Sistema inmersivo para la tutorización virtual con Oculus Go ID2018/080

PROGRAMA DE MEJORA DE LA CALIDAD – PLAN ESTRATEGICO GENERAL 2013-2018

MEMORIA DE JUSTIFICACIÓN

26 de junio de 2019

Autor: Gabriel Villarrubia Gonzalez

Índice

Sistema inmersivo para la tutorización virtual con Oculus Go ID2018/080

PROGRAMA DE MEJORA DE LA CALIDAD – PLAN ESTRATEGICO GENERAL 2013-2018

Miembros del equipo de trabajo:

NIF	Nombre y apellidos	E-mail
76125754D	Juan Francisco De Paz Santana (Profesor)	fcofds@usal.es
12420865Z	Vivian Félix López Batista (Profesor)	vivian@usal.es
70875798X	Alberto López Barriuso (Colaborador)	albarriuso@usal.es
08104619V	Juan Ramón Muñoz Rico (Profesor)	rico@usal.es
70900083F	Daniel Hernández De La Iglesia (Colaborador)	danihiglesias@usal.es
22758868T	Diego Manuel Jiménez Bravo (Becario FPI)	dmjimenez@usal.es
70909032D	Lucía Martín Gómez (Alumno Doctorado)	luciamg@usal.es
80098605R	Álvaro Lozano Murciego (Becario FPI)	loza@usal.es
71702446S	Javier Caridad Hernández (Becario Junta)	jch@usal.es
78508409W	Yanira Navarro Marrero (Becario FPI)	marnayan@usal.es
16610056P	David Peral García (Alumno Grado Ingeniería Informática)	daveral@usal.es
7833143X	Pastora Isabel Vega Cruz (Profesor)	pvega@usal.es
Y5601546W	André Filipe Sales Mendes (Personal Investigador)	andremendes@usal.es
70903746J	María Navarro Cáceces (Profesor)	maria90@usal.es

Motivación de este trabajo:

La Unesco calcula que, en 2025, la demanda de educación, únicamente universitaria, se va a incrementar en unos 80 millones de personas. Los expertos señalan que la única solución viable para cubrir estas necesidades pasa por asociarla a la educación digital. Ante esta previsión se están implementando y buscando soluciones basadas en las posibilidades que ofrece la tecnología y que, según fuentes de Telefónica Educación Digital, la división especializada en soluciones de e-learning de Telefónica, permiten desarrollar programas de transformación tecno-pedagógicas y avanzar hacia nuevas modalidades de capacitación. Se ha de mencionar que la línea de Telefónica Educación Digital tiene un volumen de más de cuatro millones de alumnos en todas sus plataformas, como Miríadax o Scolartic, y es una de las sociedades que apuestan por proporcionar nuevas herramientas de formación corporativa, educación digital para jóvenes y docentes, y educación digital en el formato de cursos a distancia.

Los avances tecnológicos de los últimos años han modificado el trabajo, los hábitos de consumo y de ocio, la manera de comunicarnos e informarnos y también están impactando, cada vez más, en los modelos de aprendizaje. De la impresión 3D hasta los juegos digitales pasando por los cursos on line (MOOCs), las aplicaciones como Skype y las capacidades del Big Data –que permiten hacer un seguimiento de todo tipo de aspectos educativos– se ha llegado a la implementación de la realidad virtual (VR) que favorece estudiar con un menor esfuerzo, al basarse en la creación de experiencias [1].

Ilustración 1. Ejemplo De Clase Construida en 3D

El objetivo más importante de este proyecto ha consistido en crear un aula virtual e inmersiva mediante la utilización del dispositivo Oculus-Go que permita al estudiante aumentar el contenido académico de una actividad o un tema en cualquier momento y de forma remota. No se ha pretendido sustituir las explicaciones del profesor, sino más bien, crear una herramienta que sirva como complemento a alumnos, que buscan resolver algún tipo de duda o estar presente en una clase a la que no han podido acudir por algún motivo o que necesiten su repetición. Se ha diseñado un escenario mucho más inmersivo que una clase

tradicional cuya experiencia se ha aproximado mediante la utilización de sensores y la utilización de realidad virtual. Los miembros del equipo y una vez finalizada la etapa de pruebas, hacen constar que la utilización del prototipo diseñado favorece ciertas dificultades que surgen en el aprendizaje, ya que este tipo de iniciativas hacen que las clases sean mucho más interesantes enganchando a los estudiantes y haciendo que sus contenidos sean mucho más tractivos. El alumno o usuario puede acudir a su clase de la facultad de forma virtual y de una forma personalizada, con vídeos y pruebas que permitan un aprendizaje interactivo de forma transparente y sin dificultad.

Ilustración 2. Utilización De Un Asistente Virtual

En el mundo virtual diseñado en este proyecto de innovación docente, se han utilizado las últimas técnicas de streaming de vídeo con el objetivo de poder impartir una clase (evento, ponencia) que se esté celebrando en la facultad de forma física mediante la utilización de una cámara de tipo 360º de tal forma que si un alumno no puede desplazarse a clase (enfermedad, viaje, etc.) pueda recibirla en el momento que el estime oportuno como si estuviese presente. Para facilitar la adquisición y que este proyecto pueda ser utilizado por la mayoría de los usuarios, se ha implementado una solución basada en el dispositivo Oculus-GO, que fue lanzado al mercado para su venta al público durante el proceso de solicitud de este proyecto de innovación. La ventaja principal de este proyecto es que este dispositivo es totalmente autónomo, no necesita estar conectado a un ordenador de alta prestaciones y su coste es inferior a los 250€ [4], siendo uno de los dispositivos de realidad virtual que van a marcar un antes y un después en el desarrollo de aplicaciones inmersivas.

[1]https://www.elperiodico.com/es/formacion/20171204/el-impacto-de-la-realidad-virtual-en-la-nueva-educacion-6461200

[2] https://www.scolartic.com/inicio

[3] LA REALIDAD VIRTUAL, UNA TECNOLOGÍA EDUCATIVA A NUESTRO ALCANCE Emilio R. Escartín

[4] https://www.oculus.com/go/

Objetivos conseguidos durante la realización de este proyecto:

#	OBJETIVO	DESCRIPCIÓN
01	Especificaciones y definición de la herramienta.	*Descripción*: Concepción y elaboración del diseño del sistema inmersivo para la tutorización virtual con Oculus Go. *Resultado*: Diseño y elaboración de los objetivos del sistema inmersivo para la tutorización virtual con Oculus Go.
01.1	Especificación de requisitos funcionales.	*Descripción*: Elicitación de los contenidos funcionales que la herramienta debe cumplir en el proceso de desarrollo. *Resultado*: Conjunto de requisitos funcionales.
01.2	Especificación de requisitos no funcionales y de interoperabilidad en una clase.	*Descripción*: Análisis de los requisitos necesarios para el correcto desarrollo de la herramienta y su incorporación en una clase. *Resultado*: Conjunto de requisitos no funcionales y medidas para la correcta incorporación del sistema en cualquier clase.
02	Investigación de técnicas.	*Descripción*: Investigación de técnicas para la elaboración de la herramienta. *Resultado*: Técnicas apropiadas la elaboración de la herramienta.
02.1	Investigación de las técnicas de realidad virtual	*Descripción*: Análisis de las técnicas existentes para la elaboración de aplicaciones de realidad virtual. *Resultado*: Estado del arte de técnicas de realidad virtual.
02.2	Investigación y estudio del SDK de Oculos GO	Descripción: Análisis del SDK y las funcionalidades disponibles del dispositivo Oculos GO. Resultado: Conocimiento detallado de todas las posibilidades disponibles del dispositivo.
02.3	Investigación y estudio de cámaras 360 grados	Descripción: Análisis de las diferentes cameras de 360 grados disponibles en el mercado. Resultado: Elección de la camera que disponga de los requisitos necesarios para el funcionamiento del sistema.
03	Elaboración de la herramienta.	*Descripción*: Desarrollo del software para el funcionamiento del sistema. *Resultado*: Software necesario para la integración de los diferentes componentes del sistema.
03.1	Desarrollar el software para las Óculos GO que permita asistir a una clase mediante un entorno virtual.	Descripción: Programación de la aplicación encargada de proporcionar al usuario una clase en un entorno virtual.

		Resultado: APK de la aplicación que se instalará en los diferentes dispositivos que pretendan asistir a la clase de manera virtual.
03.2	Creación de diferentes entornos virtuales para la simulación de diversos escenarios.	*Descripción:* Diseño y modelado de diferentes entornos virtuales. *Resultado:* Conjunto de salas en entornos virtuales que pueden ser usadas para el simulado de las clases.
04	Instalación y entono de pruebas.	Descripción: Instalación de la aplicación en las Oculus GO y generación de contendido multimedia de pruebas. *Resultado*: Versión alfa del sistema, para el testeo y realización de pruebas con contenido multimedia de prueba.
04.1	Generación de contenido multimedia de una serie de clases.	*Descripción*: Diseño del contenido multimedia para la transmisión de una clase en el entorno virtual. *Resultado*: Contenido multimedia para pruebas.
04.2	Instalación de la aplicación en las Oculus GO para las pruebas.	*Descripción*: Instalación de la aplicación en las Oculus GO. *Resultado*: Oculus GO con la aplicación preparada para la asistencia de una clase virtual.
04.3	Detección de posibles errores.	*Descripción:* Ensayo del funcionamiento de la aplicación, para hallar posibles defectos. *Resultado:* Estado de la aplicación para realizar posibles correcciones de errores.
04.4	Implementación de posibles mejoras en la usabilidad de la aplicación.	*Descripción*: Percepción de posibles mejoras que aumenten las características del sistema. *Resultado*: Incremento de la funcionalidad del sistema.
05	Aplicación de la herramienta en escenario real	Descripción: Comenzar pruebas en cursos reales en las titulaciones de ingeniería. Resultado: Resultados general y amplio es curso académico.
05.1	Evaluación de la herramienta en un caso de estudio real.	*Descripción*: Evaluar el rendimiento en las asignaturas y de la herramienta. *Resultado*: Obtención de resultados en asignaturas de plan académico.
05.2	Medición del impacto mediante indicadores de eficiencia.	*Descripción*: Medir si la herramienta presenta una eficiencia en el rendimiento de los alumnos. *Resultado*: Obtención de resultados académico gracias a la implantación del proyecto.
05.3	Obtención de resultados obtenidos mediante el empleo de la aplicación.	*Descripción*: Generar resultados de compromiso de los estudiantes con los planes de estudio para el análisis de resultados académicos en entregas.

		Resultado: Documentación de la evolución de la mejora en las técnicas de estudio aplicadas por los alumnos.
06	Difusión de los resultados.	*Descripción*: Difundir los resultados obtenidos gracias a la implantación de la herramienta en el campus. *Resultado*: Difusión general y científica del empleo del sistema.
06.1	Recogida y evaluación de la implantación del sistema.	*Descripción*: Evaluación de todos los resultados obtenidos mediante el desarrollo e implantación del sistema. *Resultado*: Documentar todos los resultados obtenidos a nivel de usabilidad, desarrollo software, empleo de los alumnos y beneficios académicos.
06.2	Publicación de los resultados obtenidos en revistas científicas.	*Descripción*: Exposición de los resultados obtenidos en Congreso y revistas del proyecto para la difusión de los beneficios de este tipo de herramientas. *Resultado*: Publicación de artículo en congreso internacional.

Temporización de las tareas realizadas:

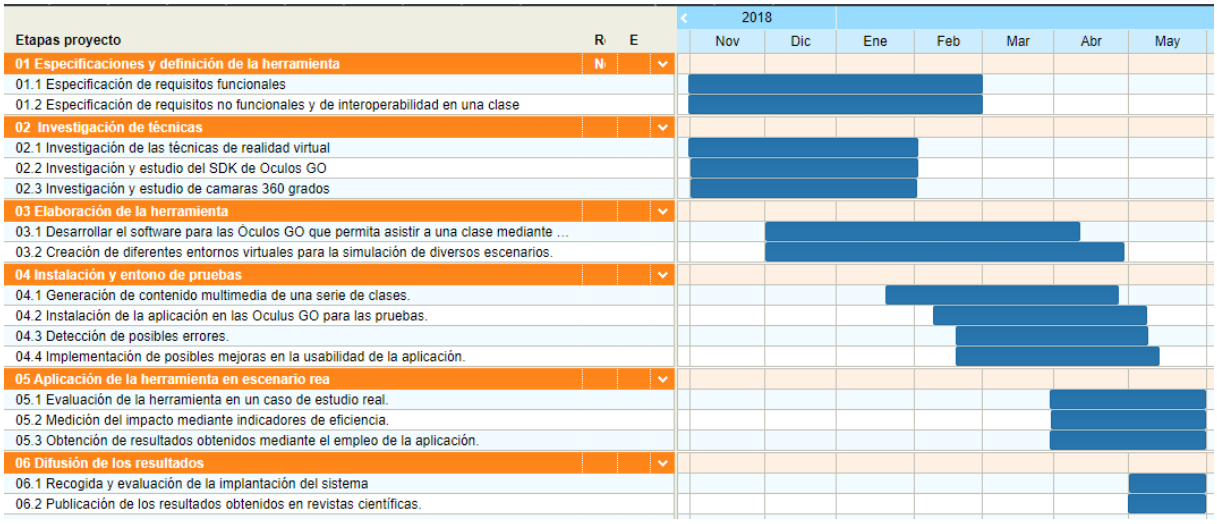


Ilustración 3. Temporización de Tareas

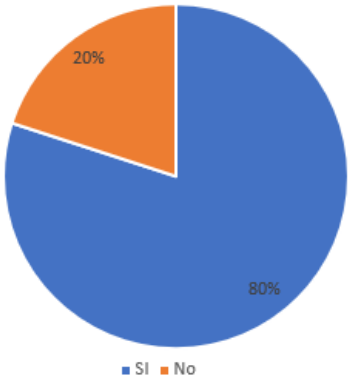
Punto de partida, encuesta a estudiantes:

Para justificar la necesidad de implantación de una herramienta de educación inmersiva es de vital importancia conocer la opinión de profesores y alumnos acerca de los métodos de enseñanza tradicionales que son llevados a cabo en el centro. Es por ello por lo que durante la realización de este proyecto se ha realizado una encuesta anónima y con carácter voluntario a personal docente y alumnos de las carreras de ingeniería con un total de 43 participantes. A continuación, se listan las preguntas que se hicieron y los resultados para cada una de ellas.

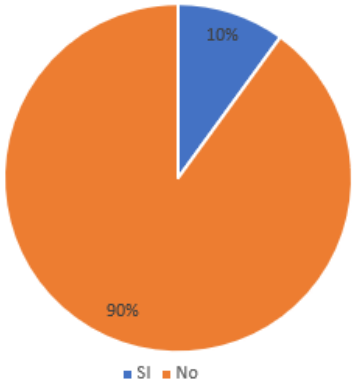
1. ¿Conoces algún profesor que utilice algún sistema basado en realidad virtual para impartir algún contenido académico en la Facultad de Ciencias?
2. ¿Has probado alguna vez algún sistema basado en realidad virtual?
3. ¿Te parecen suficientes los esfuerzos que realizan los profesores para la incorporación de nuevas tecnologías en clase?
4. ¿Tienes dificultades para recuperar la materia impartida en clase de prácticas cuando no puedes acudir un día a clase por algún motivo?
5. ¿Crees que la incorporación de sistemas de realidad virtual podría aumentar tus capacidades académicas?
6. ¿Te gustaría poder repasar los contenidos teóricos impartidos en clase de una forma más interactiva?
7. ¿Si tuvieras que adquirir un dispositivo valorado en 150€ para seguir las clases vía Streaming y poder practicar ejercicios de una forma más interactiva, lo comprarías?
8. ¿Conoces alguna herramienta que haga uso de nuevas tecnologías educativas para reforzar los contenidos de las materias adquiridas en clase?
9. ¿Crees que los profesores deberían de grabar sus clases, con objeto de poder ser visualizadas por parte de los alumnos de forma remota y en cualquier momento?
10. Del 1 (desfavorable) al 10 (muy favorable), indica el grado de satisfacción con los métodos de aprendizaje que utilizan tus profesores en el día a día.

A continuación, se muestran los resultados finales de la encuesta:

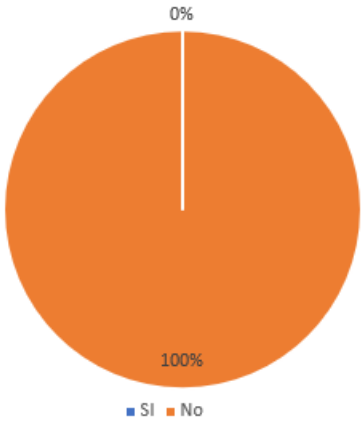
1. ¿Conoces algún profesor que utilice algún sistema basado en realidad virtual para impartir algún contenido académico en la Facultad de Ciencias?



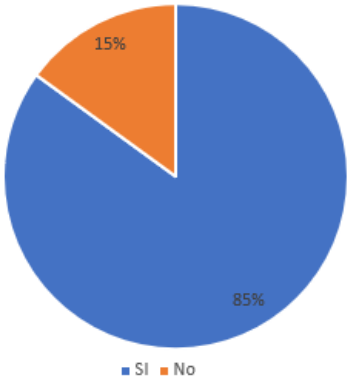
2. ¿Has probado alguna vez algún sistema basado en realidad virtual?



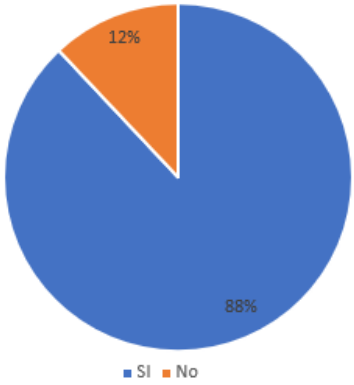
3. ¿Te parecen suficientes los esfuerzos que realizan los profesores para la incorporación de nuevas tecnologías en clase?



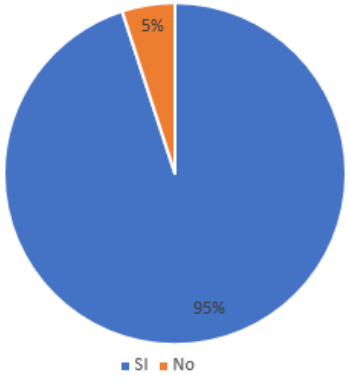
4. ¿Tienes dificultades para recuperar la materia impartida en clase de prácticas cuando no puedes acudir un día por algún motivo?



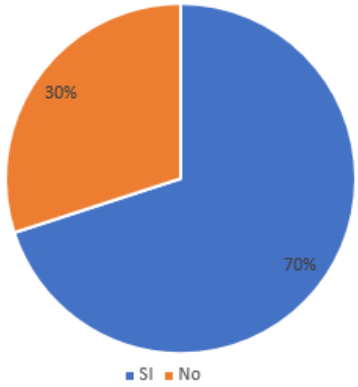
5. ¿Crees que la incorporación de sistemas de realidad virtual podría aumentar tus capacidades académicas?



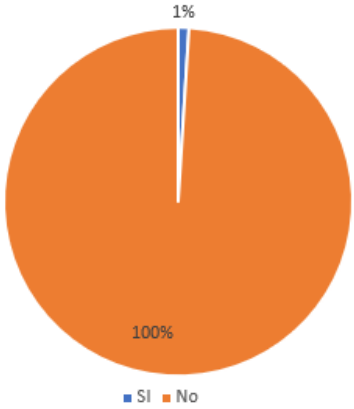
6. ¿Te gustaría poder repasar los contenidos teóricos impartidos en clase de una forma más interactiva?



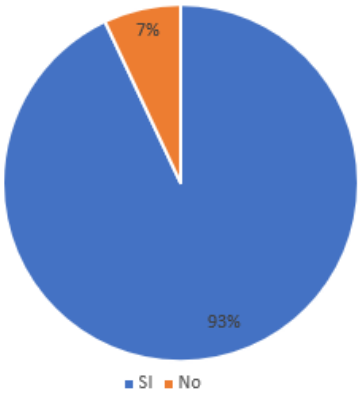
7. ¿Si tuvieras que adquirir un dispositivo valorado en 150€ para seguir las clases vía Streaming y poder practicar ejercicios de una forma más interactiva, lo harías?



8. ¿Conoces alguna herramienta que haga uso de nuevas tecnologías educativas para reforzar los contenidos de las materias adquiridas en clase?



9. ¿Crees que los profesores deberían de grabar sus clases, con objeto de poder ser visualizadas por parte de los alumnos de forma remota y en cualquier momento?



10. Del 1 (desfavorable) al 5 (muy favorable), indica el grado de satisfacción con los métodos de aprendizaje que utilizan tus profesores en el día a día.

Observando los resultados de los encuestados, se puede argumentar que los estudiantes no conocen a ningún profesor que en su centro educativo utilice algún dispositivo de realidad virtual. El (90%) de los alumnos nunca ha utilizado un dispositivo inmersivo aceptando un (70%) la posible compra de uno de ellos. Además, los alumnos hacen constar que los esfuerzos de los profesores para aplicar nuevas tecnologías en sus clases magistrales son insuficientes. Los estudiantes constatan en un 85% que presentan dificultades a la hora de recuperar la materia de clase si un día no han podido acudir y un 88% cree que la utilización de sistemas de realidad virtual ampliaría sus capacidades de aprendizaje.

Es por todo ello, que es de vital importancia la implantación de un sistema que permita a los estudiantes seguir una clase de forma remota mediante la utilización de técnicas de streaming aplicadas a dispositivos inmersivos, donde los sistemas tradicionales de educación sean apoyados por la utilización de nuevos métodos de aprendizaje. La puesta en marcha de este proyecto permitirá que los alumnos refuercen los contenidos adquiridos en sus clases de forma tradicional con un sistema inmersivo que les permita interactuar con un entorno 3D pudiendo interaccionar con otros alumnos sin necesidad de estar presente en clase.

Funcionamiento Del Escenario Virtual:

El espacio de aprendizaje inmersivo ha sido implementado atendiendo a las recomendaciones del siguiente artículo científico:

Francisco Torres, Leticia A. Neira Tovar, M. Carlos Egremy ;Virtual Interactive Laboratory Applied to High Schools Programs; Procedia Computer Science Volume 75, 2015, Pages 233-238;

https://doi.org/10.1016/j.procs.2015.12.243

Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 75 (2015) 233 – 238

Procedia Computer Science

2015 International Conference on Virtual and Augmented Reality in Education

Virtual Interactive Laboratory Applied to High Schools Programs

Francisco Torres[a]*, Leticia A. Neira Tovar[b], Carlos Egremy M. [c]

[a] Universidad Autónoma de Nuevo León, Av.Alfonso Reyes s/n, San Nicolás NL, Mex. 66450, *
[b] Universidad Autónoma de Nuevo León, Av.Alfonso Reyes s/n, San Nicolás NL, Mex 66450,
[c] Simón Bolívar 1539, Col. Mitras Centro, Monterrey, NL.Mex.64460.

Abstract

A common problem in many schools is that they do not have the necessary equipment in their science laboratories. This is often due to space and a limited budget for maintenance and equipment. Within a constructivists learning model with a multi perspective approach it is necessary for students to have an environment where they can perform scientific and technological experiments. Working in a science lab creates an interactive experience that encourages development and provides practical knowledge. Some cognitive processes arise through interaction; these play an important role in semantic construction. Usability within a platform refers to the interaction between the software and the users. This means the software will be easy to learn and navigate, allowing the user explore its potential through an attractive interface and operations. The software can be use efficiently, minimizing errors and increase user satisfaction, among others. This article focuses on studying the design and construction of a virtual laboratory through human interaction, to solve a problem through a virtual science lab. The student reads from his book *Practice of Science*, which outlines a task to perform. During the activity time, the student can interact with laboratory instruments. Even when following instructions, the student may overlook some details, which could lead to accidents or laboratory instruments damage. The result of the research includes an analysis of the construction of a virtual lab and the experience of users throughout their educational experience.

Keywords: Constructivist learning; Augmented Reality (AR); Usability;

Peer-review under responsibility of organizing committee of the 2015 International Conference on Virtual and Augmented Reality in Education (VARE 2015)

* Corresponding author. *Tel.: +5281 83294000 Ext.5889 y 1622*
E-mail address: franciscot@gmail.com

Peer-review under responsibility of organizing committee of the 2015 International Conference on Virtual and Augmented Reality in Education (VARE 2015)
doi:10.1016/j.procs.2015.12.243

1. Introduction

It is increasingly common for teachers to integrate the use of information technology in teaching courses with the goal of finding new ways to improve the classical teaching techniques, with the need to find tools that are flexible and able to withstand different learning scenarios.

A common problem for many schools is to have the necessary equipment in their science labs because the spaces, the budget for maintenance and equipment is often limited. Within a constructivist-learning model with a multi perspective approach, it is necessary for students to have a scenario where they can perform activities of science and technology. Working in a science lab includes the creation of interactive content that stimulates development and provide practical knowledge. During interaction, cognitive processes that have an important role in the processes of building semantic emerge.

The user models used in systems and devices in order to improve the human-information interaction can serve as a basis for developing strategies to find and use information in order to improve solutions to everyday problems that are significant. Usability into a platform refers to the interaction between software and users is related to attributes and characteristics, such as: ease of learning to use the software and explore its potential, the efficiency of use of the software minimizing errors, ease of navigation through the software screens and operations

The method of heuristic evaluation was developed by Nielsen [1], it is to have a standard or guidelines of usability for the system analysis, the first version was proposed by Nielsen and Molich [2], then it emerged a version [3] but Nielsen [1] published an improved version technical usability. The guidelines defined for the heuristic evaluation is 10 and is recommended to be five reviewers that apply the procedure [4, 5, 6, 7, 8].

This article focuses on studying the design and construction of a virtual laboratory through human interaction information to solve a problem from a virtual science lab. The student reads the practice from science book, which describes a task to perform. During the time of the activity, the student can interact with laboratory instruments.
Even with the instructions, the student may neglect some details on how to solve the activity may cause an accident or damage laboratory instruments.
The result of the research includes the analysis of building a virtual laboratory and user experience through their educational experience.

2. Operation

There are several investigations related to learning using devices that allow interaction in 3D, but most focus on the development of tools or content, on the other hand, research concerning usability laboratories for effective design online courses is little or almost nil.

This research is the original contribution usability to investigate the student faces to achieve learning goals during the interaction in the virtual lab. In developing educational applications, it is important to assess the quality of the interaction that takes the user to the interface because this depends on the objectives for which the application was design. To assess cognitive achievement that has to interact with the device is necessary to consider the following aspects. Lewis [7]:

- Theoretical model in which the design is based
- What are the skills related to the field of health are trying to teach
- What would be the amount of time that would have opportunity to interact
- To what extent virtual scenarios to simulate real situations
- To what extent shows a knowledge gap and conduct before and after intervention

To develop an application that can ensure the above objectives the following plan of action will be taken:

The application was develop using C# language with Unity as engine this allows us to add interactive content such as audio, video, text, 3d models, images and augmented reality tools.

The content and format of application will be evaluated by a group of professors, with the objective to present an accurate, current and complete information. Each lesson will be exposed with several teaching resources, this allows easy interaction. (Fig. 1).

To measure the quality of human computer interaction, heuristic evaluation (1990) was used, evaluating the following design factors: Visibility of system status, adequacy of the system and the real world, freedom and control by the user, consistency and standards, error prevention, recognition rather than recall, flexibility and efficiency in the use, static and minimalist design helps users and documentation.

With the use of video games in education we create components simulating the real world in order that the user learns the rules at the time is interacting with the software recognizing the consequences and facing risk in real world. Video Games in education allow to practice and to simulate scenarios several times and allowing progress only if certain objectives are achieved in order to ensure the lesson was learned correctly. In order to evaluate if there is and advance in the lesson, different goals are setting to pass the level and the success will be measured by specifics criteria's

The virtual class start with the screen of the Index of each course, then, the student can go directly to the class or select free course option.

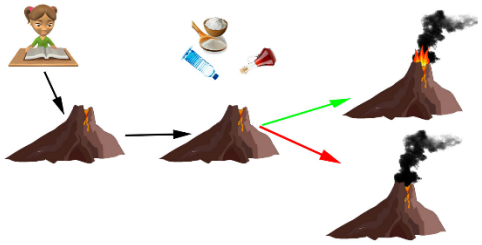


Fig. 1 Class Instructions

3. Methodology

In this research for the secondary level chemistry class an interactive laboratory was developed using Unity 3D reproducing real equipment used in school labs, everything its created in 3D for a real situation. For usability evaluation questionnaire that explores the 10 points raised by Nielsen mentioned in the previous section, saturation responses and factor analysis was developed.
Class starts when the student enters to the classroom and the teacher explain the process in order they can understand the vocabulary used increasing the understandable of each participant. For example turn of the PC, Open the program, connect the Oculus and the Kinect, etc.
Once the student is in the room dedicated for the class, with the equipment in place the class can start, Fig. 2. The teacher gives introduction of the scheduled issue proceeds to give a theoretical explanation, then begins the practice in the virtual laboratory the group will form groups in order to participate in the class with feedbacks and support.
Students will be able to move thru all the Virtual Laboratory to have a free choice to select the topic or to go directly for a specific subject, also will be able to interact with the real elements found in a Chemistry or Biology lab, once the topic is selected, the teacher is going to ask for specific actions in the class. For example in chemistry class the student need to mix chemicals to have an specific result and if the process is not completed correctly, teacher will be able to correct the student, none of this will possible with out a physical lab. Fig 2.

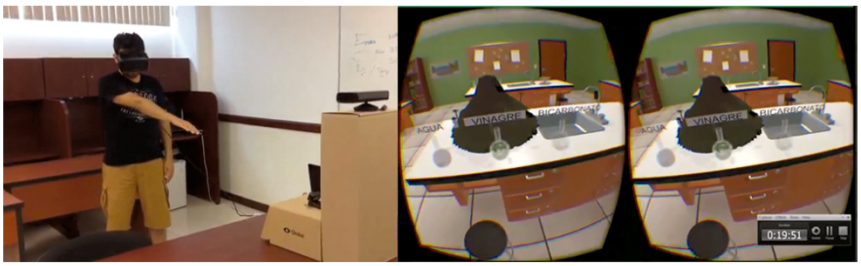


Fig. 2. Student before the practice & Oculus view

While the student is interacting with others students the rest of the class will be able to watch what is happening in the session. The advantage and convenience with this method is that with a low budget students can have the same thing that a physical lab only can have. Fig 3.

Fig. 3 Lab distribution

4. Equipment requirements

- The classroom should have a minimum space of 25 square meters and an optimum of 50 square meters.
- Empty space of 3 x 3 square meters.
- It must have an HD projector.
- Augmented reality glasses oculus rift.
- Kinect 2.0 device.
- Computers with enough hardware to run 3d renders.

Fig. 4 Equipment

5. Evaluation and metrics:

For this research two measurements were made with respect to the user experience in virtual laboratories, the first involved the construction of a method based on heuristic evaluation by Nilsen instrument and the second a questionnaire that explores the 10 points raised by Nielsen mentioned in the previous section through saturation responses and factor analysis. The instrument was performed three times, which allowed all its items were discriminatory. Responses were Likert scale show where you could qualify as follows:

[5 =] Not usability problem.
[4 =] minor problem: there needs to be fixed unless you have time to spare.
[3 =] minor problem: fix is not very important.
[2 =] serious problem: it is important to fix it.
[1 =] Disaster: It is mandatory to fix

In the different applications it explained what were the 10 categories and the vocabulary used in the instrument in order to be comprehensible to participants adapted.

6. Results and discussion

The experiment had a total of 20 participants who were divided into 11 men and 9 women. In the analysis of each ~~of~~ the dimensions of the platform it was evaluated using descriptive statistics and Cronbach's alpha used to validate the internal consistency

Table1. Cronbach's alpha values of the dimensions of the instrument

Dimension	Mean Men	Mean Women	Alpha
Visibility	3.7	4.1	.623
Similitude	2.14	3	.741
User Control	4.5	3.2	.775
Error prevention	3.8	4.3	.748
Preference	4.5	4.2	.796
Flexibility	3.8	3.7	.684
Aesthetic and minimalist design	4.2	4.7	.850
User help	1.3	1.8	.569
Documentation	2.4	2.3	.631

Being above all scales 0 5 it can be deduced that the total size of the instrument has an acceptable degree of reliability. Therefore this allows a deeper exploratory study. We can see that across the board, the results between men and women, are similar. Consequently data not fluctuate much in each dimension, which means that the views were very similar.

7. Conclusions

The use of Kinect and Oculus in the project allows users (students) to control and interact with the virtual laboratory without physical contact, through a natural user interface that recognizes gestures, which were processed through a C # code giving the software position and movements within the laboratory to be represented virtually.

One of the challenges experimented was to achieve Usability, that is a fundamental requirement for a tool of information technology to helps the user to achieve their goals in a successful manner [9,10]. In order to implement the use of an educational platform in a school must carefully evaluate their use, as it could become a workload, rather than a working tool.

With the proposal of this instrument, we assess the interface without the use of experts, but still leaves the door open for further research on this line with the same methodology to different types of population and larger samples with different features, because this research only focused on a population of young students using the interface.

Thanks to the results observed in student behaviour, could be consider the future of this research, to be extend to share experiences in remote group using the oculus cinema technology.

Acknowledgements

Authors want to thank to ENNUI STUDIO, who develop the prototype to probe the Virtual laboratory, using their experience and their equipment to develop the software.

Also thanks to Diego Flores, to support with the Unity images used in this work.

References

1. Nielsen J. *Usability Engineering*. San Francisco: Morgan Kaufmann; 1994.
2. Nielsen J, Molich R. Heuristic evaluation of user interfaces. In *Proceedings of the SIGCHI conference on Human factors in computing systems,* 1990; pp. 249-256. ACM.
3. Nielsen J. Finding usability problems through heuristic evaluation. In *Proceedings of the SIGCHI conference on Human factors in computing systems,* 1992; pp. 373-380. ACM.
4. Granić A, Glavinić V, Stankov S. Usability evaluation methodology for web-based educational systems. In *Proceedings of the 8th ERCIM Workshop on User Interfaces for All* 1990; pp. 28-29.
5. Barrera S, Takahasi H, Nakajima M. Hands-free navigation methods for moving through a virtual landscape walking interface virtual reality input devices. In *Proceedings of IEEE International Computer Graphics, 2004;* pp. 388-394.
6. Lewis M. Analysis of the Roles of "Serious Games" in Helping Teach Health-Related Knowledge and Skills and in Changing Behavior. *Journal of Diabetes Science and Technology* 2007; **1**(6): 918-920
7. Grudin J. Three faces of human-computer interaction. *IEEE Annals of the History of Computing* 2005; **4**: 46-62.
8. Coll C. *Psicología de la educación virtual: aprender y enseñar con las tecnologías de la información y la comunicación*, Ediciones Morata; 2008.
9. Kirner, TG, Saraiva AV. Software Usability Evaluation: an Empirical Study. *In Proceedings of the 9th International Conference on Enterprise Information Systems* 2007; **3**, pp. 459-465.
10. Sommerville I. *Software Engineering*. MA, USA: Addison-Wesley, Reading; 2004.

Desarrollo de La Propuesta:

La propuesta ha sido diseñada empleando tecnologías gratuitas con objeto de que el resultado de este trabajo de investigación pueda ser desplegado en los centros interesados con un coste prácticamente nulo. El escenario desarrollado en esta propuesta de innovación docente ha sido implementado con el motor gráfico Unity 2019. Entre las ventajas que podemos destacar de este motor gráfico son las siguientes:

- **Optimización** de tiempo y **multiplataforma**: con Unity3D-2019 el proyecto puede ser exportado a más de 20 plataformas en un solo clic (dispositivos móviles, consolas, ordenadores, televisores, web, realidad aumentada, realidad virtual, etc.)
- Repositorios con **diversidad de contenido**: Los assets, son elementos que componen el videojuego, tales como animaciones, modelos, sonidos, etc. Unity3D dispone de un repositorio en el que se puede encontrar prácticamente cualquier modelo 3D.
- **Potencia** en todos los entornos: Unity3D se caracteriza por su potencia tanto en entornos 2D como en entornos 3D.
- **Sencillez** en la interfaz **y fácil manejo**: Unity dispone de una interfaz muy sencilla que ayuda a los programadores a probar sus creaciones directamente en el motor, pudiendo ver el resultado final en el juego y modificarlo en el propio motor.

A continuación, se muestra un diagrama de la arquitectura planteada:

Ilustración 4. Ejemplo Arquitectura Planteada

Con objeto de evaluar la propuesta, se ha creado un espacio inmersivo de 200m^2, que replique lo más ajustado el espacio de una clase. Particularmente y en nuestro caso se ha replicado el aula Magna2 de la Facultad de Ciencias. El objetivo que se persigue es hacer sentir al usuario que se encuentra recibiendo clase, sumergirle en una realidad que no existe, transportarle a una realidad construida, una realidad virtual en la que el alumno puede interaccionar con elementos y con el profesor de una forma remota, sin necesidad de acudir presencialmente a clase. Los alumnos podrán de esta forma complementar la información que transmite el profesor, en tiempo real, mediante una serie de elementos diseñados exclusivamente para ofrecer un aprendizaje más inmersivo y de una forma más gráfica. Desde el punto de vista del profesor, se ha optado por elaborar una serie de ejercicios que apoyen los principales fundamentos teóricos con ejemplos ilustrativos con los que los alumnos podrán interactuar. Esta metodología es de gran ayuda para los alumnos, ya que, mediante el uso de la memoria visual, podrán recordar y aplicar de una mejor forma los principios teóricos aprendidos durante las clases presenciales del día a día.

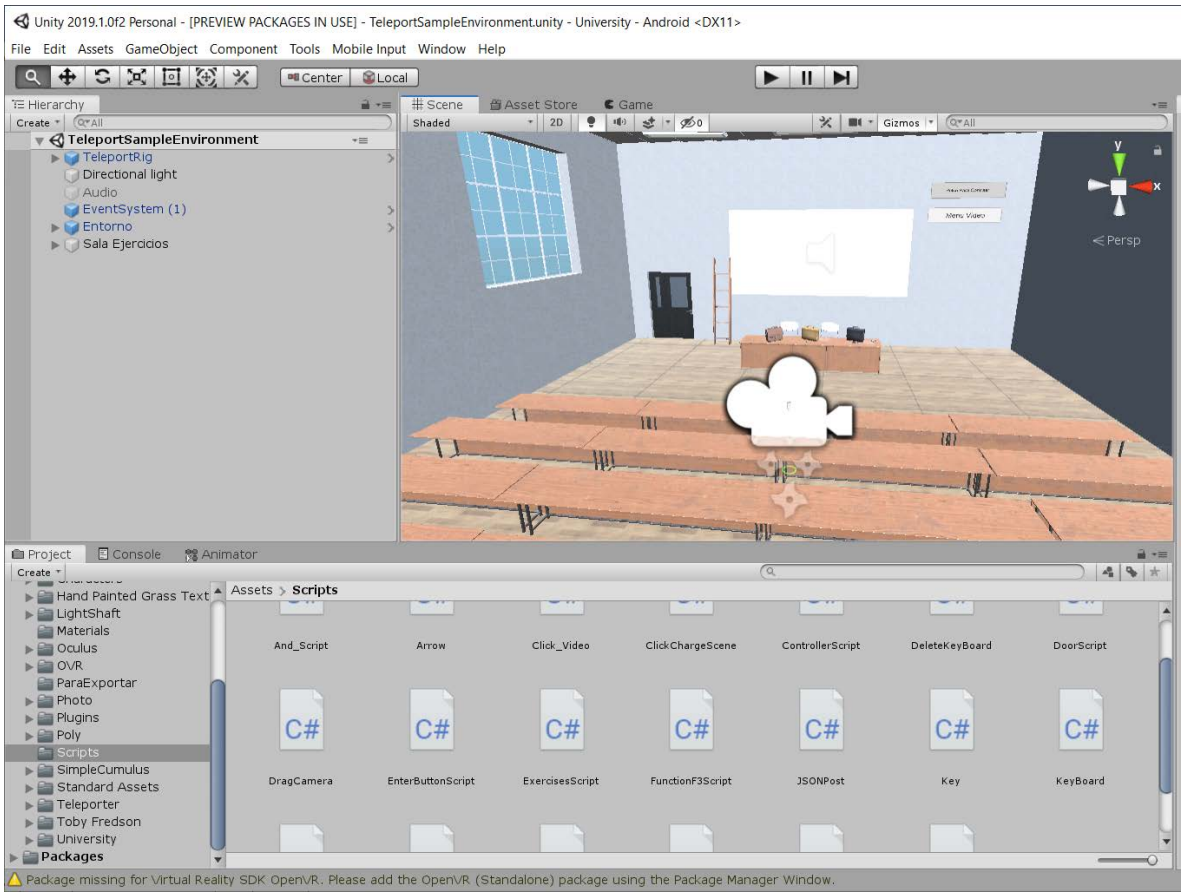


Ilustración 5. Escenario Virtual Creado

El entorno construido dispone de una sala común donde poder interactuar con otros alumnos, así como una pizarra virtual donde un proyector virtual puede reproducir cualquier contenido multimedia, desde un vídeo previamente establecido por el profesor o una clase en directo mediante la utilización de una cámara y un servicio de Streaming.

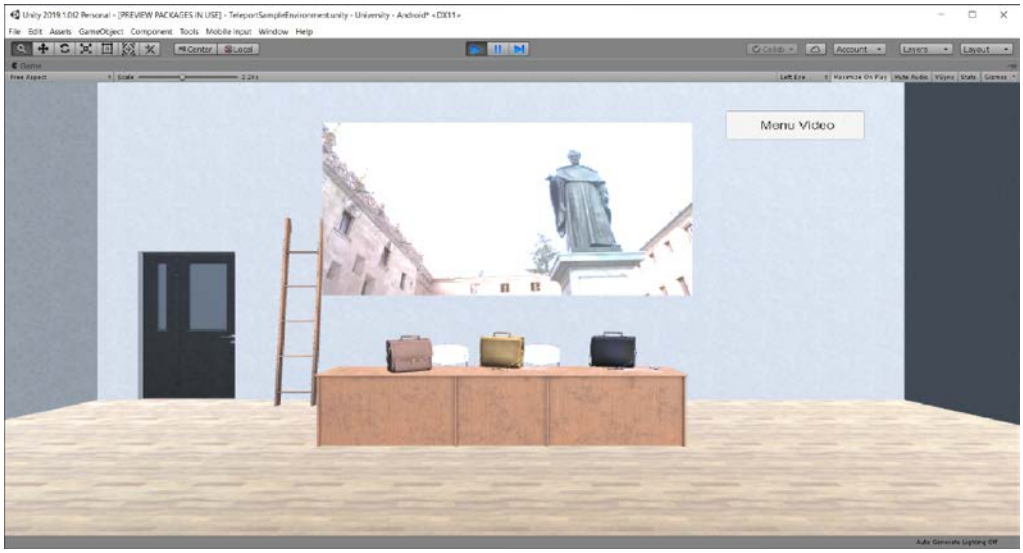


Ilustración 6. Pizarra Virtual

En la mesa se encuentran representadas las diferentes lecciones que un usuario puede practicar de forma inmersiva y están simbolizadas en forma de maletines. El alumno para iniciar una clase virtual, únicamente debe seleccionar el maletín de la lección que esté interesado y automáticamente podrá visualizar en la pizarra virtual cualquier video que el profesor haya editado, o por ejemplo, una clase en directo a través de Youtube.

Ilustración 7. Ejemplo Reproducción De Contenido en Pizarra Virtual

Además del aula principal, el alumno puede desplazarse en todo momento a una sala de ejercicios donde puede repasar los contenidos adquiridos. En esta sala el usuario puede moverse e interaccionar con sus manos para la realización de las diferentes pruebas que los profesores han preestablecido.

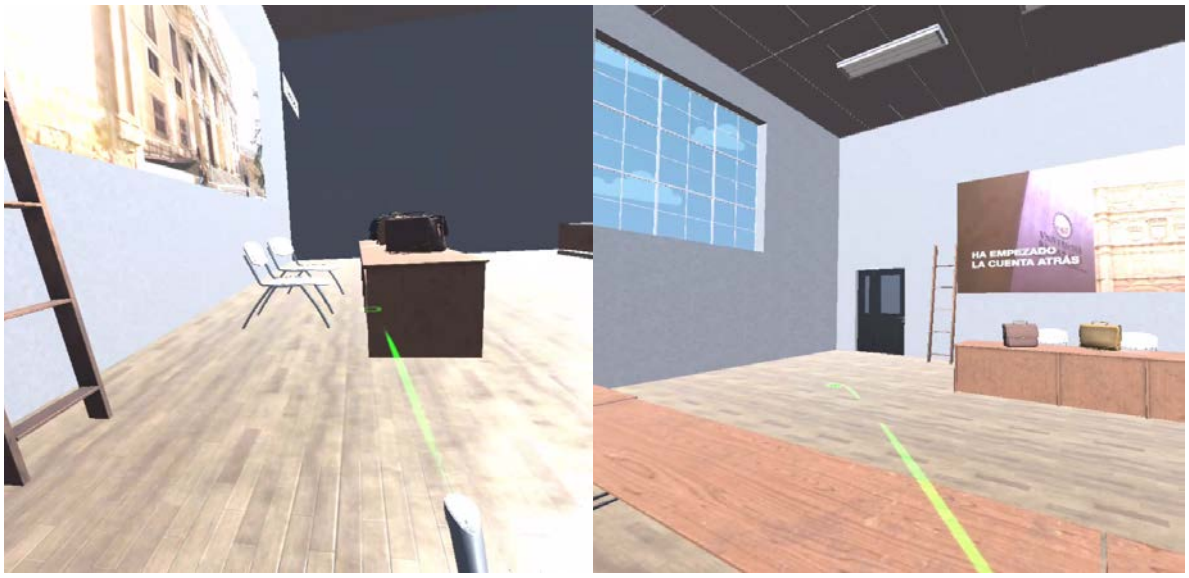


Ilustración 8. Ejemplo Movimientos Dentro Aula

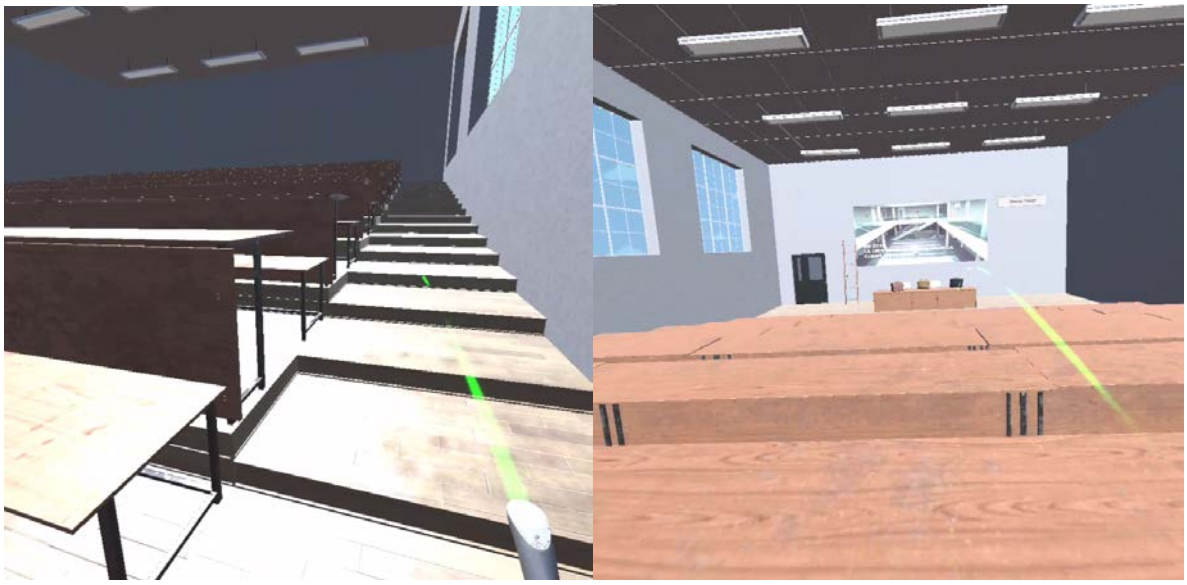


Ilustración 9. Ejemplo Movimientos Dentro Aula

La sala de ejercicios dispone de prácticas interactivas donde el usuario puede reforzar los conocimientos adquiridos en clase. Más concretamente se han diseñado ejercicios de refuerzo correspondiente a la parte práctica de la asignatura de Computadores I que es común a cualquier carrera de la rama de ingeniería. A continuación, se muestra a un alumno verificando el comportamiento de las tablas AND y OR.

Ilustración 10. Ejemplo Interacción fallo tablas and or

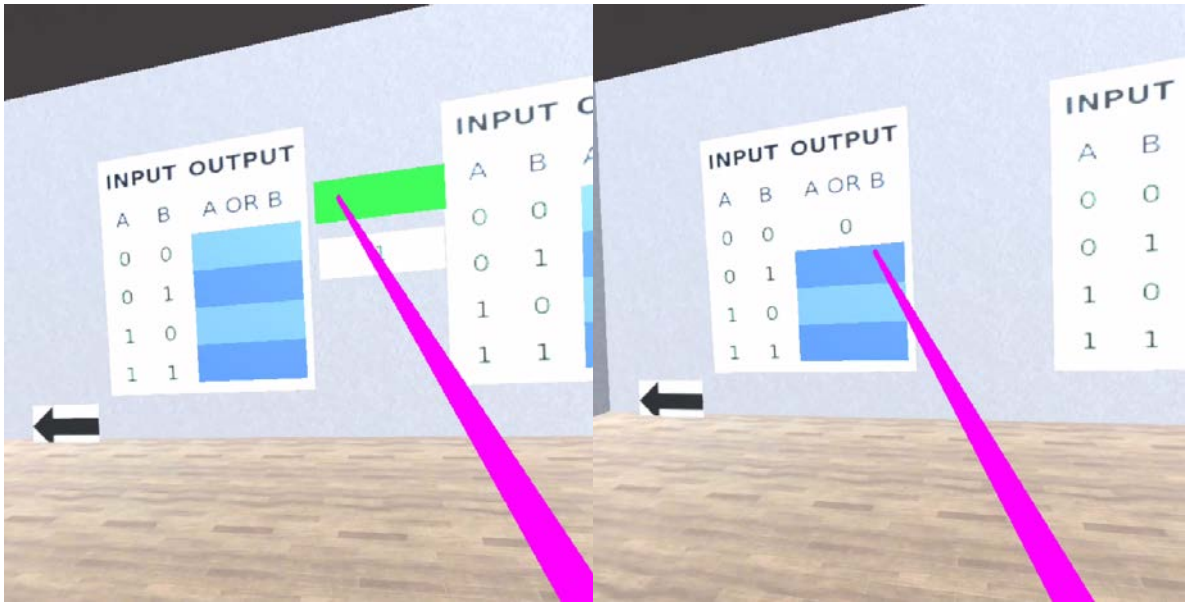


Ilustración 11. Ejemplo Interacción acierto tablas and or

Los alumnos pueden elegir la temática a repasar, en nuestra propuesta de Innovación Docente se han elaborado los diez ejercicios que son explicados en las 10 sesiones planificadas de la asignatura para el curso 2019/2020. A continuación, se muestra un ejercicio algo más complicado donde el alumno debe determinar la salida de un circuito.

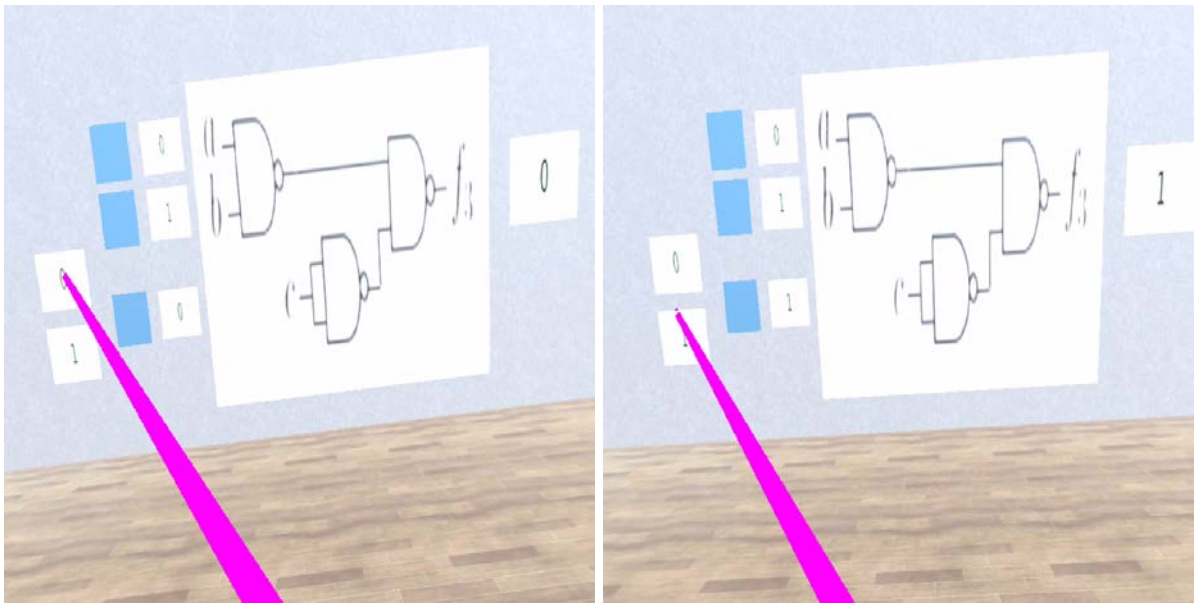


Ilustración 12. Ejemplo Determinación Salida de circuito

Con objeto de que el alumno pueda dejar constancia del trabajo realizado, el sistema permite que el alumno registre su nombre con objeto de fomentar la competitividad y que el profesor pueda recomendar material de apoyo en función de los ejercicios no resueltos de forma apropiada.

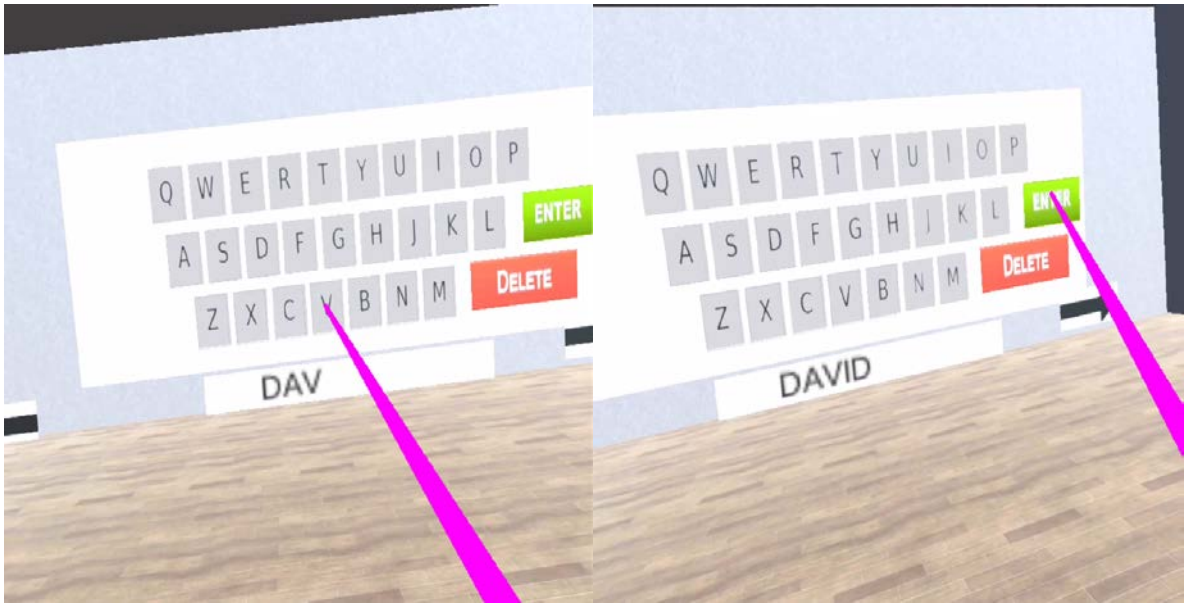


Ilustración 13. Registro de Usuario

El sistema también trackea el tiempo que los usuarios invierten en resolver cada ejercicio de forma adecuada, pudiendo el profesor visualizar que contenidos deben ser reforzados.

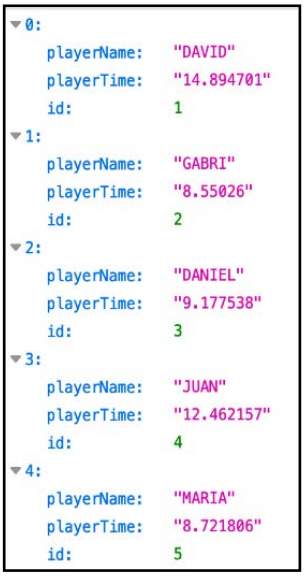


Ilustración 14. Tiempo Invertido Ejercicio 1

Conclusiones:

Una vez finalizado este proyecto, se ha diseñado una clase en 3D que puede ser utilizada por los alumnos de forma remota. La clase permite seguir clases en directo de los profesores que así lo deseen mediante la utilización de una cámara 360 y la utilización de una pizarra virtual. Además, los usuarios disponen de una zona de ejercicios interactivos donde pueden reforzar los contenidos adquiridos durante las clases magistrales. El sistema también dispone de un panel de control que permite al profesor identificar cuáles son los ejercicios que los alumnos tardan más en resolver. Los alumnos tal y como refleja la encuesta realizada durante este proyecto se encuentran mucho más motivados si se emplean nuevas herramientas educativas en los procesos de aprendizaje. La posibilidad de interactuar y hacer los ejercicios de una forma más novedosa hace que aprendan sin darse cuenta lo que rebaja la dificultad de aprendizaje. El desarrollo de este proyecto se ha realizado con tecnología open-source de forma que su implantación por parte de la Universidad permita ahorrarse costosas licencias en software. Los profesores pueden utilizar este sistema de una forma ágil, ya que el diseño y desarrollo de los diferentes ejercicios es sencillo. La posibilidad de retransmitir las clases en directo dentro del aula virtual se hace con una cuenta de Youtube académica, por lo que su puesta a punto es sencilla. Todo el código utilizado en el desarrollo de este proyecto ha sido publicado en un repositorio para que la comunidad académica pueda hacer uso de esta solución.

CODIGO ADJUNTADO

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class And_Script : MonoBehaviour {

    private bool selected, first;
    public GameObject optionZero, optionOne;
    //public GameObject[] otherOptions;
    public And_Script[] others;
    // Use this for initialization
    void Start () {
        selected = false;
        first = true;
    }

    // Update is called once per frame
    void Update () {
        if (!selected)
        {
            optionZero.SetActive(false);
            optionOne.SetActive(false);

        }
        if (selected)
        {
            optionZero.SetActive(true);
            optionOne.SetActive(true);
        }
    }


    void OnVRTriggerDownMenu()
    {
        if (!selected)
        {
            selected = true;
            PutOff();
        }
        else
        {
            selected = false;
        }
    }

    public void PutOff(){

        for (int i = 0; i < others.Length; i++)
        {
            others[i].ExternOff();
        }
    }
    public void ExternOff(){
        selected = false;
    }
}
```

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Arrow : MonoBehaviour {

    public ExercisesScript moveTo;
    public bool right;
    // Use this for initialization
    void Start () {

        }

        // Update is called once per frame
        void Update () {

        }
    void OnVRTriggerDownMenu()
    {
        if(right){
            moveTo.goRight();
        }
        else{
            moveTo.goLeft();
        }

    }

    private void OnMouseDown()
    {
        if (right)
        {
            moveTo.goRight();
        }
        else
        {
            moveTo.goLeft();
        }

    }
}
```

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Click_Video : MonoBehaviour {

    private bool selected, infoSelected;
    public GameObject videoPlay, videoPause, videoStop;
    public GameObject infoExpand, infoShrink;

    // Use this for initialization
    void Start()
    {
        selected = false;
        infoSelected = false;
    }

    // Update is called once per frame
    void Update()
    {
        if (!selected)
        {
            videoPlay.SetActive(false);
            videoPause.SetActive(false);
            videoStop.SetActive(false);


        }
        if (selected)
        {
            videoPlay.SetActive(true);
            videoPause.SetActive(true);
            videoStop.SetActive(true);

        }
        if (!infoSelected)
        {
            infoExpand.SetActive(false);
            infoShrink.SetActive(false);
        }
        if (infoSelected)
        {
            if (!videoPlay.activeSelf) {
                infoExpand.SetActive(true);
                infoShrink.SetActive(false);
            }
            else {
                infoShrink.SetActive(true);
                infoExpand.SetActive(false);
            }

        }

    }

    void OnVRTriggerDownMenu()
    {
        if (!selected)
        {
            selected = true;
```

```
        }
        else
        {
            selected = false;
        }
    }
    void OnVRTriggerUpMenu()
    {
        selected = false;
    }
    void OnVREnter()
    {
        infoSelected = true;
    }
    void OnVRExit()
    {
        infoSelected = false;
    }
}
```

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;
using UnityEngine.Video;

public class ClickChargeScene : MonoBehaviour {

    private bool selected;
    public GameObject sphere;
    public YoutubePlayer youtubePlayer;
    public string youtubeUrl;

    // Use this for initialization
    void Start()
    {
      // selected = false;
    }

    // Update is called once per frame
    void Update()
    {

    }

    void ChargeScene()
    {
        youtubePlayer.youtubeUrl = this.youtubeUrl;
        youtubePlayer.Start();
    }
}
```

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class ControllerScript : MonoBehaviour {


    public GameObject go;
    public GameObject empty;
    public GameObject panelInfo1,panelInfo2,panelInfo3;

        // Use this for initialization
        void Start () {

        empty = new GameObject();
        go = empty;
        }

        // Update is called once per frame
        void Update () {

        RaycastHit hit;
        transform.rotation = OVRInput.GetLocalControllerRotation(OVRInput.Controller.RTrackedRemote);
        //transform.rotation *= Quaternion.Euler(1,-1,1);


        if (Physics.Raycast(transform.position, transform.forward, out hit))
        {
            if(hit.collider != null)
            {
                if(go != hit.collider.gameObject)
                {
                    go.transform.SendMessage("OnVRExit");
                    go = hit.transform.gameObject;
                    go.transform.SendMessage("OnVREnter");
                    Debug.Log("On VR Raycast Enter");

                }
                if (OVRInput.GetDown(OVRInput.Button.PrimaryTouchpad))
                {

                    go.transform.SendMessage("OnVRTriggerDownMenu");
                    go.transform.SendMessage("OnVRTriggerDown");
                    go.transform.SendMessage("ChargeScene");

                }
                if (OVRInput.GetDown(OVRInput.Button.One))
                {

                }
                if (OVRInput.GetDown(OVRInput.Button.Back))
                {

                }
            }

        }
        else
        {
            if(go != null)
```

```
            {
                go.transform.SendMessage("OnVRExit");
                go = empty;
            }
        }

        if (OVRInput.GetDown(OVRInput.Button.Back))
        {
            if (panelInfo1.activeSelf) { panelInfo1.SetActive(false); }
            if (panelInfo2.activeSelf) { panelInfo2.SetActive(false); }
            if (panelInfo3.activeSelf) { panelInfo3.SetActive(false); }
        }

    }
}
```

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class DeleteKeyBoard : MonoBehaviour {

    public KeyBoard teclado;

    // Use this for initialization
    void Start () {

        }

        // Update is called once per frame
        void Update () {

        }
    void OnVRTriggerDownMenu()
    {
        teclado.borrar();
    }
}
```

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class DoorScript : MonoBehaviour {

    private bool opened;
    public Vector3 openPosition, closedPosition;

        // Use this for initialization
        void Start () {

        opened = false;
        closedPosition = transform.position;
        }

    // Update is called once per frame
    void Update()
    {
        if (!opened)
        {
            transform.position = transform.position = Vector3.Lerp(transform.position, closedPosition,
Time.deltaTime * 5f);
        }
        if (opened)
        {
            transform.position = Vector3.Lerp(transform.position, openPosition, Time.deltaTime * 5f);
        }
    }

    void OnVRTriggerDown()
    {
        opened = true;

    }
}
```

```
using System;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

//Usage: just drop into your Camera in the editor
//you can drag the view with the right mouse btn (left btn is to trigger actions)
//won't work when the HMD is connected as the HMD will take over the camera's transferm
namespace ZenvaVR
{
    public class DragCamera : MonoBehaviour
    {
#if UNITY_EDITOR

        // flag to keep track whether we are dragging or not
        bool isDragging = false;

        // starting point of a camera movement
        float startMouseX;
        float startMouseY;

        // Camera component
        Camera cam;

        // Use this for initialization
        void Start()
        {
            // Get our camera component
            cam = GetComponent<Camera>();
        }

        // Update is called once per frame
        void Update()
        {

            // if we press the left button and we haven't started dragging
            if (Input.GetMouseButtonDown(1) && !isDragging)
            {
                // set the flag to true
                isDragging = true;

                // save the mouse starting position
                startMouseX = Input.mousePosition.x;
                startMouseY = Input.mousePosition.y;
            }
            // if we are not pressing the left btn, and we were dragging
            else if (Input.GetMouseButtonUp(1) && isDragging)
            {
                // set the flag to false
                isDragging = false;
            }
        }

        void LateUpdate()
        {
            // Check if we are dragging
            if (isDragging)
            {
                //Calculate current mouse position
                float endMouseX = Input.mousePosition.x;
```

```
            float endMouseY = Input.mousePosition.y;

            //Difference (in screen coordinates)
            float diffX = endMouseX - startMouseX;
            float diffY = endMouseY - startMouseY;

            //New center of the screen
            float newCenterX = Screen.width / 2 + diffX;
            float newCenterY = Screen.height / 2 + diffY;

            //Get the world coordinate , this is where we want to look at
            Vector3 LookHerePoint = cam.ScreenToWorldPoint(new Vector3(newCenterX, newCenterY,
cam.nearClipPlane));

            //Make our camera look at the "LookHerePoint"
            transform.LookAt(LookHerePoint);

            //starting position for the next call
            startMouseX = endMouseX;
            startMouseY = endMouseY;
        }
    }

#endif
   }
}
```

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;

public class EnterButtonScript : MonoBehaviour {

    public Text texto;
    public ExercisesScript controller;
    float tiempo;

    // Use this for initialization
    void Start () {
        tiempo = 0.0f;
        }

        // Update is called once per frame
        void Update () {
        tiempo += Time.deltaTime;
        }
    void OnVRTriggerDownMenu()
    {
        POST();
        texto.text = "";
        tiempo = 0.0f;
        controller.reiniciar();
    }
    void OnMouseDown()
    {
        POST();
        texto.text = "";
        tiempo = 0.0f;
        controller.reiniciar();
    }
    WWW POST()
    {
        WWW www;
        Hashtable postHeader = new Hashtable();
        postHeader.Add("Content-Type", "application/json");
        Player playerInstance = new Player();
        playerInstance.playerName = texto.text;
        playerInstance.playerTime = tiempo.ToString();

        //Convert to Jason
        string playerToJason = JsonUtility.ToJson(playerInstance);
        // convert json string to byte
        var formData = System.Text.Encoding.UTF8.GetBytes(playerToJason);

        //www = new WWW("http://localhost:3000/score", formData, postHeader);
        www = new WWW("http://usaluniversityscore.ddns.net:3000/score", formData, postHeader);


        StartCoroutine(WaitForRequest(www));
        return www;
    }
    IEnumerator WaitForRequest(WWW data)
    {
        yield return data; // Wait until the download is done
        if (data.error != null)
        {
```

```
            Debug.Log("There was an error sending request: " + data.error);
        }
        else
        {
            Debug.Log("WWW Request: " + data.text);
        }
    }

    public class Player
    {

        public string playerName;
        public string playerTime;
    }
}
```

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class ExercisesScript : MonoBehaviour {

    public GameObject first, second, third, fourth, fifth, sixth, start, keyboard, leftArrow, rightArrow;

        // Use this for initialization
        void Start () {
        start.SetActive(true);
        first.SetActive(false);
        second.SetActive(false);
        third.SetActive(false);
        fourth.SetActive(false);
        fifth.SetActive(false);
        sixth.SetActive(false);
        keyboard.SetActive(false);

    }

        // Update is called once per frame
        void Update () {

        }
    public void iniciar(){
        start.SetActive(false);
        leftArrow.SetActive(true);
        rightArrow.SetActive(true);
        first.SetActive(true);
    }
    public void reiniciar(){
        keyboard.SetActive(false);
        leftArrow.SetActive(false);
        rightArrow.SetActive(false);
        start.SetActive(true);
    }

    public void goRight(){
        if(first.activeSelf){
            first.SetActive(false);
            second.SetActive(true);
        }
        else if (second.activeSelf)
        {
            second.SetActive(false);
            third.SetActive(true);
        }
        else if (third.activeSelf)
        {
            third.SetActive(false);
            fourth.SetActive(true);
        }
        else if (fourth.activeSelf)
        {
            fourth.SetActive(false);
            fifth.SetActive(true);
        }
        else if (fifth.activeSelf)
        {
```

```
            fifth.SetActive(false);
            sixth.SetActive(true);
        }
        else if (sixth.activeSelf)
        {
            sixth.SetActive(false);
            keyboard.SetActive(true);
        }
        else if (keyboard.activeSelf)
        {
            keyboard.SetActive(false);
            first.SetActive(true);
        }
    }
    public void goLeft(){
        if (first.activeSelf)
        {
            first.SetActive(false);
            keyboard.SetActive(true);
        }
        else if (second.activeSelf)
        {
            second.SetActive(false);
            first.SetActive(true);
        }
        else if (third.activeSelf)
        {
            third.SetActive(false);
            second.SetActive(true);
        }
        else if (fourth.activeSelf)
        {
            fourth.SetActive(false);
            third.SetActive(true);
        }
        else if (fifth.activeSelf)
        {
            fifth.SetActive(false);
            fourth.SetActive(true);
        }
        else if (sixth.activeSelf)
        {
            sixth.SetActive(false);
            fifth.SetActive(true);
        }
        else if (keyboard.activeSelf)
        {
            keyboard.SetActive(false);
            sixth.SetActive(true);
        }
    }
}
```

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class FunctionF3Script : MonoBehaviour {

    public GameObject aZero, aOne, bZero, bOne, cZero, cOne;
    public GameObject resultZero, resultOne;

        // Use this for initialization
        void Start () {

        }

        // Update is called once per frame
        void Update () {
        if( (aZero.activeSelf || aOne.activeSelf) &&
          (bZero.activeSelf || bOne.activeSelf) &&
           (cZero.activeSelf || cOne.activeSelf)){

            if( (aZero.activeSelf && bZero.activeSelf && cZero.activeSelf) ||
               (aZero.activeSelf && bOne.activeSelf && cZero.activeSelf) ||
               (aOne.activeSelf && bZero.activeSelf && cZero.activeSelf)){

                resultZero.SetActive(true);
                resultOne.SetActive(false);

            }
            else
            {
                resultZero.SetActive(false);
                resultOne.SetActive(true);
            }
        }
        }
}
```

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.Networking;

public class ClickButton : MonoBehaviour
{
    public WWW POST()
    {
        WWW www;
        Hashtable postHeader = new Hashtable();
        postHeader.Add("Content-Type", "application/json");
        Player playerInstance = new Player();
        playerInstance.playerId = "8484239823";
        playerInstance.playerLoc = "PPpPPPPPPPPPPowai";
        playerInstance.playerNick = "PPPPPPPRandom Nick";

        //Convert to Jason
        string playerToJason = JsonUtility.ToJson(playerInstance);
        // convert json string to byte
        var formData = System.Text.Encoding.UTF8.GetBytes(playerToJason);

        //www = new WWW("http://localhost:3000/posts", formData, postHeader);
        www = new WWW("http://vthemepark.ddns.net:3000/posts", formData, postHeader);


        StartCoroutine(WaitForRequest(www));
        return www;
    }
    IEnumerator WaitForRequest(WWW data)
    {
        yield return data; // Wait until the download is done
        if (data.error != null)
        {
            Debug.Log("There was an error sending request: " + data.error);
        }
        else
        {
            Debug.Log("WWW Request: " + data.text);
        }
    }
    void Start()
    {
        POST();
        /*
        Player playerInstance = new Player();
        playerInstance.playerId = "8484239823";
        playerInstance.playerLoc = "Powai";
        playerInstance.playerNick = "Random Nick";

        //Convert to Jason
        string playerToJason = JsonUtility.ToJson(playerInstance);
        //string playerToJason = JsonUtility.ToJson(playerInstance, true);
        Debug.Log(playerToJason);


        UnityWebRequest www = UnityWebRequest.Post("http://localhost:3000/posts/200",
playerToJason);
        www.SetRequestHeader("Content-Type", "application/json");
        */
```

```
    }



    void getWeb()
    {
        // A correct website page.
        StartCoroutine(GetRequest("http://localhost:3000/posts"));

        // A non-existing page.
        StartCoroutine(GetRequest("https://error.html"));
    }

    IEnumerator GetRequest(string uri)
    {
        using (UnityWebRequest webRequest = UnityWebRequest.Get(uri))
        {
            // Request and wait for the desired page.
            yield return webRequest.SendWebRequest();

            string[] pages = uri.Split('/');
            int page = pages.Length - 1;

            if (webRequest.isNetworkError)
            {
                Debug.Log(pages[page] + ": Error: " + webRequest.error);
            }
            else
            {
                Debug.Log(pages[page] + ":\nReceived: " + webRequest.downloadHandler.text);
            }
        }
    }




}


   public class Player
{

    public string playerId;
    public string playerLoc;
    public string playerNick;
}
```

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Key : MonoBehaviour {

    public KeyBoard teclado;
    public string tecla;
        // Use this for initialization
        void Start () {

        }

        // Update is called once per frame
        void Update () {

        }
    void OnVRTriggerDownMenu()
    {
        teclado.teclear(tecla);
    }
    void OnMouseDown()
    {
        teclado.teclear(tecla);
    }
}
```

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;

public class KeyBoard : MonoBehaviour {

    public Text texto;
        // Use this for initialization
        void Start () {
        texto.text = "";
        }

        // Update is called once per frame
        void Update () {

        }


    public void teclear(string tecla)
    {
        if(texto.text.Length < 10)
        texto.text = texto.text + tecla;
    }

    public void borrar()
    {
        if(texto.text.Length > 0)
            texto.text = texto.text.Substring(0, texto.text.Length - 1);

    }
}
```

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class ManageRaycastBezier : MonoBehaviour {

    public GameObject bezier, raycast;
    private bool bezierOn, raycastOn, switchButton = false, clickMode = false;
    public GameObject texto;

    private float firstClickTime;

    // Use this for initialization
    void Start () {


    }

    // Update is called once per frame
    void Update () {



    }
    void switchMode(){
        if(!switchButton){
            bezier.SetActive(true);
            raycast.SetActive(false);
        }
        else{
            bezier.SetActive(false);
            raycast.SetActive(true);
        }
        switchButton = !switchButton;


    }


    void switchToBezier()
    {
        raycast.SetActive(false);
        bezier.SetActive(true);

    }
    void switchToRaycast(){
        bezier.SetActive(false);
        raycast.SetActive(true);
    }
}
```

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class StartScript : MonoBehaviour {

    public ExercisesScript controller;

    void Start()
    {
    }
    void OnVRTriggerDownMenu()
    {
        controller.iniciar();
    }
    private void OnMouseDown()
    {
        controller.iniciar();
    }
}
```

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;

public class TooltipScript : MonoBehaviour {

    public Text helpText;
    public Text tooltipText;

    public string helpString;
    public string tooltipString;

        // Use this for initialization
        void Start () {
        helpText = GameObject.Find("HelpText").GetComponent<Text>();
        tooltipText = GameObject.Find("TooltipText").GetComponent<Text>();

    }

    // Update is called once per frame
    void Update () {

        }


    void OnVREnter()
    {
        helpText.text = helpString;
        tooltipText.text = tooltipString;
    }

    void OnVRExit()
    {
        helpText.text = "";
        tooltipText.text = "";
    }
}
```

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.Video;

public class Video_Controller : MonoBehaviour {

    public GameObject sphere;
    public enum estado {Play,Stop,Pause};
    public estado actualState;

        // Use this for initialization
        void Start () {

        }

        // Update is called once per frame
        void Update () {

        }
    void ChargeScene()
    {
        if(actualState == estado.Play)
        {
            VideoPlayer actualVideo = sphere.GetComponent<VideoPlayer>();
            actualVideo.Play();
        }
        if (actualState == estado.Stop)
        {
            VideoPlayer actualVideo = sphere.GetComponent<VideoPlayer>();
            actualVideo.Stop();
        }
        if(actualState == estado.Pause)
        {
            VideoPlayer actualVideo = sphere.GetComponent<VideoPlayer>();
            actualVideo.Pause();
        }
    }
}
```

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class ZeroOneNoneScript : MonoBehaviour {

    public GameObject toAppear;
    public GameObject[] toDissapear;
    private bool selected;

    void Start()
    {
        //selected = false;
    }

    // Update is called once per frame
    void Update()
    {/*
        if (selected)
        {
            toAppear.SetActive(true);
        }
        if (!selected)
        {
            toAppear.SetActive(false);
        }
        */
    }


    void OnVRTriggerDownMenu()
    {
        //selected = true;
        DissapearOthers();

    }
    void DissapearOthers(){
        toAppear.SetActive(true);
        for (int i = 0; i < toDissapear.Length; i++){
            toDissapear[i].SetActive(false);
        }
        gameObject.SetActive(false);
    }

}
```

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class PearlScript : MonoBehaviour {

    public GameObject player;
    bool near;
    bool closeEnough, far;
    float detectionRange = 2.5f;
   // int duration = 150;
    Animator anim;
    public GameObject target;
    public float speed;

    // Use this for initialization
    void Start()
    {
        closeEnough = false;
        near = false;
        far = true;
        speed = 500.0f;
    }

    // Update is called once per frame
    void Update()
    {


        Vector3 targetPosition = new
Vector3(target.transform.position.x,transform.position.y,target.transform.position.z);
        transform.LookAt(targetPosition);
        //float step = speed * Time.deltaTime;
        //Quaternion targetY =
Quaternion.Euler(transform.rotation.x,target.rotation.y,transform.rotation.z);
        //transform.rotation = Quaternion.RotateTowards(transform.rotation, target.rotation, step);
        //transform.RotateAround(transform.position, target.position, 20 * Time.deltaTime);

        //transform.rotation = Quaternion.Euler(transform.rotation.x, target.rotation.y,
transform.rotation.z);

        if (Vector3.Distance(player.transform.position, transform.position) <= detectionRange)
        {
            closeEnough = true;
        }
        else
        {
            closeEnough = false;
        }


        if (!closeEnough && !far)
        {
            //Close the door

            anim = gameObject.GetComponent<Animator>();
            anim.SetBool("IsNear", false);

            near = false;
            far = true;
```

```
        }

        if (closeEnough && !near)
        {
            //Open the door
            anim = gameObject.GetComponent<Animator>();
            anim.SetBool("IsNear", true);

            near = true;
            far = false;
        }


    }
}
```

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class ZeroOneScript : MonoBehaviour {

    private bool selected;
    public bool isCorrect;
    public GameObject rightWrong, deleteUI;
    public int numeroEscena;

    // Use this for initialization
    void Start()
    {
        selected = false;
    }

    // Update is called once per frame
    void Update()
    {
        if (selected)
        {
            rightWrong.SetActive(true);
            Invoke("GetOff", 1);
        }
        if (!selected)
        {
            rightWrong.SetActive(false);

        }
    }


    void OnVRTriggerDownMenu()
    {
        selected = true;

    }

    void GetOff()
    {
        selected = false;
        if(isCorrect){
            if (deleteUI != null)
            {
                deleteUI.SetActive(false);
            }
        }

    }

}
```

```
using UnityEngine;
using System;

/// <summary>
/// Draws the field/property ONLY if the compared property compared by the comparison type with the value of comparedValue returns true.
/// Based on: https://forum.unity.com/threads/draw-a-field-only-if-a-condition-is-met.448855/
/// </summary>
[AttributeUsage(AttributeTargets.Property | AttributeTargets.Field, AllowMultiple = true)]
public class DrawIfAttribute : PropertyAttribute
{
    #region Fields

    public string comparedPropertyName { get; private set; }
    public object comparedValue { get; private set; }
    public DisablingType disablingType { get; private set; }

    /// <summary>
    /// Types of comparisons.
    /// </summary>
    public enum DisablingType
    {
        ReadOnly = 2,
        DontDraw = 3
    }

    #endregion

    /// <summary>
    /// Only draws the field only if a condition is met. Supports enum and bools.
    /// </summary>
    /// <param name="comparedPropertyName">The name of the property that is being compared (case sensitive).</param>
    /// <param name="comparedValue">The value the property is being compared to.</param>
    /// <param name="disablingType">The type of disabling that should happen if the condition is NOT met. Defaulted to DisablingType.DontDraw.</param>
    public DrawIfAttribute(string comparedPropertyName, object comparedValue, DisablingType disablingType = DisablingType.DontDraw)
    {
        this.comparedPropertyName = comparedPropertyName;
        this.comparedValue = comparedValue;
        this.disablingType = disablingType;
    }
}
```

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using YoutubeLight;
using System;
using SimpleJSON;
using UnityEngine.Events;
using System.Threading;
using UnityEngine.Networking;

public class PlayYoutubeFromDeviceMediaPlayer : RequestResolver {

    RequestResolver resolver;
    public string videoUrl;
    public bool getFromWebServer = false;

    public bool playOnStart;

    public UnityEvent VideoFinished;

    private void Start()
    {
        resolver = gameObject.AddComponent<RequestResolver>();
        if (playOnStart)
            PlayVideo(videoUrl);
    }

    public void PlayVideo(string url)
    {
        CheckVideoUrlAndExtractThevideoId(videoUrl);
        if (!getFromWebServer)
            resolver.GetDownloadUrls(FinishLoadingUrls, url, null);
        else
            StartCoroutine(NewRequest(url));

    }

    void FinishLoadingUrls()
    {
        List<VideoInfo> videoInfos = resolver.videoInfos;
        foreach (VideoInfo info in videoInfos)
        {
            if (info.VideoType == VideoType.Mp4 && info.Resolution == (360))
            {
                if (info.RequiresDecryption)
                {
                    //The string is the video url
                    DecryptDownloadUrl(info.DownloadUrl, info.HtmlPlayerVersion, false);
                    break;
                }
                else
                {
                    StartCoroutine(Play(info.DownloadUrl));
                }
                break;
            }
        }
    }

    public void DecryptionFinished(string url)
```

```
    {
        StartCoroutine(Play(url));
    }

    IEnumerator Play(string url)
    {
        Debug.Log("Play!");
#if UNITY_IPHONE || UNITY_ANDROID
        Handheld.PlayFullScreenMovie(url, Color.black, FullScreenMovieControlMode.Full,
FullScreenMovieScalingMode.Fill);
#else
        Debug.Log("This only runs in mobile");
#endif
        yield return new WaitForSeconds(1);

        if(VideoFinished != null)
            VideoFinished.Invoke();
    }

    private const string serverURI = "https://unity-dev-youtube.herokuapp.com/api/info?
url=https://www.youtube.com/watch?v=";
    private const string formatURI = "&format=best&flatten=true";
    public YoutubeResultIds newRequestResults;

    IEnumerator NewRequest(string videoID)
    {
        WWW request = new WWW(serverURI + "" + videoID + "" + formatURI);
        yield return request;
        var requestData = JSON.Parse(request.text);
        var videos = requestData["videos"][0]["formats"];
        newRequestResults.bestFormatWithAudioIncluded = requestData["videos"][0]["url"];

        videoUrl = newRequestResults.bestFormatWithAudioIncluded;
#if UNITY_WEBGL
        //videoUrl = ConvertToWebglUrl(videoUrl);
        //audioVideoUrl = ConvertToWebglUrl(audioVideoUrl);
#endif
        StartCoroutine(Play(videoUrl));
    }

    private string CheckVideoUrlAndExtractThevideoId(string url)
    {
        if (url.Contains("?t="))
        {
            int last = url.LastIndexOf("?t=");
            string copy = url;
            string newString = copy.Remove(0, last);
            newString = newString.Replace("?t=", "");
            url = url.Remove(last);
        }

        if (!url.Contains("youtu"))
        {
            url = "youtube.com/watch?v=" + url;
        }

        bool isYoutubeUrl = TryNormalizeYoutubeUrlLocal(url, out url);
        if (!isYoutubeUrl)
        {
            Debug.LogError("ITS NOT A YOUTUBE URL");
        }
```

```
        return url;
    }

    private bool TryNormalizeYoutubeUrlLocal(string url, out string normalizedUrl)
    {
        url = url.Trim();
        url = url.Replace("youtu.be/", "youtube.com/watch?v=");
        url = url.Replace("www.youtube", "youtube");
        url = url.Replace("youtube.com/embed/", "youtube.com/watch?v=");

        if (url.Contains("/v/"))
        {
            url = "https://youtube.com" + new Uri(url).AbsolutePath.Replace("/v/", "/watch?v=");
        }

        url = url.Replace("/watch#", "/watch?");
        IDictionary<string, string> query = HTTPHelperYoutube.ParseQueryString(url);

        string v;


        if (!query.TryGetValue("v", out v))
        {
            normalizedUrl = null;
            return false;
        }

        normalizedUrl = "https://youtube.com/watch?v=" + v;

        return true;
    }

    public void DecryptDownloadUrl(string encryptedUrl, string htmlVersion, bool audioDecryption)
    {
        if (audioDecryption)
            EncryptUrlForAudio = encryptedUrl;
        else
            EncryptUrlForVideo = encryptedUrl;

        IDictionary<string, string> queries = HTTPHelperYoutube.ParseQueryString(encryptedUrl);
        if (queries.ContainsKey(SignatureQuery))
        {
            if (audioDecryption)
                encryptedSignatureAudio = queries[SignatureQuery];
            else
                encryptedSignatureVideo = queries[SignatureQuery];

            //decrypted = GetDecipheredSignature( encryptedSignature);
            //MagicHands.DecipherWithVersion(encryptedSignature, videoInfo.HtmlPlayerVersion);
            //string jsUrl = string.Format("http://s.ytimg.com/yts/jsbin/{0}-{1}.js",
videoInfo.HtmlscriptName, videoInfo.HtmlPlayerVersion);
            string jsUrl = string.Format("http://s.ytimg.com/yts/jsbin/player{0}.js", htmlVersion);
            if (audioDecryption)
                StartCoroutine(Downloader(jsUrl, true));
            else
                StartCoroutine(Downloader(jsUrl, false));

        }

    }
```

```
    private void FixedUpdate()
    {
        if (decryptedUrlForVideo)
        {
            decryptedUrlForVideo = false;
            DecryptionFinished(decryptedVideoUrlResult);
        }
    }

    private Thread thread1;
    public void ReadyForExtract(string r, bool audioExtract)
    {
        if (audioExtract)
        {
            SetMasterUrlForAudio(r);
#if UNITY_WEBGL
            DoRegexFunctionsForAudio();
#else
            if (SystemInfo.processorCount > 1)
            {
                thread1 = new Thread(DoRegexFunctionsForAudio);
                thread1.Start();
            }
            else
            {
                DoRegexFunctionsForAudio();
            }
#endif


        }
        else
        {
            SetMasterUrlForVideo(r);
#if UNITY_WEBGL
            DoRegexFunctionsForVideo();
#else
            if (SystemInfo.processorCount > 1)
            {
                thread1 = new Thread(DoRegexFunctionsForVideo);
                thread1.Start();
            }
            else
            {
                DoRegexFunctionsForVideo();
            }
#endif

        }

    }

    IEnumerator Downloader(string uri, bool audio)
    {
        UnityWebRequest request = UnityWebRequest.Get(uri);
        request.SetRequestHeader("User-Agent", "Mozilla/5.0 (X11; Linux x86_64; rv:10.0)
Gecko/20100101 Firefox/10.0 (Chrome)");
        yield return request.Send();
        ReadyForExtract(request.downloadHandler.text, audio);
    }
```

```
}
```

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.Video;

public class ReactingLights : MonoBehaviour {

	public VideoPlayer videoSource;
	public Light[] lights;
	public Color averageColor;
	private Texture2D tex;
	public enum VideoSide{
		up,
		left,
		right,
		down,
		center
	}

	public VideoSide videoSide;

	private void Start(){
		videoSource.frameReady += NewFrame;
		videoSource.sendFrameReadyEvents = true;
	}

	bool createTexture = false;
	private void NewFrame(VideoPlayer vplayer, long frame){
		if (!createTexture) {
			createTexture = true;
			switch (videoSide) {
			case VideoSide.up:
				tex = new Texture2D(videoSource.texture.width/2,20);
				break;
			case VideoSide.down:
				tex = new Texture2D(videoSource.texture.width/2,20);
				break;
			case VideoSide.left:
				tex = new Texture2D(20,videoSource.texture.height/2);
				break;
			case VideoSide.right:
				tex = new Texture2D(20,videoSource.texture.height/2);
				break;
			case VideoSide.center:
				tex = new Texture2D (videoSource.texture.height / 2, videoSource.texture.height /
2);
				break;
			}
		}
		RenderTexture.active = (RenderTexture)videoSource.texture;
		switch (videoSide) {
			case VideoSide.up:
			tex.ReadPixels(new
Rect((videoSource.texture.width/2),0,videoSource.texture.width/2,20),0,0);
				break;
			case VideoSide.down:
			tex.ReadPixels(new Rect((videoSource.texture.width/2),videoSource.texture.height-
20,videoSource.texture.width/2,20),0,0);
				break;
			case VideoSide.left:
```

```
                tex.ReadPixels(new Rect(0,0,20,videoSource.texture.height/2),0,0);
                    break;
                case VideoSide.right:
                tex.ReadPixels(new Rect(videoSource.texture.width-
20,0,20,videoSource.texture.height/2),0,0);
                    break;
                case VideoSide.center:
                    tex.ReadPixels(new Rect((videoSource.texture.width/2)-
(videoSource.texture.width/2),(videoSource.texture.height/2)-
(videoSource.texture.height/2),videoSource.texture.width/2,videoSource.texture.height/2),0,0);
                    break;
            }

            tex.Apply();
            averageColor = AverageColorFromTexture (tex);
            averageColor.a = 1;
            foreach (Light light in lights)
                light.color = averageColor;
        }

        Color32 AverageColorFromTexture(Texture2D tex)
        {

            Color32[] texColors = tex.GetPixels32();

            int total = texColors.Length;

            float r = 0;
            float g = 0;
            float b = 0;

            for(int i = 0; i < total; i++)
            {

                r += texColors[i].r;

                g += texColors[i].g;

                b += texColors[i].b;

            }

            return new Color32((byte)(r / total) , (byte)(g / total) , (byte)(b / total) , 0);

        }
}
```

```
using System.Reflection;
using UnityEngine.UI;

public static class UISetExtensions
{
	private static readonly MethodInfo toggleSetMethod;
	private static readonly MethodInfo sliderSetMethod;
	private static readonly MethodInfo scrollbarSetMethod;

	private static readonly FieldInfo dropdownValueField;
	private static readonly MethodInfo dropdownRefreshMethod;  // Unity 5.2 <= only

	static UISetExtensions()
	{
		// Find the Toggle's set method
		toggleSetMethod = FindSetMethod(typeof (Toggle));

		// Find the Slider's set method
		sliderSetMethod = FindSetMethod(typeof (Slider));

		// Find the Scrollbar's set method
		scrollbarSetMethod = FindSetMethod(typeof (Scrollbar));

		// Find the Dropdown's value field and its' Refresh method
		dropdownValueField = (typeof (Dropdown)).GetField("m_Value", BindingFlags.NonPublic |
BindingFlags.Instance);
		dropdownRefreshMethod = (typeof (Dropdown)).GetMethod("Refresh",
BindingFlags.NonPublic | BindingFlags.Instance);  // Unity 5.2 <= only
	}

	public static void Set(this Toggle instance, bool value, bool sendCallback = false)
	{
		toggleSetMethod.Invoke(instance, new object[] {value, sendCallback});
	}

	public static void Set(this Slider instance, float value, bool sendCallback = false)
	{
		sliderSetMethod.Invoke(instance, new object[] {value, sendCallback});
	}

	public static void Set(this Scrollbar instance, float value, bool sendCallback = false)
	{
		scrollbarSetMethod.Invoke(instance, new object[] {value, sendCallback});
	}

	public static void Set(this Dropdown instance, int value)
	{
		dropdownValueField.SetValue(instance, value);
		instance.RefreshShownValue();
	}

	private static MethodInfo FindSetMethod(System.Type objectType)
	{
		var methods = objectType.GetMethods(BindingFlags.NonPublic | BindingFlags.Instance);
		for (var i = 0; i < methods.Length; i++)
		{
			if (methods[i].Name == "Set" && methods[i].GetParameters().Length == 2)
			{
				return methods[i];
			}
```

```
            }
            return null;
        }
}
```

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.Video;
using YoutubeLight;
using SimpleJSON;
using System.Text;
using System;

public class WebGlPlayback : MonoBehaviour
{
    /*PRIVATE INFO DO NOT CHANGE THESE URLS OR VALUES*/
    private const string serverURI = "https://unity-dev-youtube.herokuapp.com/api/info?
url=https://www.youtube.com/watch?v=";
    private const string formatURI = "&format=best[ext=mp4]/mp4&flatten=true";
    private const string videoURI = "https://youtubewebgl.herokuapp.com/download.php?
mime=video/mp4&title=generatedvideo&token=";
    /*END OF PRIVATE INFO*/

    public YoutubeResultIds webGlResults;

    public string videoId = "bc0sJvtKrRM";
    private string videoUrl;
    private string audioVideoUrl;
    //If you will use high quality playback we need one video player that only will run the audio.
    public VideoPlayer unityVideoPlayer;
    //start playing the video
    public bool playOnStart = false;

    private bool noHD = false;

    RequestResolver resolver;

    public void Start()
    {
        resolver = gameObject.AddComponent<RequestResolver>();
        if (playOnStart)
        {
            PlayYoutubeVideo(videoId);
        }
    }


    public void PlayYoutubeVideo(string _videoId)
    {
#if UNITY_WEBGL
        //if (this.GetComponent<VideoController>() != null)
        //{
        //        this.GetComponent<VideoController>().ShowLoading("Loading...");
        //}
        //videoId = _videoId;
        //StartCoroutine(WebGlRequest(videoId));
#else
        Debug.LogError("Please use this script only for webgl");
#endif

    }

    IEnumerator WebGlRequest(string videoID)
    {
```

```
        WWW request = new WWW(serverURI+""+videoID+""+formatURI);
        yield return request;
        webGlResults = new YoutubeResultIds();
        Debug.Log(request.url);
        var requestData = JSON.Parse(request.text);
        var videos = requestData["videos"][0]["formats"];
        webGlResults.bestFormatWithAudioIncluded = requestData["videos"][0]["url"];

        for (int counter = 0; counter < videos.Count; counter++) {
            if(videos[counter]["format_id"] == "160")
            {
                webGlResults.lowQuality = videos[counter]["url"];
            }else if (videos[counter]["format_id"] == "133")
            {
                webGlResults.lowQuality = videos[counter]["url"];   //if have 240p quality overwrite the 144
quality as low quality.
            }else if(videos[counter]["format_id"] == "134")
            {
                webGlResults.standardQuality = videos[counter]["url"];  //360p
            }else if (videos[counter]["format_id"] == "136")
            {
                webGlResults.hdQuality = videos[counter]["url"];  //720p
            }else if(videos[counter]["format_id"] == "137")
            {
                webGlResults.fullHdQuality = videos[counter]["url"];  //1080p
            }else if (videos[counter]["format_id"] == "266")
            {
                webGlResults.ultraHdQuality = videos[counter]["url"];  //@2160p 4k
            }else if (videos[counter]["format_id"] == "139")
            {
                webGlResults.audioUrl = videos[counter]["url"];  //AUDIO
            }
        }
        //quality selection will be implemented later for webgl, i recomend use the
webGlResults.bestFormatWithAudioIncluded
        WebGlGetVideo(webGlResults.bestFormatWithAudioIncluded);
    }



    //WEBGL only...
    public void WebGlGetVideo(string url)
    {
        byte[] bytesToEncode = Encoding.UTF8.GetBytes(url);
        string encodedText = Convert.ToBase64String(bytesToEncode);
        videoUrl = videoURI+""+ encodedText;
        Debug.Log("Play!! " + videoUrl);
        unityVideoPlayer.source = VideoSource.Url;
        unityVideoPlayer.url = videoUrl;
        Debug.Log("Play or not?!"+ videoUrl);
        unityVideoPlayer.Prepare();
        videoPrepared = false;
        unityVideoPlayer.prepareCompleted += VideoPrepared;

    }

    IEnumerator WebGLPlay() //The prepare not respond so, i forced to play after some seconds
    {
        yield return new WaitForSeconds(2f);
        Play();
    }
```

```
    private bool audioDecryptDone = false;
    private bool videoDecryptDone = false;

    public VideoPlayer audioVplayer;

    bool startedPlaying = false;

    void FixedUpdate()
    {
        if(unityVideoPlayer.isPrepared)
        {
            if (!startedPlaying)
            {
                startedPlaying = true;
                StartCoroutine(WebGLPlay());
            }
        }

        CheckIfIsDesync();
    }

    private bool videoPrepared;
    private bool audioPrepared;

    void AudioPrepared(VideoPlayer vPlayer)
    {
        audioVplayer.prepareCompleted -= AudioPrepared;
        audioPrepared = true;
        if (audioPrepared && videoPrepared)
            Play();
    }

    void VideoPrepared(VideoPlayer vPlayer)
    {
        unityVideoPlayer.prepareCompleted -= VideoPrepared;
        videoPrepared = true;
        Debug.Log("Playing youtube video only, the audio separated will be implemented in the final
release of webgl support");
        noHD = true; //force now to prevent bugs...
        Play();
    }

    public void Play()
    {
        unityVideoPlayer.loopPointReached += PlaybackDone;
        StartCoroutine(WaitAndPlay());
    }

    private void PlaybackDone(VideoPlayer vPlayer)
    {
        OnVideoFinished();
    }


    IEnumerator WaitAndPlay()
    {

        if (!noHD)
        {
```

```
            audioVplayer.Play();
            if (syncIssue)
                yield return new WaitForSeconds(0.35f);
            else
                yield return new WaitForSeconds(0);
        }
        else
        {
            if (syncIssue)
                yield return new WaitForSeconds(1f);//if is no hd wait some more
            else
                yield return new WaitForSeconds(0);
        }
        unityVideoPlayer.Play();
        //if (this.GetComponent<VideoController>() != null)
        //{
        //    this.GetComponent<VideoController>().HideLoading();
        //}
    }

    IEnumerator StartVideo()
    {
#if UNITY_IPHONE || UNITY_ANDROID
            Handheld.PlayFullScreenMovie (videoUrl);
#endif
        yield return new WaitForSeconds(1.4f);
        OnVideoFinished();
    }

    public void OnVideoFinished()
    {
        if (unityVideoPlayer.isPrepared)
        {
            Debug.Log("Finished");
            if (unityVideoPlayer.isLooping)
            {
                unityVideoPlayer.time = 0;
                unityVideoPlayer.frame = 0;
                audioVplayer.time = 0;
                audioVplayer.frame = 0;
                unityVideoPlayer.Play();
                audioVplayer.Play();
            }
        }
    }

    public enum VideoQuality
    {
        mediumQuality,
        Hd720,
        Hd1080,
        Hd1440,
        Hd2160
    }

    [HideInInspector]
    public bool isSyncing = false;

    [Header("If you think audio is out of sync enable this bool below")]
    [Header("This happens in some unity versions, the most stable is the 5.6.1p1")]
    public bool syncIssue;
```

```
    //Experimental
    private void CheckIfIsDesync()
    {
        if (!noHD)
        {
            //Debug.Log(unityVideoPlayer.time+" "+ audioVplayer.time);
            double t = unityVideoPlayer.time - audioVplayer.time;
            if (!isSyncing)
            {
                if (t != 0)
                {
                    Sync();
                }
                else if (unityVideoPlayer.frame != audioVplayer.frame)
                {
                    Sync();
                }
            }
        }
        else
        {
            //unityVideoPlayer.frame -= 1;
        }
    }

    private void Sync()
    {
        //VideoController controller = GameObject.FindObjectOfType<VideoController>();
        //if (controller != null)
        //{
        //    //isSyncing = true;
        //    //audioVplayer.time = unityVideoPlayer.time;
        //    //audioVplayer.frame = unityVideoPlayer.frame;
        //    //controller.Seek();
        //}
        //else
        //{
        //    Debug.LogWarning("Please add a video controller to your scene to make the sync work! Will be
improved in the future.");
        //}
    }

    public int GetMaxQualitySupportedByDevice()
    {
        if (Screen.orientation == ScreenOrientation.Landscape)
        {
            //use the height
            return Screen.currentResolution.height;
        }
        else if (Screen.orientation == ScreenOrientation.Portrait)
        {
            //use the width
            return Screen.currentResolution.width;
        }
        else
        {
            return Screen.currentResolution.height;
        }
    }
}
```

```
[System.Serializable]
public class YoutubeResultIds
{
    public string lowQuality;
    public string standardQuality;
    public string mediumQuality;
    public string hdQuality;
    public string fullHdQuality;
    public string ultraHdQuality;
    public string bestFormatWithAudioIncluded;
    public string audioUrl;

}
```

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class YoutubeLogo : MonoBehaviour {

    public string youtubeurl;

    private void OnMouseDown()
    {
        Application.OpenURL(youtubeurl);
    }

}
```

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.Video;
using YoutubeLight;
using SimpleJSON;
using System.Text;
using System;
using UnityEngine.Events;
using UnityEngine.UI;
using System.Threading;
using UnityEngine.Networking;

public class YoutubePlayer : RequestResolver
{
    #region ENUMS
    public enum YoutubeVideoQuality
    {
        LOW,
        HD,
        FULLHD,
        UHD1440,
        UHD2160
    }
    #endregion

    #region PUBLIC VARIABLES
    [Space]
    [Tooltip("You can put urls that start at a specific time example: 'https://youtu.be/1G1nCxxQMnA?
t=67'")]
    public string youtubeUrl;

    [Space]
    [Tooltip("The desired video quality you want to play.")]
    public YoutubeVideoQuality videoQuality;

    [Space]
    [Tooltip("Start playing the video from a desired time")]
    public bool startFromSecond = false;
    [DrawIf("startFromSecond", true)]
    public int startFromSecondTime = 0;

    [Space]
    [Tooltip("Play the video when the script initialize")]
    public bool autoPlayOnStart = true;

    [Space]
    [Tooltip("Play or continue when OnEnable is called")]
    public bool autoPlayOnEnable = false;

    [Space]
    [Tooltip("If you want to use your custom player, you can enable this and set the callback
OnYoutubeUrlLoaded and get the public variables audioUrl or videoUrl of that script.")]
    public bool loadYoutubeUrlsOnly = false;

    [Space]
    [Tooltip("If the video is a 3D video sidebyside or Over/Under")]
    public bool is3DLayoutVideo = false;

    [DrawIf("is3DLayoutVideo", true)]
```

```
public Layout3D layout3d;

public enum Layout3D
{
    sideBySide,
    OverUnder
}

[Space]
[Header("Video Controller Canvas")]
public GameObject videoControllerCanvas;

[Space]
public Camera mainCamera;

[Space]
[Header("Loading Settings")]
[Tooltip("This enable and disable related to the loading needs.")]
public GameObject loadingContent;

[Header("Custom user Events")]
//User callbacks
[Tooltip("When the url's are loaded")]
public UnityEvent OnYoutubeUrlAreReady;
[Tooltip("When the videos are ready to play")]
public UnityEvent OnVideoReadyToStart;
[Tooltip("When the video start playing")]
public UnityEvent OnVideoStarted;
[Tooltip("When the video pause")]
public UnityEvent OnVideoPaused;
[Tooltip("When the video finish")]
public UnityEvent OnVideoFinished;

[Space]
[Header("Render the same video to more objects")]
[Tooltip("Render the same video player material to a different materials, if you want")]
public GameObject[] objectsToRenderTheVideoImage;

[Space]
[Header("The unity video players")][Tooltip("The unity video player")]
public VideoPlayer videoPlayer;

[Tooltip("The audio player, (Needed for videos that dont have audio included 720p+)")]
public VideoPlayer audioPlayer;

[Space]
[Tooltip("Show the output in the console")]
public bool debug;


//Youtube formated urls
[HideInInspector]
public string videoUrl;
[HideInInspector]
private string audioUrl;

[Space]
public bool ForceGetWebServer = false;

[Space]
[Tooltip("Show the video controller in screen [slider with progress, video time, play pause, etc...]")]
```

```
    public bool showPlayerControls = false;

    #endregion

    #region PRIVATE VARIABLES
    //Request from youtube url timeout
    private int maxRequestTime = 5;
    private float currentRequestTime;
    //When the video fails how much time we will try until try to get from the webserver system.
    private int retryTimeUntilToRequestFromServer = 1;
    private int currentRetryTime = 0;

    //Check when we are trying to get the url
    private bool gettingYoutubeURL = false;

    //When the urls are done and the video are ready to start playing
    private bool videoAreReadyToPlay = false;

    //The system that get the urls from youtube.
    RequestResolver resolver;

    private float lastPlayTime;

    //When a video needs decryption, most common in music videos
    private bool audioDecryptDone = false;
    private bool videoDecryptDone = false;

    private bool checkIfVideoArePrepared = false;
    private float lastVideoReadyToPlay = 0;

    //Video ready checkers
    private bool videoPrepared;
    private bool audioPrepared;

    //Retry checker
    private bool isRetry = false;

    YoutubeVideoQuality lastTryQuality = YoutubeVideoQuality.UHD2160;
    private string lastTryVideoId;

    private float lastStartedTime;
    private bool youtubeUrlReady = false;

    #endregion

    #region SERVER VARIABLES

    private YoutubeResultIds newRequestResults;

    /*PRIVATE INFO DO NOT CHANGE THESE URLS OR VALUES, ONLY IF YOU WANT HOST YOUR OWN
SERVER| TURORIALS IN THE PROJECT FILES*/
    private const string serverURI = "https://unity-dev-youtube.herokuapp.com/api/info?url=";
    private const string formatURI = "&format=best&flatten=true";
    private const string VIDEOURIFORWEBGLPLAYER =
"https://youtubewebgl.herokuapp.com/download.php?mime=video/mp4&title=generatedvideo&token=";
    /*END OF PRIVATE INFO*/

    #endregion

    #region Unity Functions
    public void Start()
```

```
    {
        FixCameraEvent();
        Skybox3DSettup();
        videoPlayer.skipOnDrop = true;
        audioPlayer.skipOnDrop = true;

#if UNITY_WEBGL
        ForceGetWebServer = true;
#endif

        PrepareVideoPlayerCallbacks();
        CheckRequestResolver();

        if (autoPlayOnStart)
        {

            PlayYoutubeVideo(youtubeUrl);
        }

        //VideoController Area
        if (videoQuality == YoutubeVideoQuality.LOW)
            lowRes = true;
        else
            lowRes = false;



    }

    private void TryToLoadThumbnailBeforeOpenVideo(string id)
    {
        string tempId = id.Replace("https://youtube.com/watch?v=", "");
        StartCoroutine(DownloadThumbnail(tempId));
    }

    IEnumerator DownloadThumbnail(string videoId)
    {
        WWW www = new WWW("https://img.youtube.com/vi/" + videoId + "/0.jpg");
        yield return www;
        Texture2D thumb = www.texture;
        videoPlayer.targetMaterialRenderer.material.mainTexture = thumb;
    }

    private void Skybox3DSettup()
    {
        if (is3DLayoutVideo)
        {
            if(layout3d == Layout3D.OverUnder)
            {
                RenderSettings.skybox =
(Material)Resources.Load("Materials/PanoramicSkybox3DOverUnder") as Material;
            }else if( layout3d == Layout3D.sideBySide)
            {
                RenderSettings.skybox = (Material)Resources.Load("Materials/PanoramicSkybox3Dside") as
Material;
            }
        }
    }

    private void FixCameraEvent()
    {
```

```
        if(mainCamera == null)
        {
            if (Camera.main != null)
                mainCamera = Camera.main;
            else
                Debug.LogError("Add the main camera to the mainCamera field");
        }

        videoControllerCanvas.GetComponent<Canvas>().worldCamera = mainCamera;
        if (videoPlayer.renderMode == VideoRenderMode.CameraFarPlane || videoPlayer.renderMode ==
VideoRenderMode.CameraNearPlane)
            videoPlayer.targetCamera = mainCamera;
    }

    //A workaround for mobile bugs.
    private void OnApplicationPause(bool pause)
    {
        if (videoPlayer.isPrepared)
        {
            if(audioPlayer != null)
                audioPlayer.Pause();

            videoPlayer.Pause();
        }

    }

    private void OnApplicationFocus(bool focus)
    {
        if (focus == true)
        {
            if (videoPlayer.isPrepared)
            {
                if(audioPlayer != null)
                    audioPlayer.Play();

                videoPlayer.Play();
            }

        }
    }

    private void OnEnable()
    {
        if (autoPlayOnEnable)
        {
            StartCoroutine(WaitThingsGetDone());
        }
    }

    IEnumerator WaitThingsGetDone()
    {
        yield return new WaitForSeconds(1);
        if (youtubeUrlReady && videoPlayer.isPrepared)
        {
            if (videoQuality == YoutubeVideoQuality.LOW)
                videoPlayer.Play();
            else
            {
                audioPlayer.Play();
                videoPlayer.Play();
```

```
        }
    }
    else
    {
        if (!youtubeUrlReady)
            LoadYoutubeVideo(youtubeUrl);

    }

}

void FixedUpdate()
{
    if (gettingYoutubeURL)
    {
        currentRequestTime += Time.deltaTime;
        if (currentRequestTime >= maxRequestTime)
        {
            gettingYoutubeURL = false;
            if(debug)
                Debug.Log("<color=blue>Max time reached, trying again!</color>");
            Debug.Break();
        }
    }

    //used this to play in main thread.
    if (videoAreReadyToPlay)
    {
        videoAreReadyToPlay = false;


    }

    ErrorCheck();

    //Video controller area
    if (showPlayerControls)
    {
        if (videoQuality != YoutubeVideoQuality.LOW)
            lowRes = false;
        else
            lowRes = true;

        if (currentTimeString != null && totalTimeString != null)
        {
            currentTimeString.text = FormatTime(Mathf.RoundToInt(currentVideoDuration));
            if (!lowRes)
            {
                if (audioDuration < totalVideoDuration && (audioPlayer.url != ""))
                    totalTimeString.text = FormatTime(Mathf.RoundToInt(audioDuration));
                else
                    totalTimeString.text = FormatTime(Mathf.RoundToInt(totalVideoDuration));
            }
            else
            {
                totalTimeString.text = FormatTime(Mathf.RoundToInt(totalVideoDuration));
            }

        }
    }
```

```
    if (!showPlayerControls)
    {
        mainControllerUi.SetActive(false);
    }
    else
        mainControllerUi.SetActive(true);
    //End video controller area

    if (decryptedUrlForAudio)
    {
        decryptedUrlForAudio = false;
        DecryptAudioDone(decryptedAudioUrlResult);
    }

    if (decryptedUrlForVideo)
    {
        decryptedUrlForVideo = false;
        DecryptVideoDone(decryptedVideoUrlResult);
    }

    //webgl
    if (videoPlayer.isPrepared)
    {
        if (!startedPlayingWebgl)
        {
            logTest = videoUrl + " Let's play";
            startedPlayingWebgl = true;
            StartCoroutine(WebGLPlay());
        }
    }

}

#endregion

#region ASSET FUNCTIONS

private void CheckRequestResolver()
{
    //if (gameObject.GetComponent<RequestResolver>() == null)
        resolver = this;
}

private void PrepareVideoPlayerCallbacks()
{
    videoPlayer.started += VideoStarted;
    videoPlayer.errorReceived += VideoErrorReceived;
    videoPlayer.frameDropped += VideoFrameDropped;
}


private void ShowLoading()
{
    if(loadingContent != null)
        loadingContent.SetActive(true);
}

private void HideLoading()
{
    if(loadingContent != null)
        loadingContent.SetActive(false);
```

```
}

public void LoadYoutubeVideo(string url)
{
    PlayYoutubeVideo(url);
}

private string CheckVideoUrlAndExtractThevideoId(string url)
{
    if (url.Contains("?t="))
    {
        int last = url.LastIndexOf("?t=");
        string copy = url;
        string newString = copy.Remove(0, last);
        newString = newString.Replace("?t=","");
        startFromSecond = true;
        startFromSecondTime = int.Parse(newString);
        url = url.Remove(last);
    }

    if (!url.Contains("youtu"))
    {
        url = "youtube.com/watch?v=" + url;
    }

    bool isYoutubeUrl = TryNormalizeYoutubeUrlLocal(url,out url);
    if (!isYoutubeUrl)
    {
        Debug.LogError("ITS NOT A YOUTUBE URL");
    }

    return url;
}

private bool TryNormalizeYoutubeUrlLocal(string url, out string normalizedUrl)
{
    url = url.Trim();
    url = url.Replace("youtu.be/", "youtube.com/watch?v=");
    url = url.Replace("www.youtube", "youtube");
    url = url.Replace("youtube.com/embed/", "youtube.com/watch?v=");

    if (url.Contains("/v/"))
    {
        url = "https://youtube.com" + new Uri(url).AbsolutePath.Replace("/v/", "/watch?v=");
    }

    url = url.Replace("/watch#", "/watch?");
    IDictionary<string, string> query = HTTPHelperYoutube.ParseQueryString(url);

    string v;


    if (!query.TryGetValue("v", out v))
    {
        normalizedUrl = null;
        return false;
    }

    normalizedUrl = "https://youtube.com/watch?v=" + v;

    return true;
```

```
    }

    private void PlayYoutubeVideo(string _videoId)
    {
        _videoId = CheckVideoUrlAndExtractThevideoId(_videoId);
        //Thumbnail
        if (showThumbnailBeforeVideoLoad)
            TryToLoadThumbnailBeforeOpenVideo(_videoId);
        youtubeUrlReady = false;
        //Show loading
        ShowLoading();

        youtubeUrl = _videoId;
        //loading for fist time, so it's not a retry
        isRetry = false;
        //store some variables to control
        lastTryQuality = videoQuality;
        lastTryVideoId = _videoId;
        lastPlayTime = Time.time;
        lastVideoReadyToPlay = 0;

#if UNITY_WEBGL
        StartCoroutine(WebGlRequest(youtubeUrl));
#else
        if (!ForceGetWebServer)
        {
            currentRequestTime = 0;
            gettingYoutubeURL = true;
            resolver.GetDownloadUrls(UrlsLoaded, youtubeUrl, this);
        }
        else
            StartCoroutine(WebRequest(youtubeUrl));


#endif
    }

    //When the audio decryption is done
    public void DecryptAudioDone(string url)
    {
        audioUrl = url;
        audioDecryptDone = true;

        if (videoDecryptDone)
        {
            videoAreReadyToPlay = true;

            OnYoutubeUrlsLoaded();
        }

    }

    //When the Video decryption is done
    public void DecryptVideoDone(string url)
    {
        videoUrl = url;
        videoDecryptDone = true;

        if (audioDecryptDone) {
            videoAreReadyToPlay = true;
            OnYoutubeUrlsLoaded();
```

```
    }
    else
    {
        if(videoQuality == YoutubeVideoQuality.LOW)
        {
            videoAreReadyToPlay = true;
            OnYoutubeUrlsLoaded();
        }
    }


}

//The callback when the url's are loaded.
private void UrlsLoaded()
{
    gettingYoutubeURL = false;
    List<VideoInfo> videoInfos = resolver.videoInfos;
    videoDecryptDone = false;
    audioDecryptDone = false;

    decryptedUrlForVideo = false;
    decryptedUrlForAudio = false;

    if(videoQuality == YoutubeVideoQuality.LOW)
    {
        //Get the video with audio first
        foreach (VideoInfo info in videoInfos)
        {
            if (info.VideoType == VideoType.Mp4 && info.Resolution == (360))
            {
                if (info.RequiresDecryption)
                {
                    //The string is the video url with audio
                    DecryptDownloadUrl(info.DownloadUrl, info.HtmlPlayerVersion, false);
                }
                else
                {
                    videoUrl = info.DownloadUrl;
                    videoAreReadyToPlay = true;
                    OnYoutubeUrlsLoaded();
                }
                break;
            }
        }
    }
    else
    {
        //Get the video with audio first
        foreach (VideoInfo info in videoInfos)
        {
            if (info.VideoType == VideoType.Mp4 && info.Resolution == (360))
            {

                if (info.RequiresDecryption)
                {
                    //The string is the video url with audio
                    DecryptDownloadUrl(info.DownloadUrl, info.HtmlPlayerVersion, true);
                }
                else
                {
```

```
            audioUrl = info.DownloadUrl;
        }
        break;
    }
}

//Then we will get the desired video quality.
int quality = 360;
switch (videoQuality)
{
    case YoutubeVideoQuality.LOW:
        quality = 360;
        break;
    case YoutubeVideoQuality.HD:
        quality = 720;
        break;
    case YoutubeVideoQuality.FULLHD:
        quality = 1080;
        break;
    case YoutubeVideoQuality.UHD1440:
        quality = 1440;
        break;
    case YoutubeVideoQuality.UHD2160:
        quality = 2160;
        break;
}

bool foundVideo = false;
//Get the high quality video
foreach (VideoInfo info in videoInfos)
{
    if (info.VideoType == VideoType.Mp4 && info.Resolution == (quality))
    {
        if (info.RequiresDecryption)
        {
            if (debug)
                Debug.Log("REQUIRE DECRYPTION!");
            //The string is the video url
            DecryptDownloadUrl(info.DownloadUrl, info.HtmlPlayerVersion, false);
        }
        else
        {
            videoUrl = info.DownloadUrl;
            videoAreReadyToPlay = true;
            OnYoutubeUrlsLoaded();
        }
        foundVideo = true;

        if (info.AudioType != YoutubeLight.AudioType.Unknown)//there's no audio atacched.
        {
            audioPlayer.audioOutputMode = VideoAudioOutputMode.None;
        }
        else
        {
            videoPlayer.audioOutputMode = VideoAudioOutputMode.None;
        }

        break;
    }
}
```

```
        if (!foundVideo && quality == 2160)
        {
            foreach (VideoInfo info in videoInfos)
            {
                if (info.FormatCode == 313)
                {
                    if (debug)
                        Debug.Log("Found but with unknow format in results, check to see if the video works
normal.");
                    if (info.RequiresDecryption)
                    {
                        //The string is the video url
                        DecryptDownloadUrl(info.DownloadUrl, info.HtmlPlayerVersion, false);
                    }
                    else
                    {
                        videoUrl = info.DownloadUrl;
                        videoAreReadyToPlay = true;
                        OnYoutubeUrlsLoaded();
                    }
                    foundVideo = true;

                    if (info.AudioType != YoutubeLight.AudioType.Unknown)//there's no audio atacched.
                    {
                        audioPlayer.audioOutputMode = VideoAudioOutputMode.None;
                    }
                    else
                    {
                        videoPlayer.audioOutputMode = VideoAudioOutputMode.None;
                    }

                    break;
                }
            }
        }

        //if desired quality not found try another lower quality.
        if (!foundVideo)
        {
            if (debug)
                Debug.Log("Desired quality not found, playing with low quality, check if the video id: " +
youtubeUrl + " support that quality!");
            foreach (VideoInfo info in videoInfos)
            {
                if (info.VideoType == VideoType.Mp4 && info.Resolution == (360))
                {
                    if (info.RequiresDecryption)
                    {
                        videoQuality = YoutubeVideoQuality.LOW;
                        //The string is the video url
                        DecryptDownloadUrl(info.DownloadUrl, info.HtmlPlayerVersion, false);
                    }
                    else
                    {
                        videoUrl = info.DownloadUrl;
                        videoAreReadyToPlay = true;
                        OnYoutubeUrlsLoaded();
                    }
                    if (info.AudioType != YoutubeLight.AudioType.Unknown)//there's no audio atacched.
                    {
                        audioPlayer.audioOutputMode = VideoAudioOutputMode.None;
```

```
                }
                else
                {
                    videoPlayer.audioOutputMode = VideoAudioOutputMode.None;
                }
                break;
            }
        }
    }
}

private void LoadPrepareCallbacks()
{
        videoPrepared = false;
        videoPlayer.prepareCompleted += VideoPrepared;
        if (videoQuality != YoutubeVideoQuality.LOW)
        {
            audioPrepared = false;
            audioPlayer.prepareCompleted += AudioPrepared;
        }
}

private void Play()
{
    videoPlayer.loopPointReached += PlaybackDone;
    StartPlayback();
}

private void StartPlayback()
{

    if (videoQuality != YoutubeVideoQuality.LOW)
    {
        audioPlayer.Play();
    }

    videoPlayer.Play();

   HideLoading();

    if (startFromSecond)
        videoPlayer.time = startFromSecondTime;
}


private void ErrorCheck()
{
    if (!ForceGetWebServer)
    {
        if (!isRetry && lastStartedTime < lastErrorTime && lastErrorTime > lastPlayTime)
        {
            if (debug)
                Debug.Log("Error detected!, retry with low quality!");
            lastTryQuality = YoutubeVideoQuality.LOW;
            RetryPlayYoutubeVideo();
        }
    }
}

//It's not in use, maybe can be usefull to you, it's just a test.
```

```
public int GetMaxQualitySupportedByDevice()
{
    if (Screen.orientation == ScreenOrientation.Landscape)
    {
        //use the height
        return Screen.currentResolution.height;
    }
    else if (Screen.orientation == ScreenOrientation.Portrait)
    {
        //use the width
        return Screen.currentResolution.width;
    }
    else
    {
        return Screen.currentResolution.height;
    }
}

IEnumerator WebRequest(string videoID)
{
    WWW request = new WWW(serverURI + "" + videoID + "" + formatURI);
    yield return request;
    newRequestResults = new YoutubeResultIds();
    var requestData = JSON.Parse(request.text);
    var videos = requestData["videos"][0]["formats"];
    newRequestResults.bestFormatWithAudioIncluded = requestData["videos"][0]["url"];
    for (int counter = 0; counter < videos.Count; counter++)
    {
        if (videos[counter]["format_id"] == "160")
        {
            newRequestResults.lowQuality = videos[counter]["url"];
        }
        else if (videos[counter]["format_id"] == "133")
        {
            newRequestResults.lowQuality = videos[counter]["url"];   //if have 240p quality overwrite the
144 quality as low quality.
        }
        else if (videos[counter]["format_id"] == "134")
        {
            newRequestResults.standardQuality = videos[counter]["url"];  //360p
        }
        else if (videos[counter]["format_id"] == "136")
        {
            newRequestResults.hdQuality = newRequestResults.bestFormatWithAudioIncluded;  //720p
        }
        else if (videos[counter]["format_id"] == "137")
        {
            newRequestResults.fullHdQuality = videos[counter]["url"];  //1080p
        }
        else if (videos[counter]["format_id"] == "266")
        {
            newRequestResults.ultraHdQuality = videos[counter]["url"];  //@2160p 4k
        }
        else if (videos[counter]["format_id"] == "139")
        {
            newRequestResults.audioUrl = videos[counter]["url"];  //AUDIO
        }
    }


        audioUrl = newRequestResults.bestFormatWithAudioIncluded;
```

```
            videoUrl = newRequestResults.bestFormatWithAudioIncluded;

        switch (videoQuality)
        {
            case YoutubeVideoQuality.LOW:
                videoUrl = newRequestResults.bestFormatWithAudioIncluded;
                break;
            case YoutubeVideoQuality.HD:
                videoUrl = newRequestResults.hdQuality;
                break;
            case YoutubeVideoQuality.FULLHD:
                videoUrl = newRequestResults.fullHdQuality;
                break;
            case YoutubeVideoQuality.UHD1440:
                videoUrl = newRequestResults.fullHdQuality;
                break;
            case YoutubeVideoQuality.UHD2160:
                videoUrl = newRequestResults.ultraHdQuality;
                break;
        }

        if (videoUrl == "")
            videoUrl = newRequestResults.bestFormatWithAudioIncluded;

#if UNITY_WEBGL
        videoUrl = ConvertToWebglUrl(videoUrl);
#endif
        videoAreReadyToPlay = true;
        OnYoutubeUrlsLoaded();
    }

    private string ConvertToWebglUrl(string url)
    {
        byte[] bytesToEncode = Encoding.UTF8.GetBytes(url);
        string encodedText = Convert.ToBase64String(bytesToEncode);
        if (debug)
            Debug.Log(url);
        string newUrl = VIDEOURIFORWEBGLPLAYER + "" + encodedText;
        return newUrl;
    }


    public void RetryPlayYoutubeVideo()
    {
        currentRetryTime++;
        if (currentRetryTime < retryTimeUntilToRequestFromServer)
        {
            if (!ForceGetWebServer)
            {
                StopIfPlaying();
                if (debug)
                    Debug.Log("Youtube Retrying...:" + lastTryVideoId);
                isRetry = true;
                ShowLoading();
                youtubeUrl = lastTryVideoId;
                PlayYoutubeVideo(youtubeUrl);
            }
        }
        else
        {
            currentRetryTime = 0;
```

```
        if (debug)
            Debug.Log("Playing From WEbServer. There's a error in the local player.");
        //Get from webserver becuase there's something wrong with the offline system.
        StopIfPlaying();
        ShowLoading();
        ForceGetWebServer = true;
        youtubeUrl = lastTryVideoId;
        PlayYoutubeVideo(youtubeUrl);
    }

}


private void StopIfPlaying()
{
    if (debug)
        Debug.Log("Stopping video");
    if (videoPlayer.isPlaying) { videoPlayer.Stop(); }
    if (audioPlayer.isPlaying) { audioPlayer.Stop(); }
}
#endregion

#region CALLBACKS

public void OnYoutubeUrlsLoaded()
{
    youtubeUrlReady = true;
    if (!loadYoutubeUrlsOnly) //If want to load urls only the video will not play
    {
        lastVideoReadyToPlay = Time.time;

        if (debug)
            Debug.Log("Play!!" + videoUrl);

        LoadPrepareCallbacks();
        lastVideoReadyToPlay = Time.time;
        videoPlayer.source = VideoSource.Url;
        videoPlayer.url = videoUrl;
        checkIfVideoArePrepared = true;
        videoPlayer.Prepare();
        if (videoQuality != YoutubeVideoQuality.LOW)
        {
            audioPlayer.source = VideoSource.Url;
            audioPlayer.url = audioUrl;
            audioPlayer.Prepare();
        }
    }
    OnYoutubeUrlAreReady.Invoke();
}

public void OnYoutubeVideoAreReadyToPlay()
{
    OnVideoReadyToStart.Invoke();
    Play();
}

public void OnVideoPlayerFinished()
{
    if (videoPlayer.isPrepared)
    {
        if (debug)
```

```
            Debug.Log("Finished");
        if (videoPlayer.isLooping)
        {
            videoPlayer.time = 0;
            videoPlayer.frame = 0;
            audioPlayer.time = 0;
            audioPlayer.frame = 0;
            videoPlayer.Play();
            audioPlayer.Play();
        }
        OnVideoFinished.Invoke();
    }
}

//UNITY VIDEO PLAYER CALLBACK
private void AudioPrepared(VideoPlayer vPlayer)
{
    audioPlayer.prepareCompleted -= AudioPrepared;
    audioPrepared = true;
    if (audioPrepared && videoPrepared)
        OnYoutubeVideoAreReadyToPlay();
}

//UNITY VIDEO PLAYER CALLBACK
private void VideoPrepared(VideoPlayer vPlayer)
{
    videoPlayer.prepareCompleted -= VideoPrepared;
    videoPrepared = true;
    if (videoQuality == YoutubeVideoQuality.LOW)
    {
        OnYoutubeVideoAreReadyToPlay();
    }
    else
    {
        if (audioPrepared && videoPrepared)
            OnYoutubeVideoAreReadyToPlay();
    }

}

//Unity Video player callback
private void PlaybackDone(VideoPlayer vPlayer)
{
    finished = true;
    OnVideoPlayerFinished();
}

private void VideoFrameDropped(VideoPlayer source)
{
    if (debug)
        Debug.Log("Youtube VideoFrameDropped!"); //[NOT IMPLEMENTED UNITY 2017.2]
}

private void VideoStarted(VideoPlayer source)
{
    lastStartedTime = Time.time;
    lastErrorTime = lastStartedTime;
    if (debug)
        Debug.Log("Youtube Video Started");

    //Render to more materials
```

```
    if(objectsToRenderTheVideoImage.Length > 0)
    {
        foreach (GameObject obj in objectsToRenderTheVideoImage)
        {
            obj.GetComponent<Renderer>().material.mainTexture = videoPlayer.texture;
        }
    }

    OnVideoStarted.Invoke();
}

float lastErrorTime;
private void VideoErrorReceived(VideoPlayer source, string message)
{
    lastErrorTime = Time.time;
    if (debug)
        Debug.Log("Youtube VideoErrorReceived!:" + message);
}

public void PlayPause()
{
    if (youtubeUrlReady && videoPlayer.isPrepared)
    {
        if (videoPlayer.isPlaying)
        {
            if (videoQuality == YoutubeVideoQuality.LOW)
            {
                videoPlayer.Pause();
            }
            else
            {
                videoPlayer.Pause();
                audioPlayer.Pause();
            }
            OnVideoPaused.Invoke();
        }
        else
        {
            if (videoQuality == YoutubeVideoQuality.LOW)
            {
                videoPlayer.Play();
            }
            else
            {
                videoPlayer.Play();
                audioPlayer.Play();
            }
        }
    }
}

#endregion

#region VIDEO CONTROLLER
[DrawIf("showPlayerControls", true)]
[Tooltip("Hide the controls if use not interact in desired time, 0 equals to not hide")]
public int autoHideControlsTime = 0;

[DrawIf("showPlayerControls", true)]
[Tooltip("The main controller ui parent")]
public GameObject mainControllerUi;
```

```
[DrawIf("showPlayerControls", true)]
[Tooltip("Slider with duration and progress")]
public Slider progressSlider;


[DrawIf("showPlayerControls", true)]
[Tooltip("Volume slider")]
public Slider volumeSlider;

[DrawIf("showPlayerControls", true)]
[Tooltip("Playback speed")]
public Slider playbackSpeed;

[DrawIf("showPlayerControls", true)]
[Tooltip("Current Time")]
public Text currentTimeString;

[DrawIf("showPlayerControls", true)]
[Tooltip("Total Time")]
public Text totalTimeString;

private float totalVideoDuration;
private float currentVideoDuration;
private bool videoSeekDone = false;
private bool videoAudioSeekDone = false;
private bool lowRes;
private float hideScreenTime = 0;
private float audioDuration;
private int savedTime = 0;
private bool useBackwardSyncImprovementWorkaround = false;
private bool finished = false;
private bool showingPlaybackSpeed = false;
private bool showingVolume = false;


private void Update()
{
    if (showPlayerControls)
    {
        if (videoPlayer.isPlaying && progressSlider != null)
        {
            totalVideoDuration = Mathf.RoundToInt(videoPlayer.frameCount / videoPlayer.frameRate);
            if (!lowRes)
            {
                audioDuration = Mathf.RoundToInt(audioPlayer.frameCount / audioPlayer.frameRate);
                if (audioDuration < totalVideoDuration && (audioPlayer.url != ""))
                {
                    currentVideoDuration = Mathf.RoundToInt(audioPlayer.frame / audioPlayer.frameRate);
                    progressSlider.maxValue = audioDuration;
                }
                else
                {
                    currentVideoDuration = Mathf.RoundToInt(videoPlayer.frame / videoPlayer.frameRate);
                    progressSlider.maxValue = totalVideoDuration;
                }
            }
            else
            {
                currentVideoDuration = Mathf.RoundToInt(videoPlayer.frame / videoPlayer.frameRate);
                progressSlider.maxValue = totalVideoDuration;
```

```
            }

            progressSlider.Set(currentVideoDuration);
        }

        //if (currentVideoDuration >= totalVideoDuration)
        //{
        //    if (!finished)
        //        Finished();
        //}

        if (autoHideControlsTime > 0)
        {
            if (UserInteract())
            {
                hideScreenTime = 0;
                if (mainControllerUi != null)
                    mainControllerUi.SetActive(true);
            }
            else
            {
                hideScreenTime += Time.deltaTime;
                if (hideScreenTime >= autoHideControlsTime)
                {
                    //not increment
                    hideScreenTime = autoHideControlsTime;
                    HideControllers();
                }
            }
        }
    }
}

private void HideControllers()
{
    if (mainControllerUi != null)
    {
        mainControllerUi.SetActive(false);
        showingVolume = false;
        showingPlaybackSpeed = false;
        volumeSlider.gameObject.SetActive(false);
        playbackSpeed.gameObject.SetActive(false);
    }
}

public void Seek()
{
    isSyncing = true;
    //check if can seek
    if (Mathf.RoundToInt(currentVideoDuration) != Mathf.RoundToInt(totalVideoDuration))
    {
        finished = false;
        if (videoPlayer.canSetTime && videoPlayer.canStep)
        {
            ShowLoading();
            //Pause the video to prevent audio error
            //workaround related to the unity but, when you seek backward the video there's a big delay
to seek:
            if (Application.isMobilePlatform)
            {
                float currentTime = (float)videoPlayer.time;
```

```
    if (Mathf.RoundToInt(progressSlider.value) > currentTime)
    {
        videoPlayer.Pause();
        if(!lowRes)
            audioPlayer.Pause();
    }
    else
    {
        if (useBackwardSyncImprovementWorkaround)
        {
            videoPlayer.Stop();
            if (!lowRes)
                audioPlayer.Stop();

            savedTime = Mathf.RoundToInt(progressSlider.value);
            StartCoroutine(WorkAroundToUnityBackwardSeek());
        }
        else
        {
            videoPlayer.Pause();
            if (!lowRes)
                audioPlayer.Pause();
        }
    }
}
else
{
    videoPlayer.Pause();
    if (!lowRes)
        audioPlayer.Pause();
}


//reset variables
videoSeekDone = false;
videoAudioSeekDone = false;
if (!lowRes)
{
    //callbacks
    audioPlayer.seekCompleted += SeekVideoAudioDone;
    videoPlayer.seekCompleted += SeekVideoDone;
    //change the time
    if (Mathf.RoundToInt(progressSlider.value) == 0)
    {
        audioPlayer.time = 1;
        videoPlayer.time = 1;
    }
    else
    {
        audioPlayer.time = Mathf.RoundToInt(progressSlider.value);
        videoPlayer.time = Mathf.RoundToInt(progressSlider.value);
    }

}
else
{
    //callback
    videoPlayer.seekCompleted += SeekVideoDone;
    if (Mathf.RoundToInt(progressSlider.value) == 0)
        videoPlayer.time = 1;
    else
```

```
                videoPlayer.time = Mathf.RoundToInt(progressSlider.value);
            }
        }
    }
    else
    {
        //              if (sourceVideo.isPlaying) {
        //                  if (!finished)
        //                      Finished();
        //              }
    }
}


public void Finished()
{
    finished = true;
    OnVideoPlayerFinished();
}

public void Volume()
{
    if (videoPlayer.audioOutputMode == VideoAudioOutputMode.Direct)
        audioPlayer.SetDirectAudioVolume(0, volumeSlider.value);
    else if (videoPlayer.audioOutputMode == VideoAudioOutputMode.AudioSource)
        videoPlayer.GetComponent<AudioSource>().volume = volumeSlider.value;
    else
        videoPlayer.GetComponent<AudioSource>().volume = volumeSlider.value;
}

public void Speed()
{
    if (videoPlayer.canSetPlaybackSpeed)
    {
        if (playbackSpeed.value == 0)
        {
            videoPlayer.playbackSpeed = 0.5f;
            audioPlayer.playbackSpeed = 0.5f;
        }
        else
        {
            videoPlayer.playbackSpeed = playbackSpeed.value;
            audioPlayer.playbackSpeed = playbackSpeed.value;
        }
    }
}

public void PlayButton()
{
    if (!videoPlayer.isPlaying)
    {

#if !UNITY_WEBGL
        if (!lowRes)
        {
            audioPlayer.time = videoPlayer.time;
            audioPlayer.frame = videoPlayer.frame;
        }
        PlayController();
#else
        PlayController();
```

```
#endif

        }
        else
        {
#if !UNITY_WEBGL
            Pause();
#else
            Pause();
#endif
        }

    }


    public void VolumeSlider()
    {
        if (showingVolume)
        {
            showingVolume = false;
            volumeSlider.gameObject.SetActive(false);
        }
        else
        {
            showingVolume = true;
            volumeSlider.gameObject.SetActive(true);
        }
    }

    public void PlaybackSpeedSlider()
    {
        if (showingPlaybackSpeed)
        {
            showingPlaybackSpeed = false;
            playbackSpeed.gameObject.SetActive(false);
        }
        else
        {
            showingPlaybackSpeed = true;
            playbackSpeed.gameObject.SetActive(true);
        }
    }


    public void Pause()
    {
        videoPlayer.Pause();
        if (!lowRes)
        {
            audioPlayer.Pause();
        }

    }


    IEnumerator WorkAroundToUnityBackwardSeek()
    {
        yield return new WaitForSeconds(0.1f);
        videoPrepared = false;
        audioPrepared = false;
        if(!lowRes)
```

```
        audioPlayer.prepareCompleted += AudioPreparedSeek;
    videoPlayer.prepareCompleted += VideoPreparedSeek;
    if(!lowRes)
        audioPlayer.Prepare();
    videoPlayer.Prepare();
}

void VideoPreparedSeek(VideoPlayer p)
{
    videoPrepared = true;
    videoPlayer.prepareCompleted -= VideoPrepared;
    if (videoPrepared && audioPrepared)
    {
        progressSlider.value = savedTime;
    }

}

void AudioPreparedSeek(VideoPlayer p)
{
    audioPrepared = true;
    audioPlayer.prepareCompleted -= AudioPrepared;
    if (videoPrepared && audioPrepared)
    {
        progressSlider.value = savedTime;
    }
}

public void Stop()
{
    videoPlayer.Stop();
    if (!lowRes)
        audioPlayer.Stop();
}

void SeekVideoDone(VideoPlayer vp)
{
    videoSeekDone = true;
    videoPlayer.seekCompleted -= SeekVideoDone;
    if (!lowRes)
    {
        //check if the two videos are done the seek | if are play the videos
        if (videoSeekDone && videoAudioSeekDone)
        {
            isSyncing = false;
            //long frame = sourceVideo.frame;
            //sourceVideo.frame = frame;
            //sourceAudioVideo.frame = frame;
            StartCoroutine(SeekFinished());

            //HideLoading();
            //Play();
        }
    }
    else
    {
        isSyncing = false;

        HideLoading();
        PlayController();
    }
```

```
}

void SeekVideoAudioDone(VideoPlayer vp)
{
    videoAudioSeekDone = true;
    audioPlayer.seekCompleted -= SeekVideoAudioDone;
    if (!lowRes)
    {
        if (videoSeekDone && videoAudioSeekDone)
        {
            isSyncing = false;
            //long frame = sourceVideo.frame;
            //sourceVideo.frame = frame;
            //sourceAudioVideo.frame = frame;
            StartCoroutine(SeekFinished());

            //HideLoading();
            //Play();
        }
    }
}

IEnumerator SeekFinished()
{
    yield return new WaitForEndOfFrame();
    HideLoading();
    PlayController();
}

private string FormatTime(int time)
{
    int hours = time / 3600;
    int minutes = (time % 3600) / 60;
    int seconds = (time % 3600) % 60;
    if (hours == 0 && minutes != 0)
    {
        return minutes.ToString("00") + ":" + seconds.ToString("00");
    }
    else if (hours == 0 && minutes == 0)
    {
        return "00:" + seconds.ToString("00");
    }
    else
    {
        return hours.ToString("00") + ":" + minutes.ToString("00") + ":" + seconds.ToString("00");
    }
}

bool UserInteract()
{
    if (Application.isMobilePlatform)
    {
        if (Input.touches.Length >= 1)
            return true;
        else
            return false;
    }
    else
    {
        if (Input.GetMouseButtonDown(0))
            return true;
```

```
            return (Input.GetAxis("Mouse X") != 0) || (Input.GetAxis("Mouse Y") != 0);
        }

    }


    private void PlayController()
    {
        if (videoQuality != YoutubeVideoQuality.LOW)
        {
            audioPlayer.Play();
        }

        videoPlayer.Play();
    }

    #endregion

    #region Decryption System
    public void DecryptDownloadUrl(string encryptedUrl, string htmlVersion, bool audioDecryption)
    {
        if (audioDecryption)
            EncryptUrlForAudio = encryptedUrl;
        else
            EncryptUrlForVideo = encryptedUrl;

        IDictionary<string, string> queries = HTTPHelperYoutube.ParseQueryString(encryptedUrl);
        if (queries.ContainsKey(SignatureQuery))
        {
            if (audioDecryption)
                encryptedSignatureAudio = queries[SignatureQuery];
            else
                encryptedSignatureVideo = queries[SignatureQuery];

            //decrypted = GetDecipheredSignature( encryptedSignature);
            //MagicHands.DecipherWithVersion(encryptedSignature, videoInfo.HtmlPlayerVersion);
            //string jsUrl = string.Format("http://s.ytimg.com/yts/jsbin/{0}-{1}.js",
videoInfo.HtmlscriptName, videoInfo.HtmlPlayerVersion);
            string jsUrl = string.Format("http://s.ytimg.com/yts/jsbin/player{0}.js", htmlVersion);
            if (audioDecryption)
                StartCoroutine(Downloader(jsUrl, true));
            else
                StartCoroutine(Downloader(jsUrl, false));

        }

    }

    private Thread thread1;
    public void ReadyForExtract(string r, bool audioExtract)
    {
        if (audioExtract)
        {
            SetMasterUrlForAudio(r);
#if UNITY_WEBGL
            DoRegexFunctionsForAudio();
#else
            if (SystemInfo.processorCount > 1)
            {
                thread1 = new Thread(DoRegexFunctionsForAudio);
                thread1.Start();
```

```
            }
            else
            {
                DoRegexFunctionsForAudio();
            }
#endif


        }
        else
        {
            SetMasterUrlForVideo(r);
#if UNITY_WEBGL
            DoRegexFunctionsForVideo();
#else
            if (SystemInfo.processorCount > 1)
            {
                thread1 = new Thread(DoRegexFunctionsForVideo);
                thread1.Start();
            }
            else
            {
                DoRegexFunctionsForVideo();
            }
#endif

        }



    }

    IEnumerator Downloader(string uri, bool audio)
    {
        UnityWebRequest request = UnityWebRequest.Get(uri);
        request.SetRequestHeader("User-Agent", "Mozilla/5.0 (X11; Linux x86_64; rv:10.0)
Gecko/20100101 Firefox/10.0 (Chrome)");
        yield return request.Send();
        ReadyForExtract(request.downloadHandler.text, audio);
    }
    #endregion

    #region WEBGLREQUEST
    YoutubeResultIds webGlResults;
    IEnumerator WebGlRequest(string videoID)
    {
        WWW request = new WWW(serverURI + "" + videoID + "" + formatURI);
        startedPlayingWebgl = false;
        yield return request;
        webGlResults = new YoutubeResultIds();
        Debug.Log(request.url);
        var requestData = JSON.Parse(request.text);
        var videos = requestData["videos"][0]["formats"];
        webGlResults.bestFormatWithAudioIncluded = requestData["videos"][0]["url"];
        logTest = "EEDone";
        for (int counter = 0; counter < videos.Count; counter++)
        {
            if (videos[counter]["format_id"] == "160")
            {
                webGlResults.lowQuality = videos[counter]["url"];
            }
```

```
            else if (videos[counter]["format_id"] == "133")
            {
                webGlResults.lowQuality = videos[counter]["url"];   //if have 240p quality overwrite the 144 quality as low quality.
            }
            else if (videos[counter]["format_id"] == "134")
            {
                webGlResults.standardQuality = videos[counter]["url"];  //360p
            }
            else if (videos[counter]["format_id"] == "136")
            {
                webGlResults.hdQuality = videos[counter]["url"];  //720p
            }
            else if (videos[counter]["format_id"] == "137")
            {
                webGlResults.fullHdQuality = videos[counter]["url"];  //1080p
            }
            else if (videos[counter]["format_id"] == "266")
            {
                webGlResults.ultraHdQuality = videos[counter]["url"];  //@2160p 4k
            }
            else if (videos[counter]["format_id"] == "139")
            {
                webGlResults.audioUrl = videos[counter]["url"];  //AUDIO
            }
        }
        //quality selection will be implemented later for webgl, i recomend use the webGlResults.bestFormatWithAudioIncluded
        WebGlGetVideo(webGlResults.bestFormatWithAudioIncluded);
    }



    //WEBGL only...
    public void WebGlGetVideo(string url)
    {
        logTest = "Getting Url Player";
        byte[] bytesToEncode = Encoding.UTF8.GetBytes(url);
        string encodedText = Convert.ToBase64String(bytesToEncode);
        videoUrl = VIDEOURIFORWEBGLPLAYER + "" + encodedText;
        videoQuality = YoutubeVideoQuality.LOW;
        logTest = videoUrl+" Done";
        Debug.Log("Play!! " + videoUrl);
        videoPlayer.source = VideoSource.Url;
        videoPlayer.url = videoUrl;
        videoPlayer.Prepare();
        videoPrepared = false;
        videoPlayer.prepareCompleted += WeblPrepared; ;
    }

    private void WeblPrepared(VideoPlayer source)
    {
        startedPlayingWebgl = true;
        StartCoroutine(WebGLPlay());
        logTest = "Playing!!";
    }

    IEnumerator WebGLPlay() //The prepare not respond so, i forced to play after some seconds
    {
        yield return new WaitForSeconds(2f);
        Play();
```

```
    }
    bool startedPlayingWebgl = false;

    public class YoutubeResultIds
    {
        public string lowQuality;
        public string standardQuality;
        public string mediumQuality;
        public string hdQuality;
        public string fullHdQuality;
        public string ultraHdQuality;
        public string bestFormatWithAudioIncluded;
        public string audioUrl;

    }

    private string logTest = "/";
    private void OnGUI()
    {
        if(debug)
            GUI.Label(new Rect(0, 0, 400, 30), logTest);
    }
    #endregion
    [HideInInspector]
    public bool isSyncing = false;

    [Header("Experimental")]
    public bool showThumbnailBeforeVideoLoad = false;
    private string thumbnailVideoID;
}
```

```
#if UNITY_EDITOR

using UnityEngine;
using UnityEditor;
using System.Collections.Generic;
using Assets.OVR.Scripts;

public class OVRProfiler : EditorWindow
{
	enum TargetPlatform
	{
		OculusGo,
		GearVR,
		SantaCruz,
		OculusRift
	};

	private static List<RangedRecord> mRecords = new List<RangedRecord>();
	private Vector2 mScrollPosition;
	static private TargetPlatform mTargetPlatform;

	[MenuItem("Tools/Oculus/OVR Profiler")]
	static void Init()
	{
		// Get existing open window or if none, make a new one:
		EditorWindow.GetWindow(typeof(OVRProfiler));
#if UNITY_ANDROID
		mTargetPlatform = TargetPlatform.OculusGo;
#else
		mTargetPlatform = TargetPlatform.OculusRift;
#endif
	}

	void OnGUI()
	{
		GUILayout.Label("OVR Profiler", EditorStyles.boldLabel);
		string[] options = new string[]
		{
			"Oculus Go", "Gear VR", "Santa Cruz", "Oculus Rift",
		};
		mTargetPlatform = (TargetPlatform)EditorGUILayout.Popup("Target Oculus Platform",
(int)mTargetPlatform, options);

		if (EditorApplication.isPlaying)
		{
			UpdateRecords();
			DrawResults();
		}
		else
		{
			ShowCenterAlignedMessageLabel("Click Run in Unity to view stats.");
		}
	}

	void OnInspectorUpdate()
	{
		Repaint();
```

```
    }

    void DrawResults()
    {
        string lastCategory = "";

        mScrollPosition = EditorGUILayout.BeginScrollView(mScrollPosition);

        foreach (RangedRecord record in mRecords)
        {
            // Add separator and label for new category
            if (!record.category.Equals(lastCategory))
            {
                lastCategory = record.category;
                EditorGUILayout.Separator();
                EditorGUILayout.BeginHorizontal();
                GUILayout.Label(lastCategory, EditorStyles.label, GUILayout.Width(200));
                EditorGUILayout.EndHorizontal();
            }

            // Draw records
            EditorGUILayout.BeginHorizontal();
            Rect r = EditorGUILayout.BeginVertical();
            EditorGUI.ProgressBar(r, record.value / (record.max * 2), record.category + " " +
record.value.ToString());
            GUILayout.Space(16);
            EditorGUILayout.EndVertical();
            EditorGUILayout.EndHorizontal();

            EditorGUILayout.BeginHorizontal();
            GUILayout.Label(record.message);
            EditorGUILayout.EndHorizontal();

            GUI.enabled = true;

        }

        EditorGUILayout.EndScrollView();
    }

    private void UpdateRecords()
    {
        mRecords.Clear();

        if (mTargetPlatform == TargetPlatform.OculusRift)
        {
            AddRecord("Client Frame CPU Time (ms)", "", UnityStats.frameTime * 1000, 0, 11);
            AddRecord("Render Frame CPU Time (ms)", "", UnityStats.renderTime * 1000, 0, 11);
        }
        else
        {
            // Graphics memory
            long memSizeByte = UnityStats.usedTextureMemorySize + UnityStats.vboTotalBytes;
            AddRecord("Graphics Memory (MB)", "Please use less than 1024 MB of vertex and texture
memory.", ConvertBytes(memSizeByte, "MB"), 0, 1024);
        }
```

```
        float triVertRec = mTargetPlatform == TargetPlatform.OculusRift ? 1000000 : 100000;
        // Triangle count
        AddRecord("Triangles", "Please use less than 100000 triangles.", UnityStats.triangles, 0, triVertRec);

        // Vertices count
        AddRecord("Vertices", "Please use less than 100000 vertices.", UnityStats.vertices, 0, triVertRec);

        float dcRec = mTargetPlatform == TargetPlatform.OculusRift ? 1000 : 100;
        // Draw call count
        AddRecord("Draw Call", "Please use less than 100 draw calls.", UnityStats.drawCalls, 0, dcRec);
    }

    private string FormatBytes(long bytes, string target)
    {
        return System.String.Format("{0:0.##} {1}", ConvertBytes(bytes, target), target);
    }

    private float ConvertBytes(long bytes, string target)
    {
        string[] Suffix = { "B", "KB", "MB", "GB", "TB" };
        int i;
        double dblSByte = bytes;
        for (i = 0; i < Suffix.Length; i++, bytes /= 1024)
        {
            if (Suffix[i] == target)
                return (float)dblSByte;
            dblSByte = bytes / 1024.0;
        }
        return 0;
    }

    private void ShowCenterAlignedMessageLabel(string message)
    {
        GUILayout.BeginVertical();
        GUILayout.FlexibleSpace();
        GUILayout.BeginHorizontal();
        GUILayout.FlexibleSpace();
        GUILayout.Label(message, EditorStyles.boldLabel);
        GUILayout.FlexibleSpace();
        GUILayout.EndHorizontal();
        GUILayout.FlexibleSpace();
        GUILayout.EndVertical();
    }

    private void AddRecord(string category, string message, float value, float min, float max)
    {
        RangedRecord record = new RangedRecord(category, message, value, min, max);
        mRecords.Add(record);
    }
}

#endif
```

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

// Hack to animate the line renderer.
public class AnimateLine : MonoBehaviour {
    public Material m;

    private Vector2 uvAnimationRate = new Vector2(-2.0f, 0.0f);
    private string textureName = "_MainTex";
    private Vector2 uvOffset = Vector2.zero;

    void LateUpdate() {
        uvOffset += (Time.deltaTime * uvAnimationRate);
        m.SetTextureOffset(textureName, uvOffset);
    }
}
```

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class AnimateTeleportPoint : MonoBehaviour {
    public Material m;
    public Shader overlayShader;

    private float baseScale = 0.25f;
    private float oscillateFactor = 0.1f;

        void Start () {
        m.shader = overlayShader;
        StartCoroutine(InitiateAnimation());
        }

    private void OnEnable() {
        StartCoroutine(InitiateAnimation());
    }

    private void OnDisable() {
        StopAllCoroutines();
    }

    // Scale up the point to the target scale.
    private IEnumerator InitiateAnimation() {
        gameObject.transform.localScale = Vector3.zero;
        float currScale = 0f;
        Vector3 scaleVec = new Vector3(0, 0, 0);

        while (currScale < baseScale) {
            yield return 0;
            currScale += (Time.deltaTime * 0.5f);
            scaleVec.x = scaleVec.y = scaleVec.z = currScale;
            Debug.Log(currScale);
            gameObject.transform.localScale = scaleVec;
        }
        StartCoroutine(Oscillate());
    }

    // Oscillate the size of the teleport point.
    private IEnumerator Oscillate() {
        float startTime = Time.time;
        Vector3 scaleVec = gameObject.transform.localScale;

        while (true) {
            float factor = Mathf.Sin( (Time.time - startTime) * 2f);
            scaleVec.x = scaleVec.y = scaleVec.z = (baseScale + factor * oscillateFactor);
            yield return 0;
            gameObject.transform.localScale = scaleVec;
        }
    }
}
```

```
using UnityEngine;
using System.Collections.Generic;
[RequireComponent(typeof(LineRenderer))]
public class Bezier : MonoBehaviour {
    public bool endPointDetected;

    public Vector3 EndPoint {
        get { return endpoint; }
    }

    public float ExtensionFactor {
        set { extensionFactor = value; }
    }

    private Vector3 endpoint;
    private float extensionFactor;
    private Vector3[] controlPoints;
    private LineRenderer lineRenderer;
    private float extendStep;
    private int SEGMENT_COUNT = 50;

    void Start() {
        controlPoints = new Vector3[3];
        lineRenderer = GetComponent<LineRenderer>();
        lineRenderer.enabled = false;
        extendStep = 5f;
        extensionFactor = 0f;
    }

    void Update() {
        UpdateControlPoints();
        HandleExtension();
        DrawCurve();
    }

    public void ToggleDraw(bool draw) {
        lineRenderer.enabled = draw;
    }

    void HandleExtension() {
        if (extensionFactor == 0f)
            return;

        float finalExtension = extendStep + Time.deltaTime * extensionFactor * 2f;
        extendStep = Mathf.Clamp(finalExtension, 2.5f, 7.5f);
    }

    // The first control is the remote. The second is a forward projection. The third is a forward and
downward projection.
    void UpdateControlPoints() {
        controlPoints[0] = gameObject.transform.position; // Get Controller Position
        controlPoints[1] = controlPoints[0] + (gameObject.transform.forward * extendStep * 2f / 5f);
        controlPoints[2] = controlPoints[1] + (gameObject.transform.forward * extendStep * 3f / 5f) +
Vector3.up * -1f;
    }


    // Draw the bezier curve.
    void DrawCurve() {
        if (!lineRenderer.enabled)
```

```
            return;
        lineRenderer.positionCount = 1;
        lineRenderer.SetPosition(0, controlPoints[0]);

        Vector3 prevPosition = controlPoints[0];
        Vector3 nextPosition = prevPosition;

        float tIncrement = 1 / (float)SEGMENT_COUNT;
        float t = 0f;

        for (int i = 1; i <= SEGMENT_COUNT*2; i++) {
            t += tIncrement;
            lineRenderer.positionCount = i + 1;

            if (i == SEGMENT_COUNT) { // For the last point, project out the curve two more meters.
                Vector3 endDirection = Vector3.Normalize(prevPosition - lineRenderer.GetPosition(i-2));
                nextPosition = prevPosition + endDirection * 2f;
            } else {
                nextPosition = CalculateBezierPoint(t, controlPoints[0], controlPoints[1], controlPoints[2]);
            }
/*
        Vector3 prevPosition = controlPoints[0];
        Vector3 nextPosition = prevPosition;
        for (int i = 1; i <= SEGMENT_COUNT; i++) {
            float t = i / (float) SEGMENT_COUNT;
            lineRenderer.positionCount = i + 1;

            if (i == SEGMENT_COUNT) { // For the last point, project out the curve two more meters.
                Vector3 endDirection = Vector3.Normalize(prevPosition - lineRenderer.GetPosition(i-2));
                nextPosition = prevPosition + endDirection * 2f;
            } else {
                nextPosition = CalculateBezierPoint(t, controlPoints[0], controlPoints[1], controlPoints[2]);
            }
 */
            int aux = CheckColliderIntersection(prevPosition, nextPosition);
            if (aux == 1) { // Surface Detected  == Ground
                lineRenderer.SetPosition(i, endpoint);
                endPointDetected = true;
                return;
            }
            if(aux == 2) // Surface Detected  != Ground
            {
                lineRenderer.SetPosition(i, endpoint);
                endPointDetected = false;
                return;
            }
            if(aux == 3) { // If the point does not intersect, continue to draw the curve.
                lineRenderer.SetPosition(i, nextPosition);
                endPointDetected = false;
                prevPosition = nextPosition;
            }
        }
    }

    // Check if the line between start and end intersect a collider.
    int CheckColliderIntersection(Vector3 start, Vector3 end) {
        Ray r = new Ray(start, end - start);
        RaycastHit hit;
        if (Physics.Raycast(r, out hit, Vector3.Distance(start, end))) {
            if (hit.collider.tag == "Hi")
                {
```

```
                endpoint = hit.point;
            return 1;
        }
        else{
           // endpoint = hit.point;
            return 2;
        }
    }

    return 3;
  }

  Vector3 CalculateBezierPoint(float t, Vector3 p0, Vector3 p1, Vector3 p2) {
    return
        Mathf.Pow((1f - t), 2) * p0 +
        2f * (1f - t) * t * p1 +
        Mathf.Pow(t, 2) * p2;
  }
}
```

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Door : MonoBehaviour {

    public GameObject player;
    bool doorOpened;
    bool closeEnough, doorClosed;
    float detectionRange = 2.0f;
    Animator anim;

        // Use this for initialization
        void Start () {
        closeEnough = false;
        doorOpened = false;
        doorClosed = true;
        }

        // Update is called once per frame
        void Update () {
        if (Vector3.Distance(player.transform.position, transform.position) <= detectionRange)
        {
            closeEnough = true;
        }
        else
        {
            closeEnough = false;
        }


        if(!closeEnough && !doorClosed)
        {
            anim = gameObject.GetComponent<Animator>();
            anim.SetBool("IsOpen", false);
            //Close the door
            /*
            Quaternion startRotation = this.transform.rotation;
            Quaternion endRotation = Quaternion.Euler(startRotation.eulerAngles.x,
startRotation.eulerAngles.y - angles, startRotation.eulerAngles.z);

            for (float t = 0; t < duration; t += Time.deltaTime)
            {
                this.transform.rotation = Quaternion.Lerp(startRotation, endRotation, t / duration);
            }
            this.transform.rotation = endRotation;
            */
            doorOpened = false;
            doorClosed = true;
        }

        if (closeEnough && !doorOpened)
        {
            anim = gameObject.GetComponent<Animator>();
            anim.SetBool("IsOpen", true);

            //Open the door
            /*
            Quaternion startRotation = this.transform.rotation;
            Quaternion endRotation = Quaternion.Euler(startRotation.eulerAngles.x,
```

```
startRotation.eulerAngles.y + angles, startRotation.eulerAngles.z) ;

            for (float t = 0; t < duration; t += Time.deltaTime)
            {
                this.transform.rotation = Quaternion.Lerp(startRotation, endRotation, t / duration);
            }
            this.transform.rotation = endRotation;
            */
            doorOpened = true;
            doorClosed = false;
        }


    }
    /*
    void OnCollisionEnter(Collision collision)
    {
        int angles = 90;
        Quaternion startRotation = this.transform.rotation;
        Quaternion endRotation = Quaternion.Euler(0,angles,0) * startRotation;

        this.transform.rotation = endRotation;


    for (float t = 0; t < duration; t += Time.deltaTime)
    {
        this.transform.rotation = Quaternion.Lerp(startRotation, endRotation, t / duration);
        yield return null;
    }

    }
    */
}
```

```
/*using UnityEngine;
using System.Collections.Generic;
[RequireComponent(typeof(LineRenderer))]
public class Bezier : MonoBehaviour
{
    public bool endPointDetected;

    public Vector3 EndPoint
    {
        get { return endpoint; }
    }

    public float ExtensionFactor
    {
        set { extensionFactor = value; }
    }

    private Vector3 endpoint;
    private float extensionFactor;
    private Vector3[] controlPoints;
    private LineRenderer lineRenderer;
    private float extendStep;
    private int SEGMENT_COUNT = 50;

    void Start()
    {
        controlPoints = new Vector3[3];
        lineRenderer = GetComponent<LineRenderer>();
        lineRenderer.enabled = false;
        extendStep = 5f;
        extensionFactor = 0f;
    }

    void Update()
    {
        UpdateControlPoints();
        HandleExtension();
        DrawCurve();
    }

    public void ToggleDraw(bool draw)
    {
        lineRenderer.enabled = draw;
    }

    void HandleExtension()
    {
        if (extensionFactor == 0f)
            return;

        float finalExtension = extendStep + Time.deltaTime * extensionFactor * 2f;
        extendStep = Mathf.Clamp(finalExtension, 2.5f, 7.5f);
    }

    // The first control is the remote. The second is a forward projection. The third is a forward and
downward projection.
    void UpdateControlPoints()
    {
        controlPoints[0] = gameObject.transform.position; // Get Controller Position
        controlPoints[1] = controlPoints[0] + (gameObject.transform.forward * extendStep * 2f / 5f);
```

```
        controlPoints[2] = controlPoints[1] + (gameObject.transform.forward * extendStep * 3f / 5f) +
Vector3.up * -1f;
    }


    // Draw the bezier curve.
    void DrawCurve()
    {
        if (!lineRenderer.enabled)
            return;
        lineRenderer.positionCount = 1;
        lineRenderer.SetPosition(0, controlPoints[0]);

        Vector3 prevPosition = controlPoints[0];
        Vector3 nextPosition = prevPosition;
        for (int i = 1; i <= SEGMENT_COUNT; i++)
        {
            float t = i / (float)SEGMENT_COUNT;
            lineRenderer.positionCount = i + 1;

            if (i == SEGMENT_COUNT)
            { // For the last point, project out the curve two more meters.
                Vector3 endDirection = Vector3.Normalize(prevPosition - lineRenderer.GetPosition(i - 2));
                nextPosition = prevPosition + endDirection * 2f;
            }
            else
            {
                nextPosition = CalculateBezierPoint(t, controlPoints[0], controlPoints[1], controlPoints[2]);
            }

            if (CheckColliderIntersection(prevPosition, nextPosition))
            { // If the segment intersects a surface, draw the point and return.
                lineRenderer.SetPosition(i, endpoint);
                endPointDetected = true;
                return;
            }
            else
            { // If the point does not intersect, continue to draw the curve.
                lineRenderer.SetPosition(i, nextPosition);
                endPointDetected = false;
                prevPosition = nextPosition;
            }
        }
    }

    // Check if the line between start and end intersect a collider.
    bool CheckColliderIntersection(Vector3 start, Vector3 end)
    {
        Ray r = new Ray(start, end - start);
        RaycastHit hit;
        if (Physics.Raycast(r, out hit, Vector3.Distance(start, end)))
        {
            if (hit.collider.tag == "Hi")
            {
                endpoint = hit.point;
                return true;
            }

            else
            {
                return false;
```

```
            }
        }

        return false;
    }

    Vector3 CalculateBezierPoint(float t, Vector3 p0, Vector3 p1, Vector3 p2)
    {
        return
            Mathf.Pow((1f - t), 2) * p0 +
            2f * (1f - t) * t * p1 +
            Mathf.Pow(t, 2) * p2;
    }
}
*/
```

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Teleport : MonoBehaviour
{
    public bool TeleportEnabled
    {
        get { return teleportEnabled; }
    }

    public Bezier bezier;
    public GameObject teleportSprite;
    public GameObject raycastSprite;
    private bool teleportEnabled;
    private bool firstClick;
    private float firstClickTime;
    private float doubleClickTimeLimit = 0.5f;


    void Start()
    {
        teleportEnabled = false;
        firstClick = false;
        firstClickTime = 0f;
        teleportSprite.SetActive(false);
    }

    void Update()
    {
        UpdateTeleportEnabled();

        if (teleportEnabled)
        {
            HandleBezier();
            HandleTeleport();
        }
    }

    // On double-click, toggle teleport mode on and off.
    void UpdateTeleportEnabled()
    {
        if (OVRInput.GetDown(OVRInput.Button.PrimaryIndexTrigger))
        { // The trigger is pressed.
            if (!firstClick)
            { // The first click is detected.
                firstClick = true;
                firstClickTime = Time.unscaledTime;
            }
            else
            { // The second click detected, so toggle teleport mode.
                firstClick = false;
                ToggleTeleportMode();
            }
        }

        if (Time.unscaledTime - firstClickTime > doubleClickTimeLimit)
        { // Time for the double click has run out.
            firstClick = false;
        }
```

```
    }


    void HandleTeleport()
    {
        if (bezier.endPointDetected)
        { // There is a point to teleport to.
            // Display the teleport point.
            teleportSprite.SetActive(true);
            teleportSprite.transform.position = bezier.EndPoint;

            if (OVRInput.GetDown(OVRInput.Button.One)) // Teleport to the position.
                TeleportToPosition(bezier.EndPoint);
        }
        else
        {
            teleportSprite.SetActive(false);
        }
    }

    void TeleportToPosition(Vector3 teleportPos)
    {
        gameObject.transform.position = teleportPos + Vector3.up * 0.5f;
    }

    // Optional: use the touchpad to move the teleport point closer or further.
    void HandleBezier()
    {
        Vector2 touchCoords = OVRInput.Get(OVRInput.Axis2D.PrimaryTouchpad);

        if (Mathf.Abs(touchCoords.y) > 0.8f)
        {
            bezier.ExtensionFactor = touchCoords.y > 0f ? 1f : -1f;
        }
        else
        {
            bezier.ExtensionFactor = 0f;
        }
    }

    void ToggleTeleportMode()
    {
        teleportEnabled = !teleportEnabled;


        bezier.ToggleDraw(teleportEnabled);
        if (!teleportEnabled)
        {
            teleportSprite.SetActive(false);
            raycastSprite.SetActive(true);
        }
        else
        {
            raycastSprite.SetActive(false);
        }
    }
}
```

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Assets.OVR.Scripts
{
	public class Record
	{
		public string category;
		public string message;
		public Record(string cat, string msg)
		{
			category = cat;
			message = msg;
		}
	}

	public class RangedRecord : Record
	{
		public float value;
		public float min;
		public float max;
		public RangedRecord(string cat, string msg, float val, float minVal, float maxVal)
			: base(cat, msg)
		{
			value = val;
			min = minVal;
			max = maxVal;
		}
	}

	public delegate void FixMethodDelegate(UnityEngine.Object obj, bool isLastInSet, int selectedIndex);

	public class FixRecord : Record
	{
		public FixMethodDelegate fixMethod;
		public UnityEngine.Object targetObject;
		public string[] buttonNames;
		public bool complete;

		public FixRecord(string cat, string msg, FixMethodDelegate fix, UnityEngine.Object target, string[]
buttons)
			: base(cat, msg)
		{
			buttonNames = buttons;
			fixMethod = fix;
			targetObject = target;
			complete = false;
		}
	}
}
```