownHelp* script to read the given value. If the user doesn't fill these fields, empty string will be used by the VR scene.

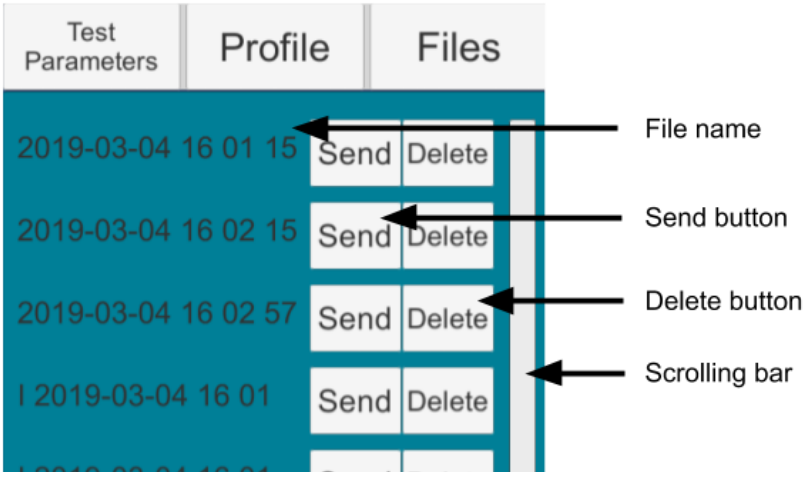


Files

This tab shows a list of locally stored files, sorted by name. They are labelled by an “I” if the recording was interrupted. This menu have a vertical scrolling bar, and some space to show every file name and buttons. The script responsible for showing the entity list to interact with the files is *FileListSpawner*. The scrolling bar allows to go through all the files if the complete list cannot fit in one screen.

The buttons use the *DataTool* script:

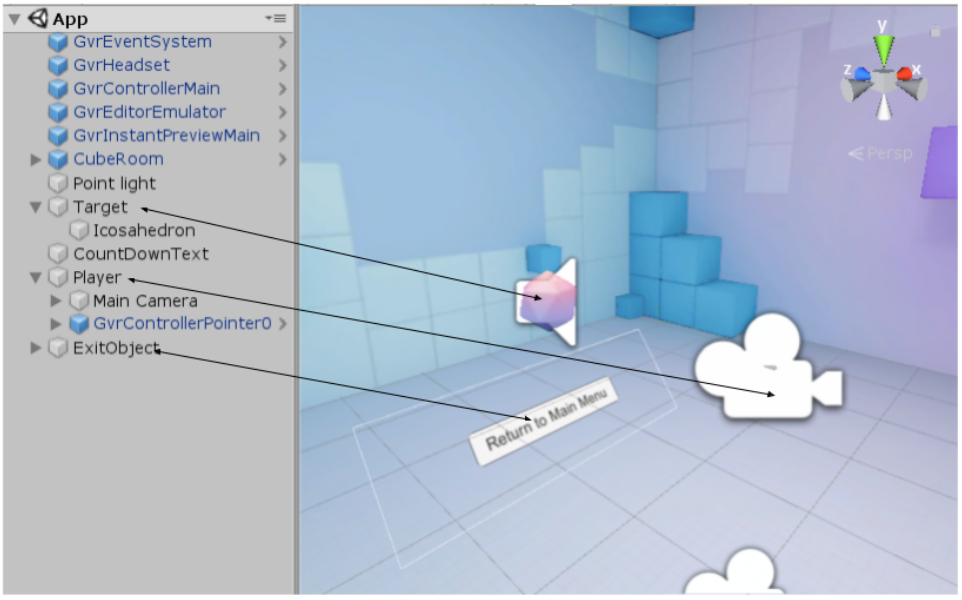
- The *Send* button creates an email with the data as the body. To do this, the application uses the OpenURL(“mailto:...”) function.
- The *Delete* button deletes the data from the phone (thus from the list too).
  The entities are made from Prefabs, you can find it in “*Resource/DataItem*”.



VR Environment

Scene

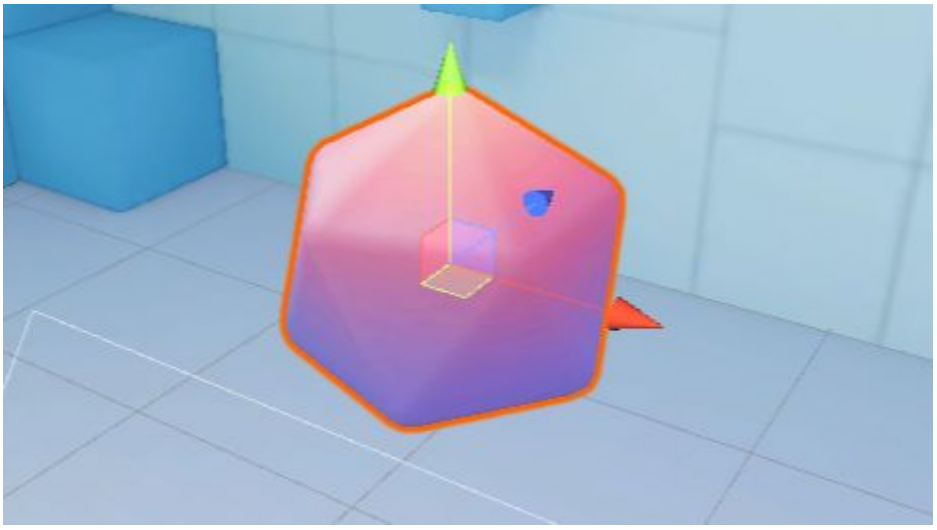
After pressing the “Start VR Test” button, the second scenes loads. It is possible to load it manually, by double-clicking on it in the editor explorer. In either case, as long as you are in the editor, the VR environment will be loaded as one window, and not 2, one for each eye. This scene is simply a copy of the one provided by *GoogleVR* to which entities and scripts have been added.



Target

Entity representing the target the user will follow. It uses the *MoveTarget* script for the target movement, the path computation, and the pause times at each end of its path. This script requires the *MoveCapture* script to notify the initialization, interruption, cancelation, or completion of the test.

The Icosahedron entity is the target in the 3D environment. Using the Unity event system, the Icosahedron changes color when the user is watching it, and notifies the *CountDown* and *MoveTarget* scripts.



CountDownText

Entity displaying the countdown before launching the test. It uses the *CountDown* script, for the displaying of it, and to initiate the target movement start. It needs to access *MoveTarget* to notify the entity of the countdown end, and start the movement.

Player

Entity linked to the player perspective. It uses the *MoveCapture* script, responsible for the capture of the head movements. The data format was made to be homogeneous to the Windows application developped during the 2018 Projet Long of ENSEEIHT.

The data is written in the persistent directory of the application, which is platform specific, in a text file format.

ExitObject

Entity allowing the user to go back to the main menu from the VR environment. It uses the *SceneLoader* and *LoadSceneVR* scripts, responsible for the initialization, the countdown, cancel it if the user stops watching, and change scene when the countdown ends.

Notes

- It is possible that the user uses letters instead if digits for his birth date.
- It is possible to join the scripts passing test arguments and profile to avoid similar scripts which are duplicated.
- A round trip between the main menu and the VR scene clears the profile data.
- The data is send in the body of the email, not as a attached file.
- In the VR environment, there is 4 scripts whose *Update()* method is called at each frame. To decrease frame computation time, you would need to write something to dynamically activate and deactivate those scripts.