



GRADO EN INGENIERÍA EN TECNOLOGÍAS INDUSTRIALES

TRABAJO FIN DE GRADO ENTRENADOR DEL PERSONAL DE MANTENIMIENTO DE LA MINIFÁBRICA DE ICAI BASADO EN REALIDAD VIRTUAL

Autor: Borja Rojo González

Director: Jose Antonio Rodríguez Mondejar

Madrid

Julio de 2019

AUTORIZACIÓN PARA LA DIGITALIZACIÓN, DEPÓSITO Y DIVULGACIÓN EN RED DE PROYECTOS FIN DE GRADO, FIN DE MÁSTER, TESIS O MEMORIAS DE BACHILLERATO

1º. Declaración de la autoría y acreditación de la misma.

El autor D. Borja Rojo González

DECLARA ser el titular de los derechos de propiedad intelectual de la obra: ENTRENADOR DEL PERSONAL DE MANTENIMIENTO DE LA MINIFÁBRICA DE ICAI BASADO EN REALIDAD VIRTUAL, que ésta es una obra original, y que ostenta la condición de autor en el sentido que otorga la Ley de Propiedad Intelectual.

2º. Objeto y fines de la cesión.

Con el fin de dar la máxima difusión a la obra citada a través del Repositorio institucional de la Universidad, el autor **CEDE** a la Universidad Pontificia Comillas, de forma gratuita y no exclusiva, por el máximo plazo legal y con ámbito universal, los derechos de digitalización, de archivo, de reproducción, de distribución y de comunicación pública, incluido el derecho de puesta a disposición electrónica, tal y como se describen en la Ley de Propiedad Intelectual. El derecho de transformación se cede a los únicos efectos de lo dispuesto en la letra a) del apartado siguiente.

3º. Condiciones de la cesión y acceso

Sin perjuicio de la titularidad de la obra, que sigue correspondiendo a su autor, la cesión de derechos contemplada en esta licencia habilita para:

- a) Transformarla con el fin de adaptarla a cualquier tecnología que permita incorporarla a internet y hacerla accesible; incorporar metadatos para realizar el registro de la obra e incorporar “marcas de agua” o cualquier otro sistema de seguridad o de protección.
- b) Reproducirla en un soporte digital para su incorporación a una base de datos electrónica, incluyendo el derecho de reproducir y almacenar la obra en servidores, a los efectos de garantizar su seguridad, conservación y preservar el formato.
- c) Comunicarla, por defecto, a través de un archivo institucional abierto, accesible de modo libre y gratuito a través de internet.
- d) Cualquier otra forma de acceso (restringido, embargado, cerrado) deberá solicitarse expresamente y obedecer a causas justificadas.
- e) Asignar por defecto a estos trabajos una licencia Creative Commons.
- f) Asignar por defecto a estos trabajos un HANDLE (URL *persistente*).

4º. Derechos del autor.

El autor, en tanto que titular de una obra tiene derecho a:

- a) Que la Universidad identifique claramente su nombre como autor de la misma
- b) Comunicar y dar publicidad a la obra en la versión que ceda y en otras posteriores a través de cualquier medio.
- c) Solicitar la retirada de la obra del repositorio por causa justificada.
- d) Recibir notificación fehaciente de cualquier reclamación que puedan formular terceras personas en relación con la obra y, en particular, de reclamaciones relativas a los derechos de propiedad intelectual sobre ella.

5º. Deberes del autor.

El autor se compromete a:

- a) Garantizar que el compromiso que adquiere mediante el presente escrito no infringe ningún derecho de terceros, ya sean de propiedad industrial, intelectual o cualquier otro.
- b) Garantizar que el contenido de las obras no atenta contra los derechos al honor, a la intimidad y a la imagen de terceros.
- c) Asumir toda reclamación o responsabilidad, incluyendo las indemnizaciones por daños, que pudieran ejercitarse contra la Universidad por terceros que vieran infringidos sus derechos e

intereses a causa de la cesión.

- d) Asumir la responsabilidad en el caso de que las instituciones fueran condenadas por infracción de derechos derivada de las obras objeto de la cesión.

6º. Fines y funcionamiento del Repositorio Institucional.

La obra se pondrá a disposición de los usuarios para que hagan de ella un uso justo y respetuoso con los derechos del autor, según lo permitido por la legislación aplicable, y con fines de estudio, investigación, o cualquier otro fin lícito. Con dicha finalidad, la Universidad asume los siguientes deberes y se reserva las siguientes facultades:

- La Universidad informará a los usuarios del archivo sobre los usos permitidos, y no garantiza ni asume responsabilidad alguna por otras formas en que los usuarios hagan un uso posterior de las obras no conforme con la legislación vigente. El uso posterior, más allá de la copia privada, requerirá que se cite la fuente y se reconozca la autoría, que no se obtenga beneficio comercial, y que no se realicen obras derivadas.
- La Universidad no revisará el contenido de las obras, que en todo caso permanecerá bajo la responsabilidad exclusiva del autor y no estará obligada a ejercitar acciones legales en nombre del autor en el supuesto de infracciones a derechos de propiedad intelectual derivados del depósito y archivo de las obras. El autor renuncia a cualquier reclamación frente a la Universidad por las formas no ajustadas a la legislación vigente en que los usuarios hagan uso de las obras.
- La Universidad adoptará las medidas necesarias para la preservación de la obra en un futuro.
- La Universidad se reserva la facultad de retirar la obra, previa notificación al autor, en supuestos suficientemente justificados, o en caso de reclamaciones de terceros.

Madrid, a 12 de Julio de 2019

ACEPTA

Fdo.....

Motivos para solicitar el acceso restringido, cerrado o embargado del trabajo en el Repositorio Institucional:

Declaro, bajo mi responsabilidad, que el Proyecto presentado con el título
.....ENTRENADOR DEL PERSONAL DE
MANTENIMIENTO DE LA MINIFÁBRICA DE ICAI BASADO EN REALIDAD
VIRTUAL.....

.....
en la ETS de Ingeniería - ICAI de la Universidad Pontificia Comillas en el
curso académico2018-2019..... es de mi autoría, original e inédito y
no ha sido presentado con anterioridad a otros efectos. El Proyecto no es
plagio de otro, ni total ni parcialmente y la información que ha sido tomada
de otros documentos está debidamente referenciada.

Fdo.: Borja Rojo González

Fecha: 12/ 06/ 2019



Autorizada la entrega del proyecto

EL DIRECTOR DEL PROYECTO

Fdo.: Jose Antonio Rodríguez Mondejar

Fecha: 15 / 07 / 2019



GRADO EN INGENIERÍA EN TECNOLOGÍAS INDUSTRIALES

TRABAJO FIN DE GRADO ENTRENADOR DEL PERSONAL DE MANTENIMIENTO DE LA MINIFÁBRICA DE ICAI BASADO EN REALIDAD VIRTUAL

Autor: Borja Rojo González

Director: Jose Antonio Rodríguez Mondejar

Madrid

Julio de 2019

ENTRENADOR DEL PERSONAL DE MANTENIMIENTO DE LA MINIFÁBRICA DE ICAI BASADO EN REALIDAD VIRTUAL

Autor: Rojo González, Borja

Director: Rodríguez Mondejar, Jose Antonio

Entidad Colaboradora: ICAI – Universidad Pontificia de Comillas

RESUMEN DEL PROYECTO

Introducción

La realidad está formada por aquello que existe de forma auténtica o verdadera. Virtual, por su parte, es lo que cuenta con la virtud de generar un efecto, pero que no se concreta en lo físico. Podría decirse, por tanto, que la idea de realidad virtual enfrenta dos conceptos que resultan opuestos. Sin embargo, la noción es de uso habitual para referirse a aquel entorno informático que representa, de manera digital, algo que simula ser real. Mediante diversos equipos y programas informáticos, la realidad virtual genera una simulación de la realidad [1].

A pesar de que el desarrollo de la RV se ha visto principalmente impulsado por su uso en el área de entretenimiento, la industria está aprovechando esta tecnología para aplicarla a lo que se denomina la cuarta revolución industrial o Industria 4.0. El ámbito de la formación de técnicos especializados en la manipulación de maquinaria industrial es una de las aplicaciones de la tecnología de Realidad Virtual. Igual que es posible generar entornos donde visualizar las máquinas funcionando de manera real, es posible permitir a los usuarios interactuar con ellas para la resolución de averías o incidencias.

Simplemente con unas gafas de realidad virtual es posible colocar a un operario en un entorno virtual donde se visualice una máquina que ha detenido su producción para que este compruebe su funcionamiento y solucione la incidencia tal y como si se la encontrara en el entorno real de fábrica.

Así la Realidad Virtual se sitúa cercana al ámbito del training ofreciendo a las empresas industriales una herramienta mucho más óptima y económica para la formación de sus trabajadores. Pues, gracias a ella, se elimina el coste del técnico que debe hacer la formación a cada trabajador, se optimizan los tiempos y se evita tener que parar la producción para formar a estos trabajadores o permitirles que accedan a las máquinas para practicar su uso.

En este sentido, más idónea incluso, es su aplicación para el entrenamiento en situaciones

de riesgo. Una de las empresas que ya ha usado esta tecnología es 3M que, a través de una simulación virtual, muestra a los trabajadores de la construcción los riesgos de trabajar sin las correctas medidas de seguridad. En este caso, al ahorro de costes antes mencionados, se debe añadir la ventaja de evitar poner en riesgo a los trabajadores de forma innecesaria [2].

El objetivo de este proyecto es desarrollar una aplicación que permita la formación de posibles técnicos de mantenimiento de la minifábrica de ICAI y estudiar las posibles ventajas de la aplicación en caso de ser desarrollada en torno a una planta que todavía no ha sido o está en proceso de ser construida. La aplicación situará al usuario en un entorno virtual que simula el laboratorio de sistemas digitales de ICAI. Una vez dentro,

se expondrán diferentes averías que el usuario tendría que identificar basándose en el funcionamiento de la planta. Con el uso de este tipo de simuladores se pretende:
Formar técnicos: El objetivo principal de la aplicación es ayudar en la formación de técnicos, simulando el entorno en el que trabajarán, sin necesidad de parar la planta y sin que los errores cometidos conlleven consecuencia alguna en la planta. La ventaja de la realidad virtual es que es mucho más inmersiva que un simulador en ordenador, y, por lo tanto, cuando el técnico comience a trabajar en la planta real, ese cambio sería mucho menos drástico, ya que estaría familiarizado con el entorno.

Seguridad: La minifábrica de ICAI consta de varios brazos robóticos. Los motores que controlan estos brazos son muy potentes, y, en caso de avería, un técnico no formado podría verse involucrado en un accidente laboral debido a la falta de experiencia. De este modo, los técnicos novatos pueden realizar las prácticas sin riesgo de lesiones u otros accidentes.

Agilizar la puesta en marcha de la planta: En caso de ser una planta nueva, mediante el uso de la realidad virtual, se puede formar a los operarios mientras que la planta está construyéndose, de modo que cuando esté terminada, los mismos puedan incorporarse inmediatamente, ya que habrían practicado en un entorno virtual, que posee las mismas características y dimensiones que la planta real. Se ahorrarán tiempos muertos, pudiendo comenzar la producción en el menor tiempo.

Resolver problemas de diseño: Al tratar con un espacio virtual, durante la formación de los técnicos se podría detectar con más facilidad fallos en el diseño de las instalaciones, de modo que antes de empezar a construir el complejo, se puedan hacer los cambios pertinentes.

Metodología

Para realizar el proyecto se ha hecho uso de Oculus Rift, Unity y Steam VR. Oculus Rift es un sistema de realidad virtual que permite visualizar espacios generados digitalmente, ofreciendo una sensación inmersiva en el entorno digital. Unity es el motor de desarrollo de videojuegos en el que se ha desarrollado la aplicación. Para poder implementar Oculus Rift en Unity se ha utilizado Steam VR, una aplicación que ofrece un SDK con códigos para facilitar la implementación de la realidad virtual en Unity.

En esta versión de la aplicación se han desarrollado dos tipos de fallos o averías de la planta, el sensor de movimiento y el gatillo. La aplicación sitúa al usuario en el laboratorio de ICAI. Aleatoriamente ocurrirá un fallo en la planta, que el usuario tendrá que identificar y detectar. Se contarán los fallos y los aciertos que el operario haya tenido. Además, para posteriores versiones de la aplicación, se ha desarrollado una plantilla de código C# para facilitar la incorporación de nuevas averías en el programa.

En primer lugar, se desarrolló la ‘escena’. Las ‘escenas’ en Unity son los entornos en los que el usuario va a interactuar con la aplicación, en este caso el laboratorio de sistemas digitales de ICAI. Éste se ha modelado utilizando un modelo de la planta importado anteriormente en Unity. También se ha modelado el laboratorio en sí para favorecer la sensación de inmersión, utilizando modelos importados de sitios web como 3D Warehouse o Unity Asset Store. En la escena se especifican como interactúan los diferentes objetos entre ellos, añadiendo componentes como *colliders*, *scripts* y *animations* entre otros.

Por otro lado, se han desarrollado los scripts que hacen funcionar los diferentes dispositivos y establecen el sistema de comunicaciones entre los objetos. Tanto el sensor de movimiento como el gatillo contienen *scripts* que funcionan de forma independiente. Éstos se comunican a su vez con el gestor de fallos (Failure Manager). El gestor de fallos es el encargado de coordinar todos los elementos. Al inicio de la aplicación, el gestor de fallos inicia una cuenta atrás. Cuando llega a 0, se establece aleatoriamente una avería. El gestor almacena el ID de la avería seleccionada. A continuación, el usuario procede a investigar la incidencia y selecciona el objeto que cree que no funciona correctamente. Una vez seleccionado, el gestor de fallos comprueba si es el objeto correcto o no. En caso negativo, se suma un fallo. En caso afirmativo, se suma un acierto y se reinicia la cuenta atrás.

Resultados y conclusiones

Unity es un motor centrado en la creación de aplicaciones de entretenimiento, y por lo tanto, carece de herramientas que faciliten el desarrollo de aplicaciones más técnicas, como compatibilidad directa con modelos en otros formatos como '.dwg' o '.sldasm'. Aun así, Unity Technologies está trabajando con grandes empresas, especialmente del sector del automóvil para solucionar los problemas.

Con el desarrollo de este proyecto, se demuestra, que, a pesar de las carencias técnicas de Unity, se pueden crear aplicaciones de simulación industriales, especialmente en el área de formación de personal. El uso de simuladores en cualquier entorno facilita el aprendizaje y la formación. En el caso de una gran fábrica industrial, el mantenimiento de la planta es de especial importancia, especialmente si los beneficios se cuentan por horas. Para formar correctamente a los técnicos haría falta parar la producción o esperar a que se produjese en una avería, lo que se traduce en importantes pérdidas económicas. No solo eso, sino que también, según el tipo de planta, se podría comprometer la seguridad de los técnicos. El uso de simuladores de este tipo permite al personal llegar a la planta real con la formación necesaria para afrontar el problema.

La gran ventaja de la realidad virtual frente a los simuladores convencionales es que no hace falta una réplica física de la máquina/dispositivo que se vaya a reparar. Se puede generar una réplica virtual que funcione de la misma manera que lo haría la real, a partir de los planos de este. De este modo hay un ahorro en el coste de producción del equipo, con la ventaja de que el entorno virtual se puede actualizar, y en caso de que se cambie algún componente de la planta de la fábrica, se podría hacer en el entorno virtual de la misma manera.

Algunas empresas, como PixoGroup, han desarrollado aplicaciones de formación, que permiten el uso simultáneo de la aplicación de varios operarios a la vez, de modo que se puedan entrenar dinámicas en las que se requiera más de una persona, desarrollando un método de formación que ayuda a los alumnos a retener mejor lo que aprenden durante la formación.

En conclusión, los programas de entrenamiento en Realidad Virtual son una realidad presente que, aunque todavía no se haya establecido completamente, está experimentando

un gran crecimiento. Grandes empresas industriales están invirtiendo en este tipo de aplicaciones, convirtiéndose la Realidad Virtual en uno de los principales impulsores hacia la implementación de la Industria 4.0.

El futuro de este tipo de aplicaciones dependería de cómo se desarrolle la Realidad

Virtual. En el caso de programas de formación, se podrían generar sistemas de enseñanza que formasen y examinasen a los operarios, mediante un sistema de tutoriales y lecciones virtuales. En cuanto a aplicaciones directamente relacionadas con el mantenimiento, sería de especial interés ser capaces de conectar directamente la planta en sí con la aplicación, de modo que los movimientos y estado de la planta física se reflejen en la virtual, así el operario podría diagnosticar los problemas reales de la planta sin necesidad de estar, ni si quiera, en el mismo edificio; o poder controlar la planta desde la realidad virtual, minimizando el contacto físico entre operario y maquinaria. Otro de los posibles desarrollos es el diseño de las plantas, mediante la creación de modelos a partir de planos de CAD, poder visualizar y testear una planta de manera virtual, para comprobar su funcionamiento y distribución. Ésto sería posible mediante la creación, en caso de Unity, de modelos 'prefabs' que se puedan reciclar para ser utilizados en diferentes proyectos, ahorrando costes de diseño y programación. Por otro lado, alejado de la industria manufacturera, se podrían diseñar aplicaciones que simulasen incidentes en centros donde los accidentes son inusuales pero críticos, como el caso de las centrales nucleares. En estas centrales es difícil entrenar operarios de manera realista, ya que las incidencias son mínimas, pero a la vez son muy peligrosas. Por esto, una formación previa en un simulador de realidad virtual podría, en casos extremos, prevenir catástrofes.

Referencias

- [1] Julián Pérez Porto y María Merino. Definición de realidad virtual. 2013. Recuperado el 1 de Julio de 2019. url: definicion.de/realidad-virtual.
- [2] Amparo Caballero. Industria 4.0 a través de Realidad Virtual y Realidad Aumentada. 2017. Recuperado el 1 de Julio de 2019. url: <http://www.innoarea.com/industria-4-0-a-traves-de-realidad-virtual-y-realidad-aumentada/>.

VIRTUAL REALITY BASED MAINTENANCE TECHNICIAN TRAINER FOR ICAI'S MINIFACTORY.

Author: Rojo González, Borja

Director: Rodríguez Mondejar, Jose Antonio

Collaborating entity: ICAI – Universidad Pontificia de Comillas

PROJECT SUMMARY

Introduction

Reality is formed by all that is real or existent. Virtual, on the other hand, is what has the virtue of generating an effect that does not materialize in the physical world. It could be said, that the idea of virtual reality faces two opposite concepts. However, the term is often used to refer to something that simulates reality. By using different gear and programs, virtual reality generates a simulation of the real world [1].

Even though VR has been developed thanks to its use in the entertainment industry, the technical industry is taking advantage of it by applying it to what it is called the 4th industrial revolution or Smart Industry. The training of specialized technicians in the handling of industrial machinery is one of the applications of virtual reality in the industry. In the same way that it is possible to generate virtual industry environments where working machines can be visualized, it is possible to allow users to interact with those machines to learn how to solve faults or failures. With a VR Headset it is possible to place the technician in a digital environment where he visualizes the failing machine that has stopped working, so he can check it and solve the problem as if he was in the real factory or plant.

This way, Virtual Reality is a great application in the training ambit, offering industrial companies a much more optimal and economical tool to train their employees. The cost of the trainer is eliminated; production times are optimized; and the stop of the production to train the technicians and allow them access to the machines is avoided. In this sense, the use of VR in occupational hazards is even better. 3M is one of the companies, that using VR, shows construction workers the risks of working without taking safety measures. In this case, to the cost savings advantages previously mentioned, the advantage of avoiding risking the workers unnecessarily must be added to the list [2].

The main purpose of this project is to create an application that allows the training of possible maintenance technicians at ICAI's minifactory and study the advantages of the application, in case of being developed for a factory that has not yet or is being built. The application will place the user in a virtual environment that replicates the digital systems laboratory of ICAI. Once inside, the technician will be exposed to different faults, that he will have to identify. With the use of this type of simulators, the intention is to:

Train technicians: The purpose of the application is to develop a training program for technicians that simulates the plant/factory where they will work, without the need of pausing the production, and without the risk that errors would entail in the plan. The advantage of VR is that it is much more immersive than traditional simulators, so when

the technician starts working in the factory, the change would be less harsh, as he would be already familiarized with the environment.

Safety: ICAI's minifactory is equipped with four robotic arms. The motors that move these robots are very powerful and, in case of fault, a non-formed technician could be injured due to lack of experience. This way, unformed technicians can practice without any kind of risk.

Speed up the start-up of the plan: By using VR, employees can be trained while the factory is being built, so when it is finished, the technicians will be able to work immediately.

Solve design problems: Since the plan has been modelled in 3D, it will be easier to detect problems in the design of the plant, so that before building the facility, this problem can be solved.

Methodology

In order to carry out this project Oculus Rift, Unity and Steam VR had been used as the main tools of development. Oculus Rift is a headset that allows the use of virtual reality in digital environments. Unity is the game engine where the application has been developed. Finally, Steam VR allows VR to be implemented in Unity, by the use of an SDK.

In this version of the application, two types of faults have been developed. The movement sensor and the stopping device. The application places the user in the digital systems laboratory at ICAI. Randomly the different devices will start failing, the user will have to detect and identify the source of the fault. Hits and misses will be scored. Additionally, a script template has been developed, in order to ease the implementation of faults in the future.

On the one hand, the 'scene' was created. 'Scenes' in Unity are the environments where the user interacts with the application, in this case, the digital systems laboratory of ICAI. It has been modelled using the model of the plant imported previously in Unity. The laboratory has also been modelled using models imported from websites such as 3D Warehouse or Unity Asset Store. In the 'scene' interactions between objects are coordinated by the addition of components such as colliders, scripts or animations.

On the other hand, scripts that make the different devices work and objects communicate between them had been developed. Both the movement sensor and the stopping device contain scripts that work independently. They communicate with the 'Failure Manager'. The Failure Manager coordinates all the objects in the scene. At the beginning of the application, it starts a countdown. Once it reaches 0, a fault is established randomly. The manager stored the fault's ID. Then, the user investigates the fault and selects the object that he believes is failing. Once selected, the failure manager checks if it is the correct one. If so, it adds a hit and restarts the countdown, else, it will add a miss.

Results and conclusions

Unity is an engine mainly used in the creation of entertainment applications, and therefore, it lacks many of the tools that would ease the development of more technical applications, such as the compatibility with formats such as '.dwg' or '.sldasm'. Even

so, Unity Technologies is working with big companies, especially from the automotive industry to solve these problems.

This project has demonstrated that, despite the lack of technical tools in Unity, industry simulation applications can be created, especially in the training ambit. The use of simulators eases training and learning. In the case of a big industrial factory, the plant maintenance is very important, especially when the benefits are counted by hours. To properly train technicians the pause of the production is needed. This translates in economic losses. Not only that, but the safety of the employees is compromised. The use of VR simulators allows staff to start working with enough knowledge to face any problem.

The great advantage of VR over traditional simulators is that there is no need to create a replica of the machine. A virtual replica can be generated based on the plans. This way, there is a cost saving in the production of the simulator, and in case of any update in the machinery, it could be done the same way.

Some companies, such as PixoGroup, have developed training programs that allow the simultaneous use by different users of the application, allowing trainees to work together.

In conclusion, VR training programs are a present reality, even though this technology is not completely established, it is experiencing a great growth. Big industrial companies are investing in this type of applications. Becoming Virtual Reality one of the boosters of the Smart Industry.

References

- [1] Julián Pérez Porto y María Merino. Definición de realidad virtual. 2013. Retrieved July 1, 2019. url: definicion.de/realidad-virtual.
- [2] Amparo Caballero. Industria 4.0 a través de Realidad Virtual y Realidad Aumentada. 2017. Retrieved July 1, 2019. url: <http://www.innoarea.com/industria-4-0-a-traves-de-realidad-virtual-y-realidad-aumentada/>.

Índice general

I	Memoria	17
1.	Introducción y estado del arte	19
1.1.	Historia	19
1.2.	Funcionamiento	21
1.3.	Uso de la realidad virtual en la industria	21
2.	Objetivos del proyecto	25
3.	Recursos	27
3.1.	Unity	27
3.2.	Steam VR	28
3.3.	Oculus Rift	28
4.	Diseño de la escena	31
4.1.	Planta	32
4.1.1.	Robot ABB IRB 120	34
4.1.2.	Sensor movimiento y gatillo	35
4.2.	Laboratorio	37
4.2.1.	Luces	39
4.2.2.	Interfaz de contadores	40
5.	Diseño del código	41
5.1.	Gestor de fallos	41
5.2.	Dispositivos susceptibles de fallo	43
5.2.1.	Sensor de movimiento	43
5.2.2.	Gatillo	43
5.3.	Objetos pulsables activos y pasivos	44
5.4.	Movimiento de las cintas	46
5.5.	Añadir nuevos fallos	48
6.	Implementación en RV	51
6.1.	Player	51
6.2.	Controles	52

7. Puesta en marcha de Oculus Rift	55
8. Conclusiones y desarrollos futuros	59
II Presupuesto	61
9. Presupuesto del proyecto	63
9.1. Ingeniería	63
9.2. Hardware y software	63
9.3. Total	64
III Código fuente	65
9.4. Código de Unity	67
9.4.1. Belt.cs	67
9.4.2. CircularBelt.cs	68
9.4.3. FailureManager.cs	69
9.4.4. MovementSensor.cs	72
9.4.5. Pallet.cs	74
9.4.6. Plantilla Fallo.cs	75
9.4.7. SDSwitch.cs	76
9.4.8. StoppingDevice.cs	77
9.4.9. ToggleLight.cs	80
9.4.10. Button.cs	80
9.5. Código de Steam VR	84
9.5.1. BodyCollider.cs	84
9.5.2. ControllerButtonHints.cs	85
9.5.3. ControllerHoverHighlight.cs	103
9.5.4. Hand.cs	105
9.5.5. IgnoreTeleportTrace.cs	146
9.5.6. InputModule.cs	146
9.5.7. Interactable.cs	148
9.5.8. Player.cs	156
9.5.9. SteamVR_ ActivateActionSetOnLoad.cs	165
9.5.10. SteamVR_ Behaviour.cs	166
9.5.11. SteamVR_ Behaviour_ Pose.cs	170
9.5.12. SteamVR_ Fade.cs	177
9.5.13. SteamVR_ Overlay.cs	180
9.5.14. SteamVR_ Render.cs	184
9.5.15. Teleport.cs	193
9.5.16. TeleportArc.cs	216
9.5.17. TeleportArea	223
9.5.18. TeleportMarkerBase.cs	227

<i>ÍNDICE GENERAL</i>	15
9.5.19. TeleportPoint.cs	228
9.5.20. UIElement.cs	235
A. Assets importados	239
B. Imágenes adicionales de la aplicación	241

Parte I

Memoria

Capítulo 1

Introducción y estado del arte

La realidad está formada por aquello que existe de forma auténtica o verdadera. Virtual, por su parte, es lo que cuenta con la virtud de generar un efecto, pero que no se concreta en lo físico. Podría decirse, por tanto, que la idea de realidad virtual enfrenta dos conceptos que resultan opuestos. Sin embargo, la noción es de uso habitual para referirse a aquel entorno informático que representa, de manera digital, algo que simula ser real. Mediante diversos equipos y programas informáticos, la realidad virtual genera una simulación de la realidad. Para ello, se hace uso de un casco con un visor que proyecta las imágenes, además de contar con diversos sensores para poder interactuar con el entorno[15].

En la actualidad, esta tecnología está siendo desarrollada en diversos campos, como pueda ser la educación, entretenimiento, software, videojuegos o medicina. En la industria se están desarrollando numerosas aplicaciones tanto de realidad virtual como aumentada, especialmente en el área de simulación y entrenamiento, con objetivos tales como: formación, resolución de problemas o mejorar las medidas de seguridad. Con el desarrollo de la industria 4.0, el consumo de realidad virtual aplicada a la industria es actualmente del 40 % [19]. Es por ello que las empresas deben actualizarse y adaptarse a este tipo de tecnologías que están cobrando cada año más importancia.

En este trabajo de fin de grado se pretende crear una aplicación que permita formar a técnicos de mantenimiento sin necesidad de interactuar físicamente con la planta. La aplicación se centrará en la planta de la minifábrica de ICAI. Ésta consta de una pequeña línea de montaje con varios brazos robóticos. La aplicación situará al técnico en la planta, donde se expondrá a diferentes fallos comunes, que tendrá que identificar y señalar.

1.1. Historia

Los verdaderos orígenes de la realidad virtual se remontan dos siglos atrás, concretamente al año 1844, cuando Charles Wheatstone creó el estereoscopio, un medio que consiste en obtener dos fotografías prácticamente idénticas, cuya única diferencia radica en el punto de toma de la imagen, lo que provoca que sean observadas por cada

ojo de forma independiente, lo que hace que el cerebro las mezcle en una sola creando un objeto tridimensional. Esta técnica sería la base de los primeros visores de realidad virtual, e incluso a día de hoy siguen utilizándose dichos patrones.



Figura 1.1: Estereoscopio

Fuente: caminosantiago360.com[6]

Los verdaderos primeros pasos llegaron en el año 1958. La empresa *Philco Corporation* ideó un sistema de realidad virtual que fue diseñado para generar entornos artificiales a los que podían acceder personas mediante un casco que captaba los movimientos que los usuarios realizaban con sus cabezas. Con el paso de los años se crearon dispositivos más eficientes, e incluso la NASA apoyó a varias empresas en el desarrollo de diversos dispositivos con el objetivo de crear ambientes interactivos con los que se pudiera captar el movimiento de cualquier parte del cuerpo, incorporando además sustanciosas mejoras visuales, como entornos tridimensionales y estereoscópicos. El primer casco verdaderamente VR llegaría en 1961, construido por Corneau y Bryan, empleados de *Philco Corporation*.

El concepto en sí de 'Realidad Virtual' nace en el año 1965, cuando Ivan Sutherland, considerado como uno de los pioneros de la informática, publicó un artículo llamado *The Ultimate Display*, en el que explicaba el concepto que los científicos llevaban años tratando de asentar. El propio Sutherland desarrollaría dos años después el primer programa diseñado para crear mundos virtuales con imágenes en 3D[13].

En 2007 Google introdujo lo que se podría considerar como el inicio de realidad virtual moderna, con *Google Street View*, un servicio que nos muestra vistas panorámicas de gran parte de las carreteras de nuestro planeta. En 2010, Palmer Luckey diseñó el primer prototipo de Oculus Rift, que es en la actualidad uno de los sistemas de RV de referencia.

Actualmente otras grandes empresas tecnológicas como Facebook, Apple, Amazon, Microsoft, Sony o Samsung han abierto departamentos centrados en realidad virtual y aumentada. En 2016 la empresa HTC lanzó al mercado el casco HTC VIVE, que a día de hoy está compitiendo por el liderazgo del sector con Oculus Rift[4].

1.2. Funcionamiento

Para hacer uso de la realidad virtual se requieren de dos elementos fundamentales: un equipo informático, y unas lentes o casco. El equipo puede ser un ordenador, un móvil o una tablet. Por otro lado, las lentes VR son por lo general productos diseñados por empresas, que pueden ir desde un simple dispositivo construido en cartón, hasta cascos completamente independientes, como puede ser el recientemente lanzado Oculus Go, que incluye el equipo informático y es completamente portátil. Las gafas/casco de realidad virtual más utilizadas actualmente son Oculus Rift y HTC Vive. Las características principales por las que se diferencian los diferentes sistemas de RV son:

- **Resolución:** La cantidad de píxeles que tiene la pantalla. En realidad virtual es de especial importancia, puesto que los ojos están muy cerca de la pantalla, por tanto, a mayor cantidad de píxeles, mayor será la calidad de la imagen.
- **Tasa de refresco:** Mide la fluidez de las imágenes en pantalla. Es la cantidad de veces que se actualiza la imagen en el tiempo. Es también importante, ya que al controlar los movimientos con la cabeza, éstos son a menudo bruscos y rápidos, por lo que una tasa de refresco baja podría producir mareos.
- **Ángulo de visión:** Campo de visión que ofrecen las gafas.
- **Sensores:** Son necesarios para registrar los movimientos, conocer la posición del usuario y poder interactuar con el entorno. Los hay internos (incorporados en el casco) y externos (normalmente cámaras).
- **Área de rastreo:** Superficie dentro de la cual los movimientos son registrados por los sensores[20].

1.3. Uso de la realidad virtual en la industria

A pesar de que el desarrollo de la RV se ha visto principalmente impulsado por su uso en el área de entretenimiento, la industria está aprovechando esta tecnología para aplicarla a lo que se denomina la cuarta revolución industrial o Industria 4.0.

Hay sectores industriales donde el prototipado de productos requiere de altos costes de inversión; es necesario crear físicamente los productos para que se puedan apreciar sus características y analizar pormenorizadamente su diseño.

Un ejemplo de industria que requiere de estos procesos de alto coste es la automovilística. Hasta el momento era necesario construir, aunque con calidades inferiores, el modelo que luego se iba a comercializar para analizar su uso real.

La Realidad Virtual es la herramienta perfecta para reducir los costes derivados de las tareas de prototipado. Esta tecnología permite crear una simulación casi real del futuro producto donde se visualizarán todas las características del mismo como si se tuviera

físicamente delante de los ojos. Esta creación virtual, además, permite probar diferentes opciones de acabado sin inversión en prototipado.

BMW ha sido una de las primeras marcas en hacer uso de la Realidad Virtual para visualizar, entre otros aspectos, nuevos diseños de interiores de los vehículos, así como otras características físicas de sus coches. Estas simulaciones sirven para comprobar aspectos como la visibilidad interior del coche o si el tablero de instrumentos y controles se ve adecuadamente desde la posición del conductor. Esta tecnología ha permitido a la compañía alemana tomar decisiones sin tener que recurrir a los costosos túneles de simulación utilizados anteriormente para el prototipado.



Figura 1.2: Aplicación de realidad virtual de BMW

Fuente: zerolight.com[10]

El ámbito de la formación de técnicos especializados en la manipulación de maquinaria industrial es otra de las aplicaciones de la tecnología de Realidad Virtual. Igual que es posible generar entornos donde visualizar las máquinas funcionando de manera real, es posible permitir a los usuarios interactuar con ellas para la resolución de averías o incidencias.

Simplemente con unas gafas de realidad virtual es posible colocar a un operario en un entorno virtual donde se visualice una máquina que ha detenido su producción para que este compruebe su funcionamiento y solucione la incidencia tal y como si se la encontrara en el entorno real de fábrica.

Así la Realidad Virtual se sitúa cercana al ámbito del *training* ofreciendo a las empresas industriales una herramienta mucho más óptima y económica para la formación de sus trabajadores. Pues, gracias a ella, se elimina el coste del técnico que debe hacer la formación a cada trabajador, se optimizan los tiempos y se evita tener que parar la producción para formar a estos trabajadores o permitirles que accedan a las máquinas para practicar su uso.

En este sentido, más idónea incluso, es su aplicación para el entrenamiento en situacio-

nes de riesgo. Una de las empresas que ya ha usado esta tecnología es 3M que, a través de una simulación virtual, muestra a los trabajadores de la construcción los riesgos de trabajar sin las correctas medidas de seguridad. En este caso, al ahorro de costes antes mencionados, se debe añadir la ventaja de evitar poner en riesgo a los trabajadores de forma innecesaria[3].



Figura 1.3: Aplicación de realidad virtual para seguridad de 3M

Fuente: 3m.com[21]

Capítulo 2

Objetivos del proyecto

El objetivo de este proyecto es desarrollar una aplicación que permita la formación de posibles técnicos de mantenimiento de la minifábrica de ICAI y estudiar las posibles ventajas de la aplicación en caso de ser desarrollada en torno a una planta que todavía no ha sido o está en proceso de ser construida. La aplicación situará al usuario en un entorno virtual que simula el laboratorio de sistemas digitales de ICAI. Una vez dentro, se expondrán diferentes averías que el usuario tendrá que identificar basándose en el funcionamiento de la planta. Con el uso de este tipo de simuladores se pretende:

- **Formar técnicos:** El objetivo principal de la aplicación es ayudar en la formación de técnicos, simulando el entorno en el que trabajarán, sin necesidad de parar la planta y sin que los errores cometidos conlleven consecuencia alguna en la planta. La ventaja de la realidad virtual es que es mucho más inmersiva que un simulador en ordenador, y, por lo tanto, cuando el técnico comience a trabajar en la planta real, ese cambio será mucho menos drástico, ya que estará familiarizado con el entorno.
- **Seguridad:** La minifábrica de ICAI consta de varios brazos robóticos. Los motores que controlan estos brazos son muy potentes, y, en caso de avería, un técnico no formado podría verse involucrado en un accidente laboral debido a la falta de experiencia. De este modo, los técnicos novatos pueden realizar las prácticas sin riesgo de lesiones u otros accidentes.
- **Agilizar la puesta en marcha de la planta:** En caso de ser una planta nueva, mediante el uso de la realidad virtual, se puede formar a los operarios mientras que la planta está construyéndose, de modo que cuando esté terminada, los mismos puedan incorporarse inmediatamente, ya que habrán practicado en un entorno virtual, que posee las mismas características y dimensiones que la planta real. Se ahorrarán tiempos muertos, pudiendo comenzar la producción en el menor tiempo.
- **Resolver problemas de diseño:** Al tratar con un espacio virtual, durante la formación de los técnicos se podrá detectar con más facilidad fallos en el diseño de las instalaciones, de modo que antes de empezar a construir el complejo, se puedan hacer los cambios pertinentes.

Capítulo 3

Recursos

Para crear una aplicación de este tipo, se necesitan principalmente dos elementos: el dispositivo de realidad virtual y un motor gráfico. El motor gráfico es el encargado de dar forma a la aplicación y el proyecto en sí, en este caso Unity. Por otro lado, como visor de realidad virtual, se ha escogido Oculus Rift, con el que se podrá visualizar la aplicación como si se tratase de un entorno físico. Para poder coordinar la aplicación y el visor, se ha hecho uso de Steam VR, una aplicación que permite la inclusión de realidad virtual en Unity, y que ofrece diversas herramientas para facilitar la programación de RV en el motor de desarrollo.

3.1. Unity

Unity es un motor de desarrollo para la creación de juegos y contenido 3D interactivo con las características que es completamente integrado y que ofrece innumerables funcionalidades para facilitar el desarrollo de este tipo de aplicaciones. Perteneció a la empresa Unity Technologies, fundada en 2004 en Copenhague, Dinamarca.

Actualmente, es la herramienta más utilizada por los creadores de experiencias XR (Realidad aumentada y virtual). Utilizada por estudios de juegos, agencias creativas como Weiden+Kennedy, la NASA e incluso Google. Es por ello que se ha optado por este software frente a otros similares como Unreal Engine. Otra de las ventajas es la comunidad que ha desarrollado, ofreciendo grandes cantidades de información en foros propios y externos.

Unity se puede usar de forma gratuita en el entorno educativo. El precio del software para uso profesional es de 125\$/mes. Posee un plan de suscripción de 25\$/mes para aficionados, siendo por esto la preferida de los desarrolladores independientes[17].



Figura 3.1: Logo de Unity

Fuente: Unity Technologies[17]

3.2. Steam VR

Steam VR es una herramienta que permite ejecutar aplicaciones en realidad virtual. Pertenece a la compañía Valve que comenzó su desarrollo en 2014. Comenzó como un modo experimental que soportaba y utilizaba Oculus Rift.

Actualmente es una de las principales plataformas para ejecutar aplicaciones en la realidad virtual, ofreciendo una configuración rápida. Además, posee un SDK de Unity para la creación de experiencias VR. Este kit de desarrollo facilita código, herramientas y modelos de Unity para poder incorporar fácilmente el equipo de VR en la aplicación[16].



Figura 3.2: Logo de Steam VR

Fuente: Valve[16]

3.3. Oculus Rift

El dispositivo de VR utilizado es Oculus Rift. Desarrollado por la empresa Oculus, actualmente pertenece a Facebook. Empezó como un proyecto en *Kickstarter.com* que consiguió recaudar 2.5 millones de dólares[12]. El equipo que se utiliza en este proyecto es la primera versión lanzada dirigida al consumidor (Oculus Rift CV1). Actualmente la

compañía ha lanzado dos productos nuevos, el primero, una actualización del Oculus Rift tradicional, y el segundo, Oculus Go, que no requiere de un ordenador. Oculus Rift CV1 se compone de dos pantallas OLED de 1080x1200 cada una (2160x1200 en total), con un ángulo de visión de 110°. El casco incluye unos auriculares. En la caja, además, se incluyen dos mandos (uno para cada mano) y dos sensores. Para poder ejecutarlo se recomienda un ordenador con:

Windows 10

Tarjeta gráfica Nvidia GTX1060/AMD Radeon R9 290 o superior

CPU Intel i5-4590 equivalente o superior

Memoria 8Gb RAM

Salida de vídeo HDMI 1.3

Puertos USB 3x USB 3.0 y 1x USB 2.0[11]



Figura 3.3: Logo de Oculus

Fuente: Oculus[11]

Capítulo 4

Diseño de la escena

En Unity se denomina 'escena' al espacio con el que el usuario va a interactuar. Éste solo puede estar en una escena al mismo tiempo, pudiendo moverse entre diferentes escenas dentro de la misma aplicación. Para este caso sólo se ha hecho uso de una escena, el laboratorio. En la escena se colocarán todos los objetos y componentes que se vayan a ejecutar, y se establecerán las relaciones entre los mismos. En primer lugar, se genera el espacio, al que se le añaden los modelos 3D del laboratorio. A continuación, se les asigna a cada modelo los componentes necesarios y se establecen las relaciones entre los mismos, ya sea mediante código u otros elementos de Unity.

El primer paso para empezar a trabajar en Unity es generar la escena 'física' en la que se va a desarrollar la escena. En este caso, el escenario es el Laboratorio de Sistemas Digitales. Para importar modelos, en primer lugar, hay que arrastrar los objetos a la ventana de la escena. Una vez arrastrado el modelo a la escena, este se convierte en un *Game Object* (Objeto de juego). Los *Game Objects* son instancias que pueden contener diferentes componentes. Dependiendo de los componentes que se le agreguen se comportará de una forma u otra. Estos componentes se pueden ver en la ventana 'inspector' a la derecha. Los *Game Objects* importados poseen todos el componente *Transform*, que indica la posición y la escala del objeto en la escena. Además, se pueden añadir otros componentes como *colliders* (permite interactuar físicamente con otros *Game Objects*), animaciones, scripts, materiales y *renderers* entre otros. (ver Figura 4.1).

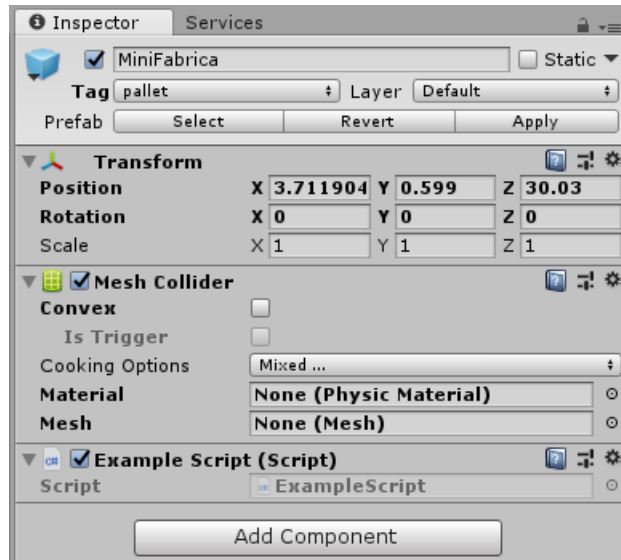


Figura 4.1: Ventana del inspector de Unity.

En este capítulo se ofrece una descripción más detallada de los modelos principales. Otros modelos como las mesas, setas de emergencia y puertas no aparecen desarrollados, ya que solo funcionan como objetos que ayudan a la inmersión y no poseen función específica dentro del programa, conteniendo su instancia de *Game Object* los componentes básicos para que aparezcan en escena.

4.1. Planta

El diseño y modelado de la planta se ha extraído del trabajo fin de máster de Carlos Álvarez Vereterra[7], que importó el modelo a su vez de otros trabajos fin de grado y máster. El modelo original estaba creado en AutoCAD, éste utiliza un formato de archivo con la extensión '.dwg' (DrawWinG, o plano) para almacenar sus archivos 3D. Este formato desarrollado por Autodesk no es compatible con Unity, como se puede comprobar en la documentación oficial. Esto implica que no se puede soportar directamente, sino que será necesario convertirlo a otra extensión. Entre los formatos compatibles se encuentra '.fbx' un formato desarrollado también por Autodesk. AutoCAD permite exportar directamente a este formato, así que esta fue la primera opción. Sin embargo, el modelo en '.fbx' visto a través de la realidad virtual genera sensación de mareo, impidiendo que se pueda usar el sistema de RV por más de unos segundos. Visto el problema de rendimiento, es necesario buscar una alternativa.

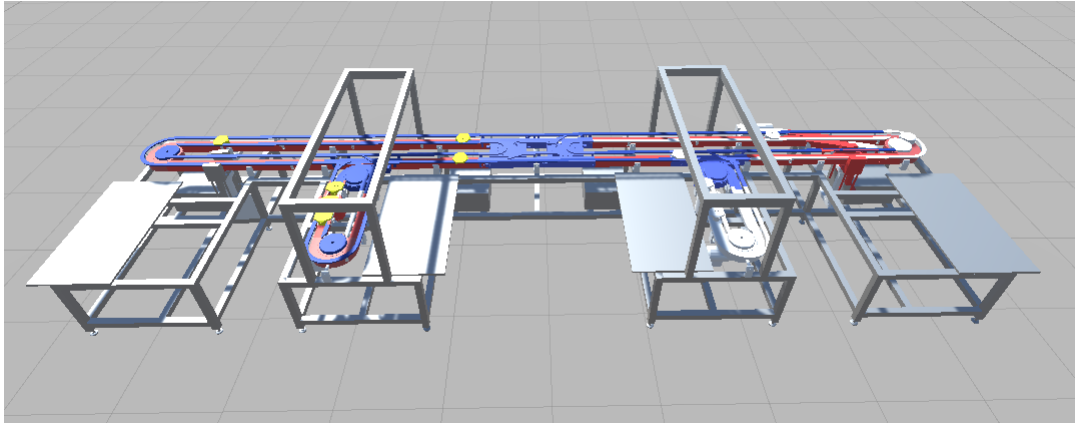


Figura 4.2: Modelo de la planta en Unity.

Este problema de rendimiento se debe a que AutoCAD no es un software de modelado 3D para videojuegos, sino un software CAD para aplicaciones de diseño industrial donde la precisión y la fidelidad son cruciales, sobre todo en campos como el prototipado rápido o impresión 3D.

Una alternativa de Autodesk, la misma compañía que desarrolla AutoCAD, es utilizar 3DS Max, un programa especializado en modelado y animación para videojuegos.

3DS Max no es un programa gratuito, sino que tiene un modelo de negocio de suscripción. Sin embargo, para esta aplicación se ha utilizado la versión de prueba de 30 días. Al estar desarrollado por Autodesk es compatible con archivos '.dwg', por lo que no será necesario realizar ninguna conversión para abrirlo[8].

Para este proyecto no se han realizado más modificaciones más allá de los materiales de los modelos, para los cuales se han utilizado colores básicos para poder diferenciar las diferentes partes de la planta. A continuación, se ofrece una breve descripción de los componentes de la planta:

- **Transformar:** Indica posición y escala de la planta en la escena. En este caso posición (4.42,0.0750,-3.352) y escala (1,1,1). La escala se ha dejado intacta, porque la planta estaba diseñada con las medidas reales, sirviendo de referencia para ajustar la escala de los demás objetos de la escena.
- **Scripts:** En este caso tres scripts: *UIElement*, *Interactable*, *Button*.

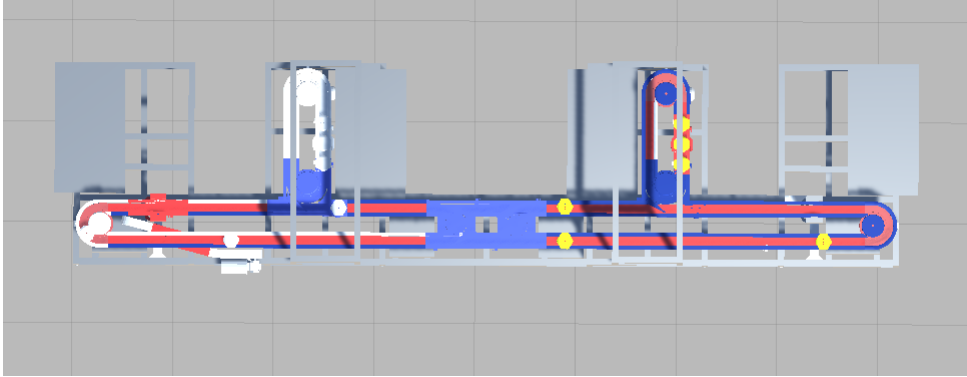


Figura 4.3: Vista en planta del modelo de la planta en Unity.

4.1.1. Robot ABB IRB 120

Al igual que la planta, el modelo del robot ha sido importado del trabajo fin de máster de Carlos Álvarez Vereterra[7]. La importación del modelo 3D del robot ABB IRB 120 es bastante sencilla. ABB proporciona el modelo 3D en su página web[2], en varios formatos. Utilizando el formato *Google Sketchup*, Unity es capaz de reconocer el modelo 3D directamente, sin necesidad de convertirlo a otro formato. El único problema encontrado con ese modelo es la escala. En la ventada 2D del editor de Unity, es muy complicado tener una sensación de escala realista. Al visualizar el modelo en 3D utilizando el Oculus Rift por primera vez, éste medía más de 100 metros de alto. En las opciones de importación del modelo de Unity, es necesario hacer la conversión de pulgadas a metros, haciendo los siguientes cambios:

- Unit conversion: 1 Meters = 0,03937008.
- Scale factor: 0; 001.
- Use file Scale: activado. Una vez hechas estas correcciones, el modelo ya estaría a escala 1:1, lo cual es fácil de comprobar utilizando un HMD de realidad virtual como el Oculus Rift[9].



Figura 4.4: Modelo robot ABB IRB 120

4.1.2. Sensor movimiento y gatillo

Para este proyecto se modelaron dos objetos específicos, el sensor de movimiento y el gatillo, que se encuentran en la planta real (Ver 4.5) El primero se encarga de detectar cuándo un pallet se encuentra cerca del mismo, y el segundo es un dispositivo que al activarse bloquea el movimiento del pallet. Estos objetos han sido modelados con formas primitivas de Unity.

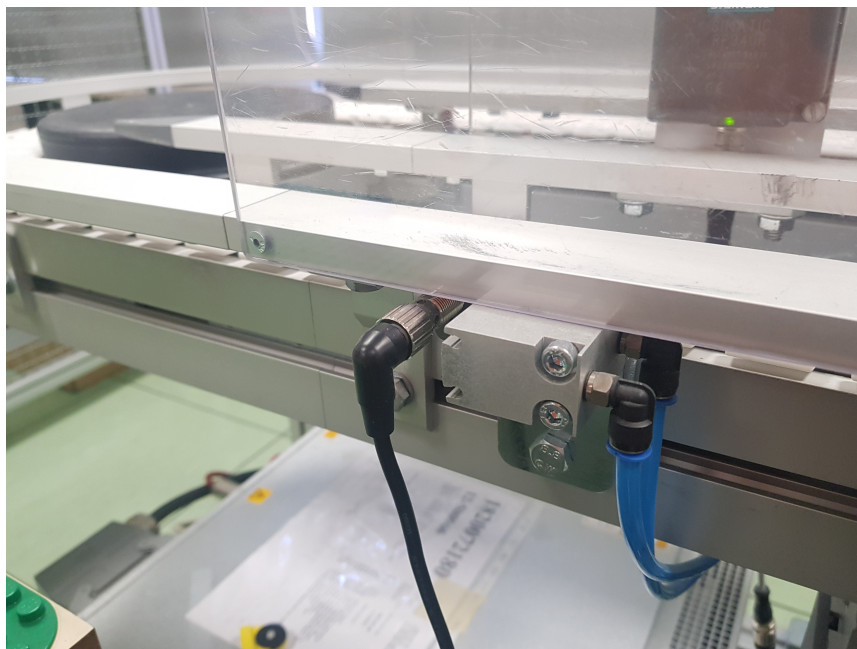


Figura 4.5: Sensor de movimiento (izquierda) y gatillo (derecha).

El sensor de movimiento está formado por una cápsula (cuerpo del sensor), una esfera (dispositivo del LED), una luz de punto (luz del LED) y un objeto sin render(invisible), que es el que se encarga de detectar si un pallet ha pasado por el punto especificado. Ver Figura 4.6.

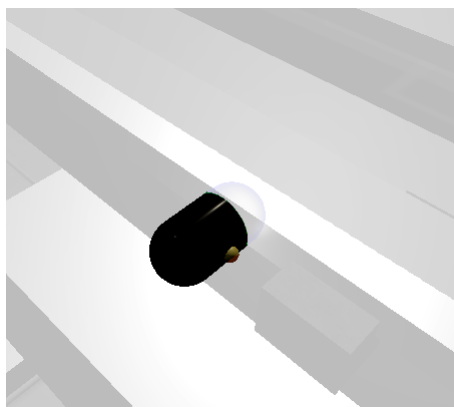


Figura 4.6: Modelo en Unity del sensor de movimiento.

El gatillo se ha modelado con dos cubos. Uno que hace de la parte visible del dispositivo, y otro que hace de gatillo para bloquear el pallet. Al activarse, el gatillo se desplaza, de modo que el componente *Collider* del pallet, choca contra el *Collider* del

gatillo, impidiendo que pueda seguir desplazándose. Ver Figura 4.7

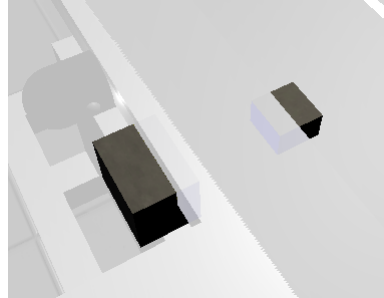


Figura 4.7: Modelo en Unity del gatillo.

Además, se ha añadido un modelo de las válvulas de aire situadas en la parte inferior, con un LED que se ilumina indicando qué válvula está funcionando, y cuando está funcionando. Ver Figura 4.8.

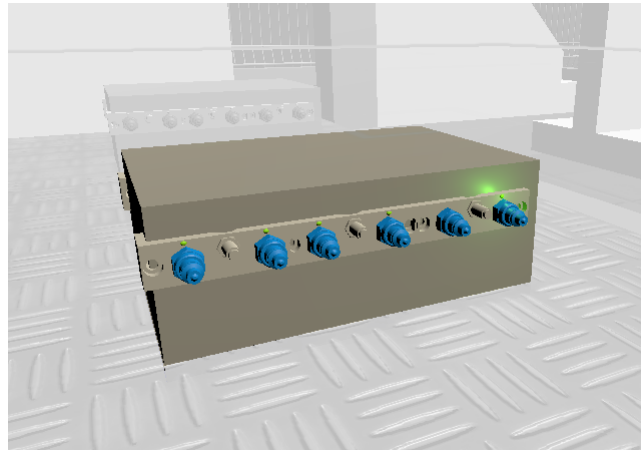


Figura 4.8: Modelo de las válvulas con LED encendido.

4.2. Laboratorio

El laboratorio se ha intentado modelar de la manera más fiel a la realidad. Para ello se tomaron medidas aproximadas de las dimensiones del laboratorio, y se ajustaron según la escala de la planta. El objetivo de los simuladores no es dar una representación exacta del entorno, sino ofrecer la sensación al usuario de que se encuentra en ese espacio, por ello se tomó cierta libertad en las medidas.

La estructura del laboratorio (paredes y suelo) está modelada con formas básicas ofrecidas por Unity, en concreto planos y cubos. La ventaja de estas formas es que, al ser simples, contienen un número limitado de polígonos, lo que facilita el *render* y por lo

tanto mejora el rendimiento de la aplicación, ofreciendo mayor número de fotogramas por segundo. Es por esto, que hay que controlar el detalle y la complejidad de los objetos que se importan en la escena, ya que la potencia de la tarjeta gráfica del ordenador es limitada. Si se quiere mayor nivel de detalle en los modelos, se requerirá de una tarjeta gráfica más potente.

El plano del suelo y las paredes están compuestas del ya comentado componente *Transform*, así como de *Colliders* y *Renderers*. *Collider* es en este caso, hace que los objetos no puedan atravesarse. De este modo, el jugador no puede salir del espacio delimitado. Además, en el caso del suelo, hace que los objetos no se caigan al vacío. El componente *Renderer* aparece generalmente en todos los objetos de la escena, permite que el objeto en sí aparezca en pantalla. Ver Figura 4.9.

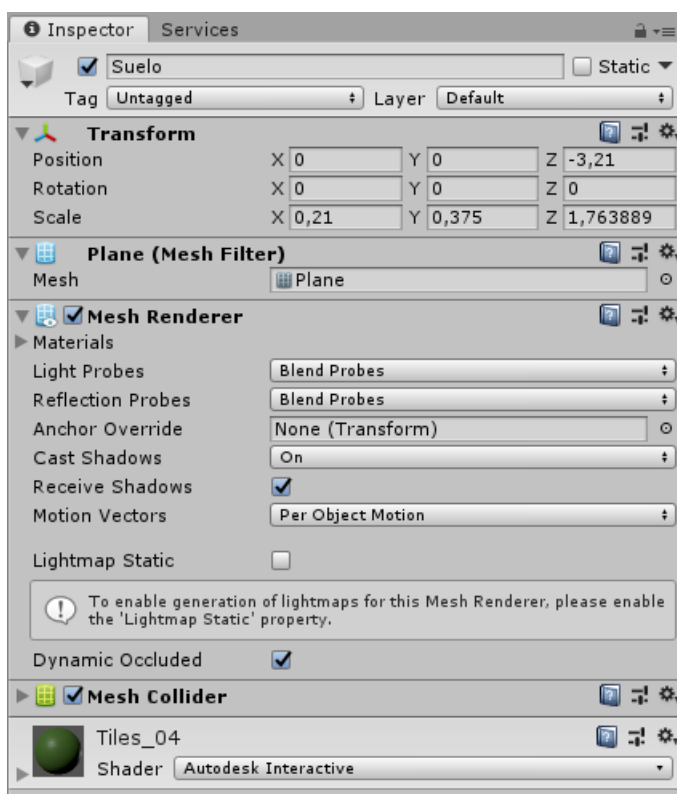


Figura 4.9: Ventana inspector del plano del suelo.

Los otros objetos como mesas de laboratorio, ordenadores, ventanas, rejillas y demás, han sido importados de *3D Warehouse*[1]. Éste es un catálogo online, perteneciente a SketchUp (Google), en el que los usuarios suben modelos creados por ellos. Todos los autores de los modelos utilizados están citados en el apéndice A. Para posicionar los diferentes objetos se ha hecho uso de nuevo del componente *Transform*, alterando la posición y, en algunos casos, la escala, para ajustarse lo mejor posible a la realidad.

4.2.1. Luces

En Unity, normalmente se ilumina la escena con una luz direccional, diferentes luces de punto y de foco. Las luces direccionales son aquellas que emiten con rayos paralelos y uniformemente distribuidos, iluminando todo de forma homogénea (como el Sol). Las luces de punto son aquellas en las que todos los rayos de luz se originan desde un mismo punto (como una bombilla). Por otro lado en las luces de foco los rayos se originan desde un punto, pero se proyectan en forma de cono. [5].

Para este proyecto se ha utilizado una luz direccional y un foco. El motivo de utilizar tan pocas luces es el rendimiento de la tarjeta gráfica. Según la documentación de Unity[18] cuantas más luces haya, mayor será la reducción de fotogramas por segundo (FPS). Al usar el visor de realidad virtual, se experimenta un retraso en la actualización de las imágenes, resultando en fragmentos negros o imágenes deformadas que causan mareo. Una de las posibles soluciones sería modificar la forma en que se iluminan los objetos de la escena, pero esto está fuera del alcance de este trabajo.

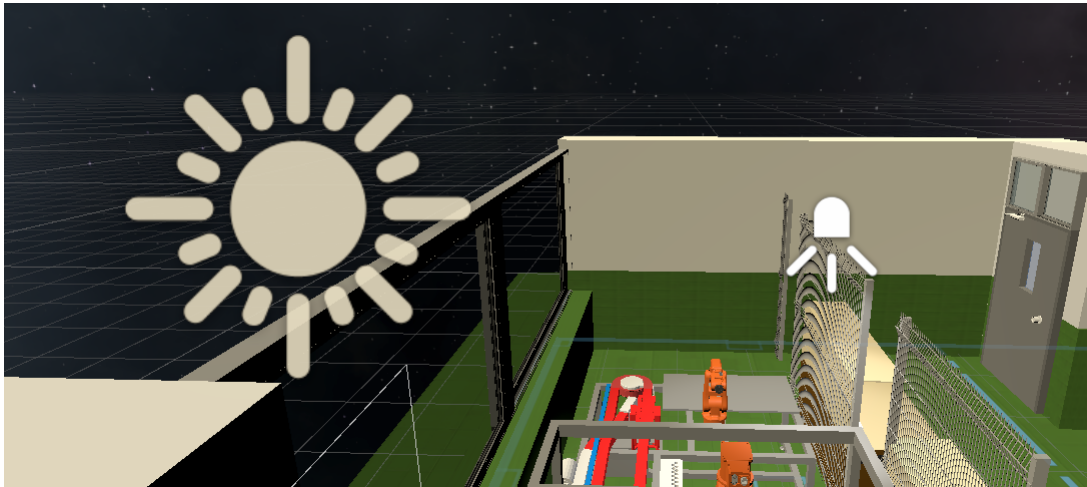


Figura 4.10: Vista de los gizmos de las luces.

Al principio se utilizó un techo en el modelo del laboratorio, que impedía que la luz direccional entrase, por lo que se tuvieron que añadir numerosas luces interiores de punto y foco. Al añadir estas luces se experimentó el problema comentado anteriormente. La solución final fue quitar el techo del laboratorio, y aprovechar la luz direccional, además de añadir otra luz para iluminar la pared izquierda del laboratorio, que quedaba en penumbra. De este modo se consiguió una iluminación correcta sin comprometer el número de FPS.

4.2.2. Interfaz de contadores

En una de las columnas se ha añadido un objeto de Unity tipo *canvas*. Este objeto permite representar texto en la escena. Este *canvas* muestra al usuario el número de aciertos, el número de fallos y el tiempo que queda hasta la siguiente avería (Figura 4.11). Este tiempo se puede modificar en el script `FailureManager.cs` (9.4.3). Si el tiempo está a 0, significa que la planta está en fallo. El texto se actualiza cada fotograma, ya que está asociado a `FailureManager.cs`. Se explica con más detalles en la sección 5.1.



Figura 4.11: Interfaz de usuario para recuento de aciertos, fallos y temporizador.

Capítulo 5

Diseño del código

En este capítulo se explica cómo se ha diseñado el código. Unity hace uso de programación orientada a objetos. El funcionamiento de la aplicación gira en torno al gestor de fallos. Éste se encarga de coordinar los demás elementos de la escena, específicamente, los dispositivos que son susceptibles de avería. Éstos últimos funcionan cada uno de forma independiente, pero supervisados por el gestor. Los objetos se comunican directamente a través de 'mensajes' que llegan al gestor de fallos, y éste responde según la situación.

5.1. Gestor de fallos



Figura 5.1: Modelo que representa el gestor de fallos en la escena.

El gestor de fallos es el encargado de coordinar todos los eventos que se dan en la escena. Se ha designado uno de los ordenadores como el *Game Object* que representa el gestor de fallos(Ver Figura 5.1). Éste contiene el código FailureManager.cs (9.4.3). Este código contiene campos públicos modificables en los que añadir el número de dispositivos susceptibles de fallos así como campos para cada uno de esos objetos(Ver Figura 5.2).

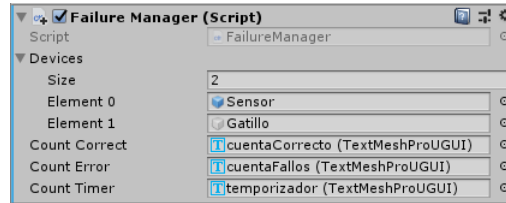


Figura 5.2: Ventana inspector del modelo del gestor de fallos.

1. Variables:

- `devices[]`: Vector de *Game Objects* en el que se almacenan los objetos susceptibles de avería. Variable pública que se puede modificar directamente desde Unity (tamaño y *Game Objects* que contiene).
- `n`: Índice de `devices[]` de la avería actual.
- `failure_id`: Almacena el ID de la avería actual.
- `object_id`: ID del objeto que ha sido pulsado.
- `OnFailure`: Booleano que indica si la planta está actualmente en estado de avería o funcionando correctamente.
- `TimeLeft`: Tiempo hasta que se produzca el siguiente fallo en la planta.
- `c_correct`: Contador de número de aciertos.
- `c_error`: Contador de número de errores.
- `countCorrect`: Variable del tipo *TextMeshPro(TMP)*, convierte el número de errores en texto *TMP*, para ser mostrado en la escena.
- `countError`: Igual que `countCorrect`, pero cuenta errores.
- `countTimer`: igual que `countCorrect`, pero muestra el tiempo restante hasta siguiente avería.

2. Funciones:

- `Update()`: Función que se llama cada fotograma. Se encarga de actualizar las variables del tipo *TMP*.
- `FailureTimer()`: Función temporizador que realiza la cuenta atrás hasta la siguiente avería.
- `FailureSet()`: Se encarga de seleccionar aleatoriamente una de las averías especificadas en `devices[]`, y envía al elemento elegido el mensaje para activar su función de avería.
- `OptionCheck(int id)`: Recibe el ID de un objeto pulsado por el usuario y lo compara con el ID de la avería actual. En caso de ser el mismo, envía el mensaje correspondiente al objeto averiado para que vuelva a su estado normal. En caso contrario llama a `OptionFalse()`.
- `OptionFalse()`: Suma un error.
- `StoreFailureID(int id)`: Recibe el ID de la avería actual y lo almacena en `failure_id`.

5.2. Dispositivos susceptibles de fallo

El sensor de movimiento y el gatillo son los elementos de este proyecto susceptibles de avería. Ambos se han codificado utilizando la plantilla incluida en la sección 5.5. En esta sección se explica el código referente exclusivamente a la función de avería de los dispositivos. Para ver las otras funciones, hay que dirigirse a la sección 5.5.

5.2.1. Sensor de movimiento

El sensor de movimiento es un dispositivo situado al lado de la cinta (Ver figura 4.6), que detecta cuando un pallet está adyacente. Cuando pasa un pallet, el LED del sensor se ilumina. En ocasiones el sensor no responde bien y no detecta el paso del pallet. Para identificar esta avería hay que fijarse en dicha luz. Si el sensor se encuentra fallando, el LED no se iluminará al paso del pallet. El código de este dispositivo, `MovementSensor.cs` (9.4.4) se encuentra en el *Game Object* 'sensor' que pertenece a su vez al *Game Object* 'movement sensor'.

1. Variables:

- `myLight`: Luz del sensor

2. Funciones:

- `Start()`: Función que se llama al inicio de la escena. Apaga la luz del LED.
- `Update()`: Se llama en cada fotograma. Esta función hace que el objeto del sensor identifique qué clase de objeto está pasando al lado. Si es un pallet el sensor se iluminará en caso de funcionar normalmente. En caso de avería, el LED no se encenderá.
- `LightOff()`: Apaga la luz.
- `LigthOn()`: Enciende la luz.

5.2.2. Gatillo

Este dispositivo se encarga de bloquear el paso del pallet en un punto de la cinta transportadora (Figura 4.7). Funciona a través de una válvula de aire comprimido. A veces el gatillo se atasca, por lo que a pesar de estar la válvula en funcionamiento el gatillo no se acciona. Para identificar esta avería hay que activar el gatillo. En la planta real, el gatillo se acciona mediante código. Como no se puede meter el mismo código en Unity, se ha añadido un interruptor manual dentro de la planta virtual que acciona el gatillo como si se hubiese accionado por código. Si se acciona el interruptor y el pallet no se para en el lugar indicado, significa que está fallando. Además, se ha incluido un LED en la válvula (Figura 4.8), que se encuentra debajo de las cintas transportadoras, que también indica cuando debería estar actuando el gatillo (encendido) y cuando no (apagado). El archivo de código es `StoppingDevice.cs` (9.4.8).

1. Variables:

- `belt`: Cinta sobre la que está situado el gatillo. Se especifica en el editor de Unity.
- `myLight`: Luz situada en la válvula (Figura 4.8)
- `button`: Booleano que indica si el interruptor ha sido accionado o no.
- `active`: Booleano que indica si el gatillo está bloqueando la cinta.

2. Funciones:

- `Start()`: Función que se llama al inicio de la escena. Apaga la luz de la válvula.
- `Update()`: Se llama en cada fotograma. Controla el funcionamiento del gatillo. Según el gatillo esté activo o no, modificará la posición del mismo para bloquear la cinta, siempre y cuando no se encuentre en estado de avería.
- `OnCollisionEnter()` y `OnCollisionExit()`: Estas funciones son auxiliares, evitan movimientos indeseados del *pallet* cuando los *colliders* de los objetos están en contacto.
- `DeviceActivated()`: Activa el gatillo y enciende la luz de la válvula.
- `DeviceOff()`: Desactiva el gatillo y apaga la luz de la válvula.

5.3. Objetos pulsables activos y pasivos

Para poder interactuar con los objetos desde la realidad virtual es necesario incluir diferentes scripts. Se han diferenciado dos tipos de objetos pulsables: los activos y los pasivos. Los activos son aquellos que tienen algún tipo de función, en este caso son el sensor de movimiento, el gatillo y el interruptor del gatillo. El resto de objetos que se pueden pulsar son pasivos, como los robots o la planta.

Para que se pueda interactuar con los objetos se necesitan tres scripts, proporcionados por Steam VR[16]: `Interactable.cs` (9.5.7), `Button.cs`(9.4.10) y `UIElement.cs`(9.5.20). Basta con arrastrar los scripts al *Game Object* correspondiente. Hay que tener en cuenta que el *Game Object* debe tener un componente *Collider*, que delimitará la zona a la que hay que acercar la mano para que se pueda pulsar el objeto. Finalmente, en el código `UIElement.cs` (9.5.20) hay que especificar qué función activa el objeto cuando es pulsado. Para ello, este mismo código ofrece unos campos en los que podemos especificar la función (pública) del *Game Object* (no tiene por qué ser el mismo) que se ha de llamar(Figura 5.3).

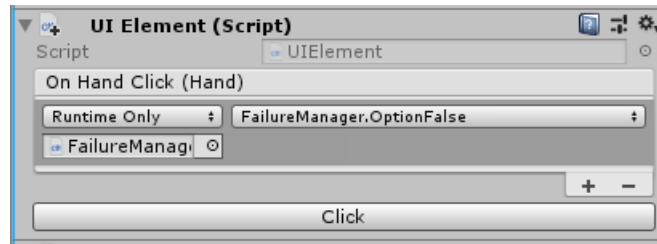


Figura 5.3: Vista del código UIElement.cs en el inspector.

Con estos códigos se crea una interfaz, que al acercar la mano sobre un objeto, el gatillo del mando y el objeto se resaltan en amarillo, de forma que el usuario sepa qué objeto está enfocando y qué botón pulsar. Ver Figura 5.4.

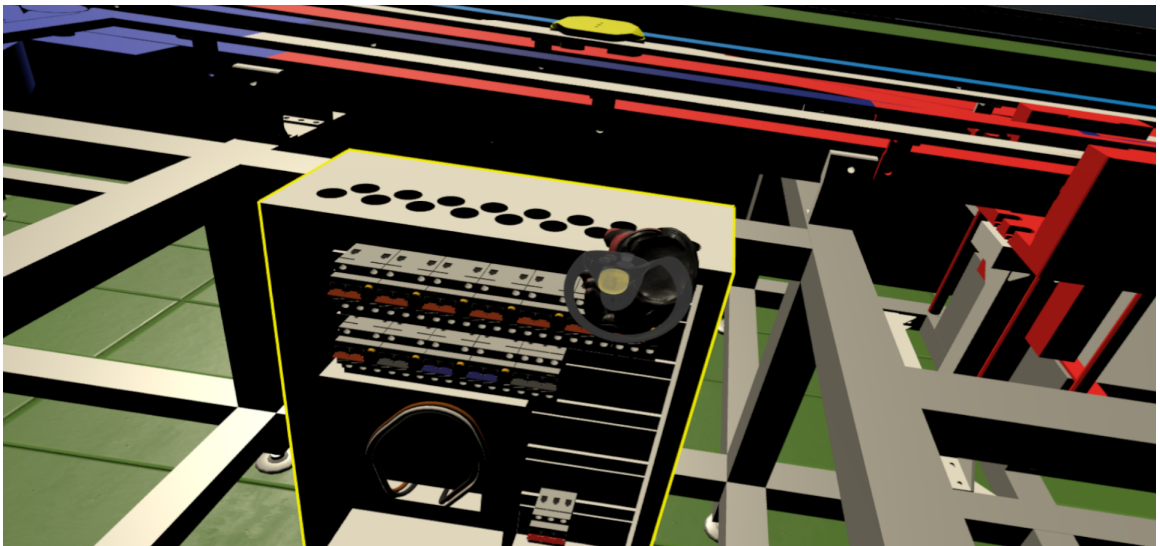


Figura 5.4: Ventana inspector del modelo del gestor de fallos.

Los scripts UIElement.cs del gatillo y el sensor están configurados para llamar a la función CheckOption() de FailureManager.cs(9.4.3). Los demás objetos llaman directamente a la función OptionFalse(). En el caso del interruptor del gatillo, éste llama a la función ButtonSwieth() de SDSwitch.cs(9.4.7), para activar/desactivar el dispositivo.

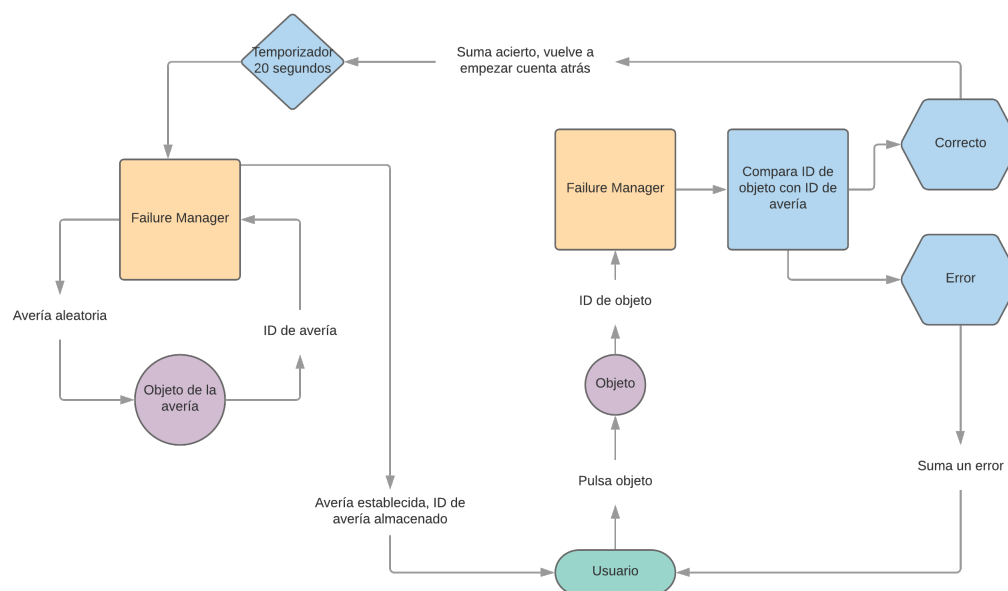


Figura 5.5: Diagrama resumen de las comunicaciones entre objetos.

5.4. Movimiento de las cintas

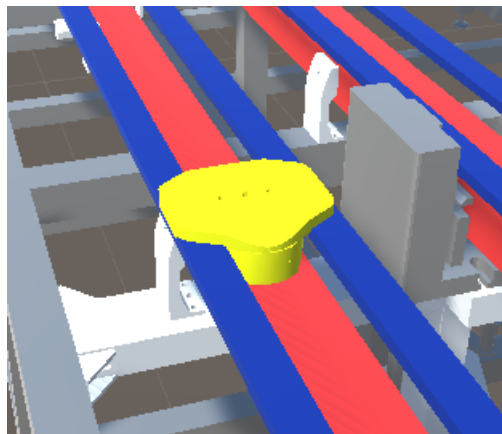


Figura 5.6: Pallet colocado en las cintas en Unity.

Para mover el pallet se hace uso de dos scripts para cada tipo de cinta. Para la cinta recta, se utiliza Belt.cs(9.4.1):

1. Variables:

- endpoint: Punto hacia donde se transporta el objeto.

- speed: Velocidad de la cinta.
- active: Booleano que indica si está activa la cinta o no.

2. Funciones:

- OnTriggerStay(Collider other): Al entrar en contacto con el collider de otro objeto, mueve dicho objeto hacia delante.
- StopMovement(): Para la cinta.
- IniMovement(): Inicia movimiento de la cinta.

Por otro lado, para la cinta curva, se han utilizado las ecuaciones de la circunferencia. En este script, CircularBelt.cs(9.4.2) se han declarado variables públicas como el centro de giro y el radio, para poder ser modificadas desde el editor de Unity y ajustadas a la forma de la cinta de la planta. El script permite un movimiento de rotación y traslación, aunque el de rotación se ha desactivado para este proyecto, por motivos de coordinación el otro movimiento y con la forma de la cinta. En caso de querer activarlo, simplemente hace falta establecer una velocidad de rotación desde el editor de Unity. El script contiene:

1. Variables:

- center: Centro de giro.
- radius: Radio de giro.
- traslationSpeed: Velocidad de traslación.
- posX: Posición en plano X del objeto a transportar.
- posZ: Posición en plano Z del objeto a transportar.
- angle: Ángulo de inicio.
- ini_angle: Ángulo de reinicio.
- rotationSpeed: Velocidad de rotación.

2. Funciones:

- OnTriggerStay(Collider other): Al entrar en contacto con el collider de otro objeto, mueve dicho objeto en una trayectoria circular de rotación y traslación.
- OnCollisionExit(): Resetea el ángulo cuando el pallet sale de la cinta transportadora.

Al ejecutarse estos scripts, el pallet, debido a la posición de los Colliders de las cintas, acumulaba un pequeño error en cada vuelta que hacía que se cayese. Para corregir este error, se añadió el código Pallet.cs(9.4.5) levanta el pallet de forma imperceptible en cada vuelta, de modo que cada vez que desciende cierta altura, se corrige su posición vertical a la que tenía inicialmente.

5.5. Añadir nuevos fallos

Debido al carácter de continuidad de este proyecto, se decidió añadir una plantilla de código para poder añadir nuevas averías, de modo que funcionen con el código actual de `FailureManager.cs`(5.1). La plantilla se encuentra en la sección 9.4.6. Esta plantilla contiene:

1. Variables:

- `failure_manager`: `GameObject` público. Hay que seleccionar el Game Object que contiene el código `FailureManager.cs`. Ver Figura 5.7.
- `id`: ID del fallo. Es el ID que se enviará a `FailureManager.cs` para que lo almacene/compare.
- `fail`: Booleano que indica si el dispositivo está fallando o no.

2. Funciones:

- `SensorFailure()`: Activa la avería y llama a `FailIDSet()`.
- `FailureDetected()`: Desactiva el fallo.
- `FailIDSet()`: Envía el ID del objeto a `FailureManager.cs` para que lo almacene.
- `SendID()`: En caso de ser pulsado, envía el ID a `FailureManager.cs` para que compare con la avería actual.

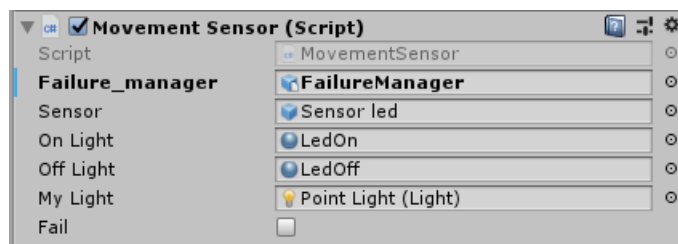


Figura 5.7: Vista del script de la plantilla en el inspector.

Las funciones anteriores están diseñadas para funcionar con `FailureManager.cs`, es importante no cambiar el nombre de las mismas, ya que al usar la función de unity 'SendMessage', cualquier cambio en el nombre, haría que se perdiera el mensaje entre los objetos. El código correspondiente a la avería tendrá que funcionar conforme a la variable 'fail'. De modo que si es 'true' el dispositivo esté fallando. También hay que asegurarse de que el ID no coincide con ninguno de los otros(Ver cuadro 6.1), lo que podría dar lugar a falsos aciertos.

Una vez se tenga diseñado el objeto, hay que añadir los scripts mencionados en la sección 5.3. En UIElement se seleccionará la función propia del objeto `SendID()`(Ver Figura 5.8). Finalmente, hay que añadir en los campos del inspector de `FailureManager.cs` en el *Game Object* 'Failure Manager', el nuevo *Game Object*, como se puede ver en la figura 5.2.

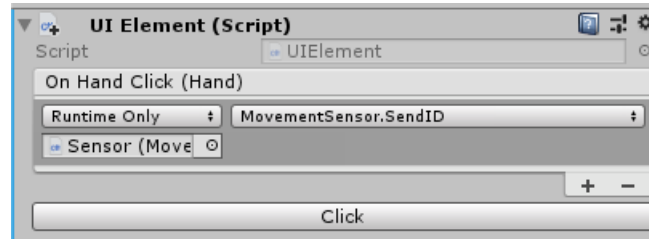


Figura 5.8: Vista del script de UIElement.cs en el inspector.

Dispositivo	GameObject	ID
Sensor movimiento	Sensor	101
Gatillo	Gatillo	102
Nuevo dispositivo	Nuevo GameObject	XXX

Cuadro 5.1: Tabla de objetos susceptibles de fallo.

Listing 5.1: Script plantilla para añadir nuevos dispositivos susceptibles de fallo

```

1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4
5  public class PlantillaFallo : MonoBehaviour
6  {
7
8      //Gestor de fallos.
9      [SerializeField]
10     public GameObject failure_manager;
11
12     //ID del fallo.
13     int id = 000;
14
15
16     //Fallo en el sensor
17     public bool fail = false;
18
19
20     // Start is called before the first frame update
21     void Start()
22     {
23
24     }
25
26     // Update is called once per frame
27     void Update()
28     {
29
30     }

```

```
31
32
33 //Funcion: SensorFailure()
34 //Descripcion: Activa la averia y envía el ID a Failure Manager para
    almacenarlo.
35 void SensorFailure()
36 {
37
38     fail = true;
39     FailIDSet();
40 }
41
42 //Funcion: Failure detected()
43 //Descripcion: Desactiva la averia.
44 void FailureDetected()
45 {
46     //Desactiva el fallo
47     fail = false;
48 }
49
50 //Función: FailIDSet()
51 //Descripción: Envía el ID del fallo a Failure Manager para que lo
    almacene.
52 public void FailIDSet()
53 {
54     failure_manager.SendMessage("StoreFailureID", id);
55 }
56
57 //Función: SendID()
58 //Descripción: Envía el id del fallo a Failure Manager para comprobar
    si es el correcto.
59 public void SendID()
60 {
61     failure_manager.SendMessage("OptionCheck", id);
62 }
63
64 }
```

Capítulo 6

Implementación en RV

La implementación de la RV en Unity se ha realizado a través de SteamVR. En un principio se comenzó utilizando el SDK propio de Oculus para Unity, pero presentaba problemas de compatibilidad entre actualizaciones de Unity y el propio *SDK*. Adicionalmente, SteamVR ofrece un mayor número de herramientas y permite la compatibilidad con otros dispositivos a parte de Oculus Rift, como el HTC Vive.



Figura 6.1: Representación de las manos y los mandos dentro de la aplicación.

6.1. Player

Player es el nombre del *Game Object* que representa al jugador en la escena. Es un 'prefab' de SteamVR que incluye todos los scripts necesarios para una implementación básica de la RV en Unity. Éste *Game Object* está compuesto a su vez de otros *Game*

Objects entre los que se incluyen las manos, el cuerpo y la cabeza(cámara). Cada uno de los *Game Objects* contiene uno o varios scripts de SteamVR que en conjunto, compatibilizan el hardware y el software. De este modo los sensores detectan el visor de realidad virtual y los mandos, y a través de SteamVR lo llevan a Unity. Los scripts utilizados de SteamVR están recogidos en la sección 9.5.

6.2. Controles

Oculus Rift incluye dos mandos, uno para cada mano(Figura 6.2). Dentro de la escena, aparecen representados los mandos en sí, además de las manos del usuario, que imitan los movimientos de las manos reales, para facilitar el uso de los controles(Figura 6.1). Para esta aplicación se han configurado los mandos de forma simétrica, de modo que el botón en uno de los mandos realiza la misma acción que el botón correspondiente en el otro mando. Los controles en esta aplicación tienen dos funciones principales: movimiento y selección de objetos.

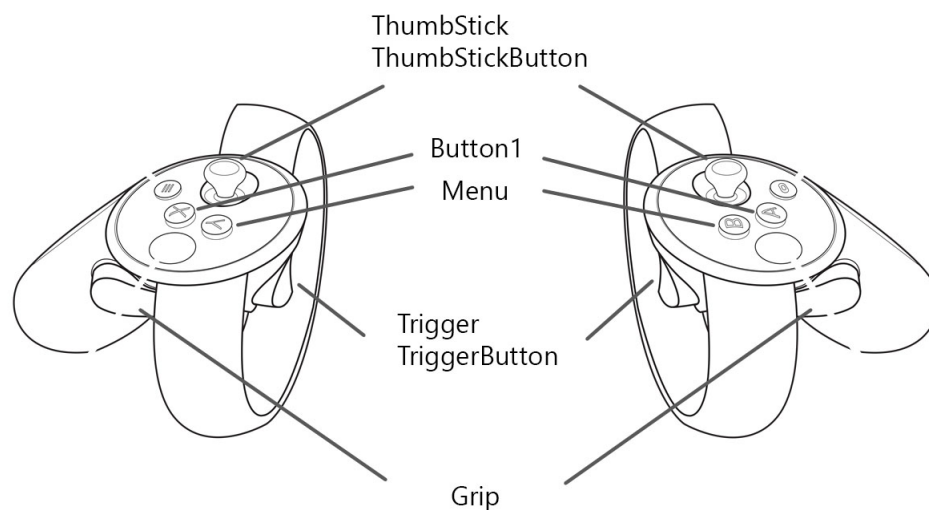


Figura 6.2: Controles Oculus Rift

Fuente: Documentación Unity[18]

Para moverse dentro del laboratorio virtual, se ha incluido un sistema de 'teletransporte'. Este sistema funciona a través de scripts de SteamVR. Es una forma fácil y segura de desplazarse en el entorno virtual, permitiendo avanzar grandes distancias sin moverse del sitio. Se ha optado por este sistema debido principalmente al límite de espacio que hay en el laboratorio, además de la longitud del cable del visor. Otra forma de movimiento que se probó fue el tradicional por *joysticks* utilizados en videojuegos

FPS(*First Person Shooter*). El principal problema que presenta este tipo de movimiento, es que, en realidad virtual, acaba resultando en mareos y pérdidas de equilibrio, quedando por lo tanto descartado.

Para activar el sistema de teletransporte, hay que pulsar cualquiera de los dos *joysticks*. Una vez pulsado aparecerá un haz parabólico desde el mando, y en el otro extremo un círculo iluminado(Figura 6.3). Al dejar de pulsar el *joystick*, el usuario se moverá al punto marcado por el círculo.

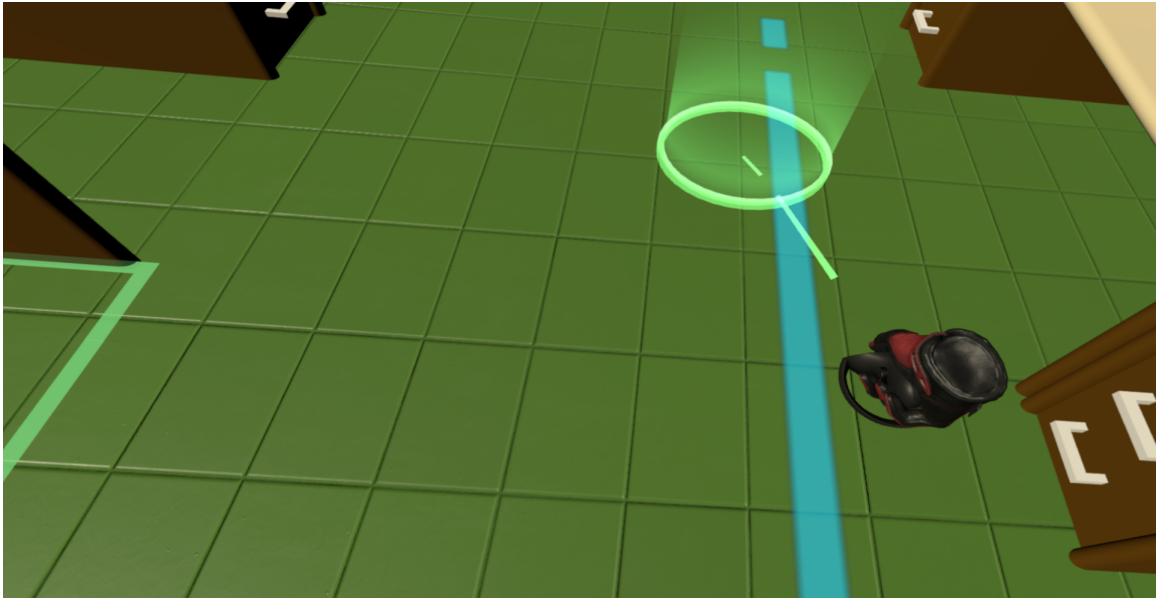


Figura 6.3: Sistema de teletransporte.

Para implementar este código, se tuvieron que agregar los scripts correspondientes al *Game Object* del jugador. Además, se tuvo que designar el área en el que el usuario puede moverse, mediante *TeleportArea.cs*(9.5.17). Para ello se asigna el script a un plano superpuesto con el plano del suelo del laboratorio(Figura 6.4).



Figura 6.4: Área de teletransporte ajustada al tamaño del laboratorio.

La selección de objetos permite al usuario interactuar con la aplicación, pudiendo seleccionar el objeto que crea que está averiado (Ver sección 5.3). Para seleccionar un objeto el usuario debe acercar la mano al objeto en cuestión, éste se resaltará en color amarillo. Para seleccionar ese objeto hay que pulsar el gatillo/*trigger* del mando con el que se está tocando el objeto deseado.

Capítulo 7

Puesta en marcha de Oculus Rift

1. Conectar el visor y los sensores al ordenador: Es recomendable conectar los USB del visor y los sensores a puertos USB 3.0 para disminuir la latencia. El HDMI del visor se ha de conectar al HDMI de la tarjeta gráfica, en caso contrario, no funcionará correctamente. Una vez conectados, se abrirá la aplicación de Oculus en el ordenador, si no se ha abierto, hay que abrirla manualmente. La aplicación hará de guía para continuar configurando el dispositivo.

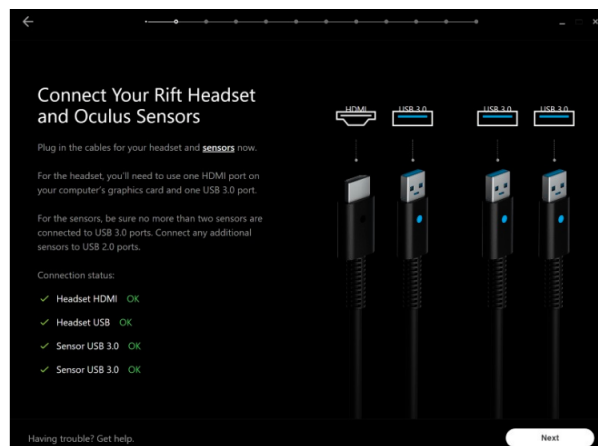


Figura 7.1: Primer paso.

2. Conectar mandos inalámbricos: Para encender los mandos hay que pulsar durante unos segundos el botón 'B' y 'oculus' a la vez. Hacer lo mismo con los botones correspondientes en el otro mando.

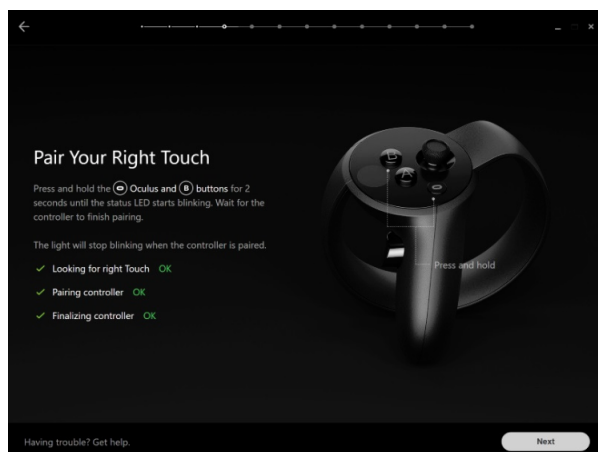


Figura 7.2: Segundo paso.

3. Posicionar los sensores: Hay que colocar los sensores de modo que cubran todo el área que se va a usar. La distancia entre los sensores recomendada es de 1 a 1,5 metros. Una vez colocados, seguimos los pasos de la pantalla, que verificará si se cubre el área mínima, y nos dirá si hay que modificar la posición de alguno de los sensores.

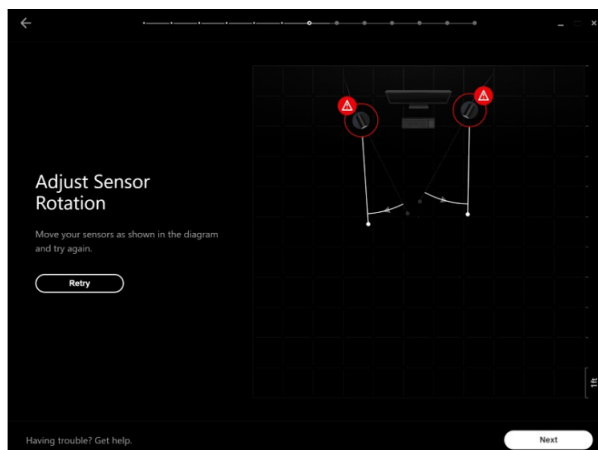


Figura 7.3: Tercer paso.

4. Delimitar el área de juego: En este paso hay que 'pintar' tal como indica la pantalla, el área que vayamos a utilizar. Este paso es importante, ya que colocará una malla virtual, que aparecerá cuando estemos muy próximos a ella, para evitar golpear a otras personas u objetos.

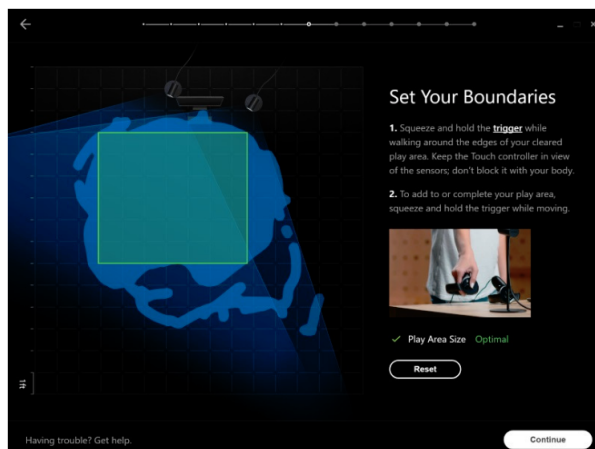


Figura 7.4: Cuarto paso.

5. Ajustar el visor a la cabeza y la distancia de las lentes.
6. Finalmente, hay que abrir SteamVR si no se ha abierto ya, y asegurarse de que todos los elementos han sido vinculados (aparecen en una ventana pequeña en la parte derecha inferior de la pantalla). Una vez hecho esto, ya se puede iniciar la aplicación .exe del proyecto.

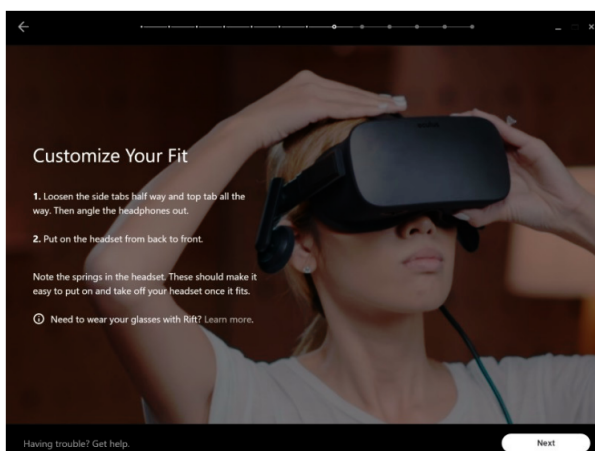


Figura 7.5: Quinto paso.

Capítulo 8

Conclusiones y desarrollos futuros

Unity es un motor centrado en la creación de aplicaciones de entretenimiento, y por lo tanto, carece de herramientas que faciliten el desarrollo de aplicaciones más técnicas, como compatibilidad directa con modelos en otros formatos como '.dwg' o '.sldasm'. Aun así, Unity Technologies está trabajando con grandes empresas, especialmente del sector del automóvil para solucionar los problemas.

Con el desarrollo de este proyecto, se demuestra, que, a pesar de las carencias técnicas de Unity, se pueden crear aplicaciones de simulación industriales, especialmente en el área de formación de personal. El uso de simuladores en cualquier entorno facilita el aprendizaje y la formación. En el caso de una gran fábrica industrial, el mantenimiento de la planta es de especial importancia, especialmente si los beneficios se cuentan por horas. Para formar correctamente a los técnicos haría falta parar la producción o esperar a que se produjese en una avería, lo que se traduce en importantes pérdidas económicas. No solo eso, sino que también, según el tipo de planta, se podría comprometer la seguridad de los técnicos. El uso de simuladores de este tipo permite al personal llegar a la planta real con la formación necesaria para afrontar el problema.

La gran ventaja de la realidad virtual frente a los simuladores convencionales es que no hace falta una réplica física de la máquina/dispositivo que se vaya a reparar. Se puede generar una réplica virtual que funcione de la misma manera que lo haría la real, a partir de los planos de este. De este modo hay un ahorro en el coste de producción del equipo, con la ventaja de que el entorno virtual se puede actualizar, y en caso de que se cambie algún componente de la planta de la fábrica, se podría hacer en el entorno virtual de la misma manera.

Algunas empresas, como PixoGroup[14], han desarrollado aplicaciones de formación que permiten el uso simultáneo de la aplicación de varios operarios a la vez, de modo que se puedan entrenar dinámicas en las que se requiera más de una persona, desarrollando un método de formación que ayuda a los alumnos a retener mejor lo que aprenden durante la formación.

En conclusión, los programas de entrenamiento en Realidad Virtual son una realidad presente que, aunque todavía no se haya establecido completamente, está experimentando un gran crecimiento. Grandes empresas industriales están invirtiendo en este tipo

de aplicaciones, convirtiéndose la Realidad Virtual en uno de los principales impulsores hacia la implementación de la Industria 4.0.

El futuro de este tipo de aplicaciones dependerá de cómo se desarrolle la Realidad Virtual. En el caso de programas de formación, se podrían generar sistemas de enseñanza que formasen y examinasen a los operarios, mediante un sistema de tutoriales y lecciones virtuales. En cuanto a aplicaciones directamente relacionadas con el mantenimiento, sería de especial interés ser capaces de conectar directamente la planta en sí con la aplicación, de modo que los movimientos y estado de la planta física se reflejen en la virtual, así operario podría diagnosticar los problemas reales de la planta sin necesidad de estar, ni si quiera, en el mismo edificio; o poder controlar la planta desde la realidad virtual, minimizando el contacto físico entre operario y maquinaria. Otro de los posibles desarrollos es el diseño de las plantas, mediante la creación de modelos a partir de planos de CAD, poder visualizar y testear una planta de manera virtual, para comprobar su funcionamiento y distribución. Ésto sería posible mediante la creación, en caso de Unity, de modelos 'prefabs' que se puedan reciclar para ser utilizados en diferentes proyectos, ahorrando costes de diseño y programación. Por otro lado, alejado de la industria manufacturera, se podrían diseñar aplicaciones que simulasen incidentes en centros donde los accidentes son inusuales pero críticos, como el caso de las centrales nucleares. En estas centrales es difícil entrenar operarios de manera realista, ya que las incidencias son mínimas, pero a la vez son muy peligrosas. Por esto, una formación previa en un simulador de realidad virtual podría, en casos extremos, prevenir catástrofes.

Parte II

Presupuesto

Capítulo 9

Presupuesto del proyecto

9.1. Ingeniería

Tarea	Horas	Precio/hora[€/h]	Precio[€]
Documentación	30	20.00	600.00
Anteproyecto	55	20.00	1100.00
Diseño	25	25.00	625.00
Programación	180	30	5400.00
TOTAL			7725.00

Cuadro 9.1: Presupuesto de ingeniería

9.2. Hardware y software

Hardware	Precio[€]
Oculus Rift	499.00
Ordenador RV	1400.00
Ordenador personal	800.00
TOTAL	2699.00

Cuadro 9.2: Presupuesto de Hardware.

Software	Precio[€]
Oculus VR	Gratuito
Steam VR	Gratuito
Unity	Gratuito
TOTAL	0.00

Cuadro 9.3: Presupuesto de Software.

9.3. Total

Sección	Precio[€]
Hardware	2699.00
Software	0.00
Ingeniería	7725.00
TOTAL	10424.00

Cuadro 9.4: Presupuesto total del proyecto.

Parte III

Código fuente

9.4. Código de Unity

9.4.1. Belt.cs

```
1 //Nombre: Belt.cs
2 //Descripcion: Mueve los objetos de la cinta recta hacia delante
3 //Entradas: speed(velocidad de mov.), endpoint(punto final de trayecto)
4 using System.Collections;
5 using System.Collections.Generic;
6 using UnityEngine;
7
8 public class Belt : MonoBehaviour {
9
10     //Punto final
11     public Transform endpoint;
12     //Velocidad
13     public float speed;
14     //Activo
15     bool active = true;
16
17     private void Update()
18     {
19
20     }
21
22     private void OnTriggerStay(Collider other)
23     {
24         //Activa el movimiento
25
26         //Debug.Log("Triggered");
27         Move(other);
28
29     }
30
31     private void Move(Collider other)
32     {
33         //Mueve el objeto hacia delante
34
35         //Debug.Log("Moving");
36         if (active)
37         {
38             other.transform.position = Vector3.MoveTowards(other.
39                 transform.position, endpoint.position, speed * Time.
40                 deltaTime);
41
42         }
43     }
44
45     void StopMovement()
46     {
47         active = false;
48     }
49
50     void IniMovement()
51     {
52     }
```

```

48         active = true;
49     }
50
51 }

```

9.4.2. CircularBelt.cs

```

1  //Nombre: CircularBelt.cs
2  //Descripcion: Movimiento circular para la cinta transportadora.
   Movimiento de rotación y traslación.
3  //Entradas: Center, radius, traslationSpeed, angle, iniAngle,
   rotationSpeed
4  using System.Collections;
5  using System.Collections.Generic;
6  using UnityEngine;
7
8  public class CircularBelt : MonoBehaviour {
9
10     //Centro de traslacion
11     [SerializeField]
12     Transform Center;
13
14     //Radio de traslacion. Aviso: Ajustar si se reescala la planta.
15     [SerializeField]
16     float radius = 2f;
17
18     //Velocidad de traslacion
19     [SerializeField]
20     float traslationSpeed = 2f;
21
22     float posX, posZ;
23
24     //Angulo inicio
25     [SerializeField]
26     float angle = -90f;
27
28     //Angulo de reseteo
29     [SerializeField]
30     float iniAngle = -90f;
31
32     //Velocida de rotacion
33     [SerializeField]
34     float rotationSpeed = 0.5f;
35
36     // Use this for initialization
37     void Start () {
38
39     }
40
41     // Update is called once per frame
42     void Update () {
43

```



```

44     }
45
46     private void OnTriggerStay(Collider other)
47     {
48         //Cálculo de posición
49         float posY = other.gameObject.transform.position.y;
50         posX = Center.position.x + Mathf.Cos(angle) * radius;
51         posZ = Center.position.z + Mathf.Sin(angle) * radius;
52
53         //Transformación de posición
54         other.transform.position = new Vector3(posX, posY, posZ);
55
56         //Actualización del ángulo de traslación
57         angle = angle + Time.deltaTime * traslationSpeed;
58
59         //Debug.Log(angle);
60
61         //Giro de rotación
62         other.transform.Rotate(Vector3.forward, rotationSpeed*Time.
            deltaTime);
63
64
65     }
66
67     private void OnCollisionExit(Collision collision)
68     {
69         //Reseteo del ángulo al entrar un objeto nuevo.
70         angle = iniAngle;
71     }
72 }
73
74
75 }

```

9.4.3. FailureManager.cs

```

1 //Nombre: FailureManager.cs
2 //Descripción: Escoge fallo de forma aleatoria. Administra las
   interacciones con los objetos, temporizador, aciertos y fallos.
3 //Entradas:
4
5 using System.Collections;
6 using System.Collections.Generic;
7 using UnityEngine;
8 using UnityEngine.UI;
9 using TPro;
10
11 public class FailureManager : MonoBehaviour
12 {
13     //GameObjects que contienen los scripts con la plantilla de fallo.
14     [SerializeField]
15     public GameObject[] devices;

```

```

16
17 //Índice de la avería actual, dependerá del orden en el que se hayan
    introducido los GameObjects en el campo anterior.
18 int n;
19
20 //ID del fallo actual
21 int failure_id;
22
23 //ID del objeto pulsado
24 int object_id;
25
26 //Indicador si la planta está en estado de fallo
27 bool OnFailure = false;
28
29 //Tiempo hasta siguiente fallo en la planta
30 float TimeLeft = 20.0f;
31 //Cuenta de aciertos y fallos
32 int c_correct=0;
33 int c_error=0;
34 int timeLeft_i = 0;
35 //IU para fallos, aciertos y temporizador.
36 public TMP_Text countCorrect;
37 public TMP_Text countError;
38 public TMP_Text countTimer;
39
40
41 // Start is called before the first frame update
42 void Start()
43 {
44
45 }
46
47 // Update is called once per frame
48 void Update()
49 {
50     FailureTimer();
51     countCorrect.text = "ACIERTOS: " + c_correct.ToString();
52     countError.text = "FALLOS: " + c_error.ToString();
53     countTimer.text = "TIEMPO: " + timeLeft_i.ToString();
54 }
55
56 //Función: FailureTimer()
57 //Descripción: Temporizador hasta siguiente avería.
58 //Entradas: -
59
60 void FailureTimer()
61 {
62     //Si la planta no se encuentra en estado de fallo, se inicia el
        temporizador.
63     if (!OnFailure)
64     {
65         TimeLeft -= Time.deltaTime;
66         Debug.Log(TimeLeft);
67         timeLeft_i = (int)TimeLeft;

```

```

68         if (TimeLeft < 0)
69         {
70             FailureSet();
71             TimeLeft = 20.0f;
72         }
73     }
74 }
75
76 //Función:FailureSet()
77 //Descripción: Selección aleatoria del fallo
78 //Entradas: -
79
80 void FailureSet()
81 {
82     OnFailure = true;
83     n = Random.Range(0, devices.Length);
84     Debug.Log("Length=" + devices.Length);
85     devices[n].SendMessage("SensorFailure");
86     Debug.Log("Fallo establecido =" + n);
87
88 }
89
90 //Función:OptionCheck(int id)
91 //Descripción: Comprueba si el objeto seleccionado es el fallo. Si es
92 //correcto, desactiva la avería.
93 //Entradas: id
94
95 void OptionCheck(int id)
96 {
97     if (id == failure_id)
98     {
99         OnFailure = false;
100         devices[n].SendMessage("FailureDetected");
101         Debug.Log("Averia detectada");
102         if(timeLeft_i==0)
103             c_correct++;
104         //Mensaje de acierto
105     }
106     else
107     {
108         OptionFalse();
109     }
110 }
111
112 //Función:OptionFalse()
113 //Descripción: Se llama cuando el objeto seleccionado no es el
114 //correcto. Suma un fallo.
115 //Entradas: -
116
117 public void OptionFalse()
118 {
119

```

```

120         //Mensaje de fallo
121         Debug.Log("Opción falsa");
122         if(timeLeft_i==0)
123             c_error++;
124     }
125
126     //Función: StoreFailureID(int id)
127     //Descripción: Almacena el ID del dispositivo que está fallando
128     //Entradas: id
129
130     void StoreFailureID(int id)
131     {
132         failure_id = id;
133     }
134
135
136
137
138
139 }

```

9.4.4. MovementSensor.cs

```

1  //Nombre: MovementSensor.cs
2  //Descripcion: Detecta el movimiento de pallets al pasar por el sensor.
   Activa y desactiva el fallo en el sensor.
3  //Entradas: Luz, sensor*, materiales sensor.
4  //*Objeto que contiene la luz del sensor
5
6  using System.Collections;
7  using System.Collections.Generic;
8  using UnityEngine;
9
10 public class MovementSensor : MonoBehaviour {
11
12     //Gestor de fallos
13     [SerializeField]
14     public GameObject failure_manager;
15     //ID del fallo
16     int id = 101;
17     //Objeto que contiene la luz
18     public GameObject sensor;
19     //Materiales del objeto en encendido/apagado
20     public Material onLight;
21     public Material offLight;
22     //Luz
23     public Light myLight;
24
25     //Fallo en el sensor
26     public bool fail = false;
27
28

```

```

29 // Use this for initialization
30 void Start () {
31     //Empieza con luz apagada
32     LightOff();
33 }
34
35 // Update is called once per frame
36 void Update () {
37
38
39     RaycastHit hit;
40
41     //Debug.DrawRay(transform.position, -Vector3.up, Color.blue);
42
43     //Si el objeto es un pallet y no hay error en el sensor,
44     //se enciende la luz. En caso contrario se apaga.
45     if (Physics.Raycast(transform.position, -Vector3.up, out hit
46         ,100.0f))
47     {
48         //Debug.Log("Objeto detectado");
49
50         if (hit.transform.gameObject.tag == "pallet" && fail==false)
51         {
52             //Debug.Log("Sensor activado");
53             LightOn();
54         }
55         else
56         {
57             LightOff();
58         }
59     }
60 }
61
62 //Funcion: LightOn()
63 //Descripcion: Enciende la luz del sensor.
64 void LightOn()
65 {
66     //Cambio de material del objeto LED
67     sensor.GetComponent<Renderer>().material = onLight;
68     //Luz encendida
69     myLight.enabled = true;
70
71 }
72
73 //Funcion: LightOff()
74 //Descripcion: Apaga la luz del sensor.
75 void LightOff()
76 {
77     //Cambio de material del objeto LED
78     sensor.GetComponent<Renderer>().material = offLight;
79     //Luz apagada
80     myLight.enabled = false;
81 }

```

```

82
83 //Funcion: SensorFailure()                                PLANTILLA
84 //Descripcion: Activa la averia y envía el ID a Failure Manager para
      almacenarlo.
85 void SensorFailure()
86 {
87
88     fail = true;
89     FailIDSet();
90     Debug.Log("Avería sensor movimiento");
91 }
92
93 //Funcion: Failure detected()                                PLANTILLA
94 //Descripcion: Desactiva la averia.
95 void FailureDetected()
96 {
97     //Desactiva el fallo
98     fail = false;
99 }
100
101 //Función: FailIDSet()                                    PLANTILLA
102 //Descripción: Envía el ID del fallo a Failure Manager para que lo
      almacene.
103 public void FailIDSet()
104 {
105     failure_manager.SendMessage("StoreFailureID", id);
106 }
107
108 //Función: SendID()                                        PLANTILLA
109 //Descripción: Envía el id del fallo a Failure Manager para comprobar
      si es el correcto.
110 public void SendID()
111 {
112     failure_manager.SendMessage("OptionCheck", id);
113 }
114 }

```

9.4.5. Pallet.cs

```

1 //Nombre: Pallet.cs
2 //Descripción: Corrige rotacion del pallet y la altura.
3 //Entradas: -
4 using System.Collections;
5 using System.Collections.Generic;
6 using UnityEngine;
7
8 public class Pallet : MonoBehaviour
9 {
10     // Start is called before the first frame update
11     void Start()
12     {
13

```

```

14     }
15
16     // Update is called once per frame
17     void Update()
18     {
19         //Comprobar rotación del pallet
20         if (transform.rotation.z != 0 || transform.rotation.x != -90)
21         {
22             //Corrige rotación del pallet
23             transform.eulerAngles = new Vector3(-90, 0, 0);
24         }
25
26         if (transform.position.y < 0.83f)
27         {
28             Debug.Log("Subir");
29             while (transform.position.y < 0.83f)
30             {
31                 transform.Translate(Vector3.up * Time.deltaTime, Space.
32                     World);
33             }
34         }
35     }

```

9.4.6. Plantilla Fallo.cs

```

1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4
5  public class PlantillaFallo : MonoBehaviour
6  {
7
8      //Gestor de fallos.
9      [SerializeField]
10     public GameObject failure_manager;
11
12     //ID del fallo.
13     int id = 000;
14
15
16     //Fallo en el sensor
17     public bool fail = false;
18
19
20     // Start is called before the first frame update
21     void Start()
22     {
23
24     }
25
26     // Update is called once per frame

```

```

27     void Update()
28     {
29
30     }
31
32
33     //Funcion: SensorFailure()
34     //Descripcion: Activa la averia y envía el ID a Failure Manager para
        almacenarlo.
35     void SensorFailure()
36     {
37
38         fail = true;
39         FailIDSet();
40     }
41
42     //Funcion: Failure detected()
43     //Descripcion: Desactiva la averia.
44     void FailureDetected()
45     {
46         //Desactiva el fallo
47         fail = false;
48     }
49
50     //Función: FailIDSet()
51     //Descripción: Envía el ID del fallo a Failure Manager para que lo
        almacene.
52     public void FailIDSet()
53     {
54         failure_manager.SendMessage("StoreFailureID", id);
55     }
56
57     //Función: SendID()
58     //Descripción: Envía el id del fallo a Failure Manager para comprobar
        si es el correcto.
59     public void SendID()
60     {
61         failure_manager.SendMessage("OptionCheck", id);
62     }
63
64 }

```

9.4.7. SDSwitch.cs

```

1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4
5  public class SDSwitch : MonoBehaviour
6  {
7      //Stopping device
8      [SerializeField]

```



```
9      GameObject device;
10
11      bool active = false;
12
13
14      // Start is called before the first frame update
15      void Start()
16      {
17
18      }
19
20      // Update is called once per frame
21      void Update()
22      {
23
24      }
25
26      public void ButtonSwitch()
27      {
28          active = !active;
29          if (active)
30          {
31              device.SendMessage("DeviceActivated");
32              transform.Rotate(0, 0, 90);
33              Debug.Log("Activated");
34          }
35          else if (!active)
36          {
37              device.SendMessage("DeviceOff");
38              transform.Rotate(0, 0, -90);
39              Debug.Log("Off");
40          }
41      }
42  }
43 }
```

9.4.8. StoppingDevice.cs

```
1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4
5  public class StoppingDevice : MonoBehaviour
6  {
7
8      //Gestor de fallos.
9      [SerializeField]
10     public GameObject failure_manager;
11
12     //ID del fallo.
13     int id = 102;
14 }
```

```

15 //Cinta sobre la que está el gatillo
16 [SerializeField]
17 GameObject belt;
18
19 //Luz de la válvula
20 [SerializeField]
21 Light myLight;
22
23 //Indica la pulsación del botón que activa el gatillo.
24 public bool button = false;
25
26 //Fallo
27 public bool fail = false;
28 //Active False: no bloquea el paso    True: Bloquea el paso
29 public bool active = false;
30 // Start is called before the first frame update
31 void Start()
32 {
33     myLight.enabled = false;
34 }
35
36 // Update is called once per frame
37 void Update()
38 {
39
40     if (active && fail == false)
41     {
42
43         transform.localPosition = new Vector3(0, -0.09f, -2.05f);
44
45
46     } else if (!active && fail == false)
47     {
48         transform.localPosition = Vector3.zero;
49
50     } else
51     {
52         //Si hay fallo, el gatillo se queda en la misma posición.
53         transform.localPosition = transform.localPosition;
54     }
55 }
56
57 //Funciones OnCollisionEnter/OnCollisionExit
58 //Descripción: Evita movimientos no deseados en el pallet cuando el
    gatillo está activado
59 //Entradas: collision
60 private void OnCollisionEnter(Collision collision)
61 {
62     if (collision.gameObject.tag == "pallet")
63     {
64         collision.rigidbody.velocity = Vector3.zero;
65         belt.SendMessage("StopMovement");
66     }
67

```

```
68     }
69     private void OnCollisionExit(Collision collision)
70     {
71         //Reanuda el movimiento de la cinta
72         if (collision.gameObject.tag == "pallet")
73         {
74             belt.SendMessage("IniMovement");
75         }
76     }
77
78     //Función: DeviceActivated()
79     //Descripción: Activa el gatillo
80     //Entradas: -
81     void DeviceActivated()
82     {
83         active = true;
84         myLight.enabled = true;
85     }
86
87     //Función: DeviceOff()
88     //Descripción: Desactiva el gatillo.
89     //Entradas: -
90     void DeviceOff()
91     {
92         active = false;
93         myLight.enabled = false;
94     }
95
96
97
98
99
100    //Funcion: SensorFailure()
101    //Descripcion: Activa la averia y envía el ID a Failure Manager para
102    almacenarlo.
103    void SensorFailure()
104    {
105        fail = true;
106        FailIDSet();
107        //Debug.Log("Averia gatillo");
108    }
109
110    //Funcion: Failure detected()
111    //Descripcion: Desactiva la averia.
112    void FailureDetected()
113    {
114        //Desactiva el fallo
115        fail = false;
116    }
117
118    //Función: FailIDSet()
119    //Descripción: Envía el ID del fallo a Failure Manager para que lo
120    almacene.
```

```

120     public void FailIDSet()
121     {
122         failure_manager.SendMessage("StoreFailureID", id);
123     }
124
125     //Función: SendID()
126     //Descripción: Envía el id del fallo a Failure Manager para comprobar
127     //                si es el correcto.
128     public void SendID()
129     {
130         failure_manager.SendMessage("OptionCheck", id);
131         Debug.Log("ID enviado =" + id);
132     }
133 }
134

```

9.4.9. ToggleLight.cs

```

1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4
5  public class ToggleLight : MonoBehaviour {
6
7      // Use this for initialization
8      void Start () {
9
10     }
11
12     // Update is called once per frame
13     void Update () {
14
15     }
16
17     //void LightOn()
18     //{
19         //    //Luz encendida
20         //    enabled = true;
21
22     //}
23
24     //void LightOff()
25     //{
26         //    //Luz encendida
27         //    enabled = false;
28     //}
29 }

```

9.4.10. Button.cs

```

1  using System;
2  using System.Collections;
3  using UnityEngine.Events;
4  using UnityEngine.EventSystems;
5  using UnityEngine.Serialization;
6
7  namespace UnityEngine.UI
8  {
9      /// <summary>
10     /// A standard button that sends an event when clicked.
11     /// </summary>
12     [AddComponentMenu("UI/Button", 30)]
13     public class Button : Selectable, IPointerClickHandler,
14         ISubmitHandler
15     {
16         [Serializable]
17         /// <summary>
18         /// Function definition for a button click event.
19         /// </summary>
20         public class ButtonClickedEvent : UnityEvent {}
21
22         // Event delegates triggered on click.
23         [FormerlySerializedAs("onClick")]
24         [SerializeField]
25         private ButtonClickedEvent m_OnClick = new ButtonClickedEvent();
26
27         protected Button()
28         {}
29
30         /// <summary>
31         /// UnityEvent that is triggered when the button is pressed.
32         /// Note: Triggered on MouseUp after MouseDown on the same object
33         .
34         /// </summary>
35         /// <example>
36         /// <code>
37         /// using UnityEngine;
38         /// using UnityEngine.UI;
39         /// using System.Collections;
40         ///
41         /// public class ClickExample : MonoBehaviour
42         /// {
43         ///     public Button yourButton;
44         ///
45         ///     void Start()
46         ///     {
47         ///         Button btn = yourButton.GetComponent<Button>();
48         ///         btn.onClick.AddListener(TaskOnClick);
49         ///     }
50         ///
51         ///     void TaskOnClick()
52         ///     {
53         ///         Debug.Log("You have clicked the button!");
54         ///     }
55         /// }
56     }
57 }

```

```

52     ///     }
53     /// }
54     ///</code>
55     ///</example>
56     public ButtonClickedEvent onClick
57     {
58         get { return m_OnClick; }
59         set { m_OnClick = value; }
60     }
61
62     private void Press()
63     {
64         if (!IsActive() || !IsInteractable())
65             return;
66
67         UISystemProfilerApi.AddMarker("Button.onClick", this);
68         m_OnClick.Invoke();
69     }
70
71     /// <summary>
72     /// Call all registered IPointerClickHandlers.
73     /// Register button presses using the IPointerClickHandler. You
74     /// can also use it to tell what type of click happened (left,
75     /// right etc.).
76     /// Make sure your Scene has an EventSystem.
77     /// </summary>
78     /// <param name="eventData">Pointer Data associated with the
79     /// event. Typically by the event system.</param>
80     /// <example>
81     /// <code>
82     /// //Attatch this script to a Button GameObject
83     /// using UnityEngine;
84     /// using UnityEngine.EventSystems;
85     ///
86     /// public class Example : MonoBehaviour, IPointerClickHandler
87     /// {
88     ///     //Detect if a click occurs
89     ///     public void OnPointerClick(PointerEventData
90     ///         pointerEventData)
91     ///     {
92     ///         //Use this to tell when the user right-clicks on
93     ///         the Button
94     ///         if (pointerEventData.button == PointerEventData.
95     ///             InputButton.Right)
96     ///         {
97     ///             //Output to console the clicked GameObject's name
98     ///             and the following message. You can replace this with your
99     ///             own actions for when clicking the GameObject.
100            Debug.Log(name + " Game Object Right Clicked!");
101        }
102    }
103    ///
104    /// //Use this to tell when the user left-clicks on the
105    /// Button

```

```

96         ///         if (pointerEventData.button == PointerEventData.
           InputButton.Left)
97         ///         {
98         ///             Debug.Log(name + " Game Object Left Clicked!");
99         ///         }
100        ///     }
101    /// }
102    /// </code>
103    /// </example>
104
105    public virtual void OnPointerClick(PointerEventData eventData)
106    {
107        if (eventData.button != PointerEventData.InputButton.Left)
108            return;
109
110        Press();
111    }
112
113    /// <summary>
114    /// Call all registered ISubmitHandler.
115    /// </summary>
116    /// <param name="eventData">Associated data with the event.
           Typically by the event system.</param>
117    /// <remarks>
118    /// This detects when a Button has been selected via a "submit"
           key you specify (default is the return key).
119    ///
120    /// To change the submit key, either:
121    ///
122    /// 1. Go to Edit->Project Settings->Input.
123    ///
124    /// 2. Next, expand the Axes section and go to the Submit section
           if it exists.
125    ///
126    /// 3. If Submit doesnt exist, add 1 number to the Size field.
           This creates a new section at the bottom. Expand the new
           section and change the Name field to Submit.
127    ///
128    /// 4. Change the Positive Button field to the key you want (e.g.
           space).
129    ///
130    ///
131    /// Or:
132    ///
133    /// 1. Go to your EventSystem in your Project
134    ///
135    /// 2. Go to the Inspector window and change the Submit Button
           field to one of the sections in the Input Manager (e.g. "
           Submit"), or create your own by naming it what you like, then
           following the next few steps.
136    ///
137    /// 3. Go to Edit->Project Settings->Input to get to the Input
           Manager.
138    ///

```

```

139     /// 4. Expand the Axes section in the Inspector window. Add 1 to
        the number in the Size field. This creates a new section at
        the bottom.
140     ///
141     /// 5. Expand the new section and name it the same as the name
        you inserted in the Submit Button field in the EventSystem.
        Set the Positive Button field to the key you want (e.g. space
        )
142     /// </remarks>
143
144     public virtual void OnSubmit(BaseEventData eventData)
145     {
146         Press();
147
148         // if we get set disabled during the press
149         // don't run the coroutine.
150         if (!IsActive() || !IsInteractable())
151             return;
152
153         DoStateTransition(SelectionState.Pressed, false);
154         StartCoroutine(OnFinishSubmit());
155     }
156
157     private IEnumerator OnFinishSubmit()
158     {
159         var fadeTime = colors.fadeDuration;
160         var elapsedTime = 0f;
161
162         while (elapsedTime < fadeTime)
163         {
164             elapsedTime += Time.unscaledDeltaTime;
165             yield return null;
166         }
167
168         DoStateTransition(currentSelectionState, false);
169     }
170 }
171 }

```

9.5. Código de Steam VR

9.5.1. BodyCollider.cs

```

1  //===== Copyright (c) Valve Corporation, All rights reserved.
        =====
2  //
3  // Purpose: Collider dangling from the player's head
4  //
5  //
        =====

```



```

6
7 using UnityEngine;
8 using System.Collections;
9
10 namespace Valve.VR.InteractionSystem
11 {
12     //
13     [RequireComponent( typeof( CapsuleCollider ) )]
14     public class BodyCollider : MonoBehaviour
15     {
16         public Transform head;
17
18         private CapsuleCollider capsuleCollider;
19
20         //-----
21         void Awake()
22         {
23             capsuleCollider = GetComponent<CapsuleCollider>();
24         }
25
26         //-----
27         void FixedUpdate()
28         {
29             float distanceFromFloor = Vector3.Dot( head.localPosition, Vector3.
30                 up );
31             capsuleCollider.height = Mathf.Max( capsuleCollider.radius,
32                 distanceFromFloor );
33             transform.localPosition = head.localPosition - 0.5f *
34                 distanceFromFloor * Vector3.up;
35         }
36     }
37 }

```

9.5.2. ControllerButtonHints.cs

```

1 //===== Copyright (c) Valve Corporation, All rights reserved.
2 //=====
3 //
4 // Purpose: Displays text and button hints on the controllers
5 //
6 //=====
7
8 using UnityEngine;
9 using System.Collections;
10 using System.Collections.Generic;
11 using UnityEngine.UI;
12 using System.Text;

```

```

12
13 namespace Valve.VR.InteractionSystem
14 {
15     //
16     -----
17
18     public class ControllerButtonHints : MonoBehaviour
19     {
20         public Material controllerMaterial;
21         public Color flashColor = new Color( 1.0f, 0.557f, 0.0f );
22         public GameObject textHintPrefab;
23
24         public SteamVR_Action_Vibration hapticFlash = SteamVR_Input.
25             GetAction<SteamVR_Action_Vibration>("Haptic");
26
27         [Header( "Debug" )]
28         public bool debugHints = false;
29
30         private SteamVR_RenderModel renderModel;
31         private Player player;
32
33         private List<MeshRenderer> renderers = new List<MeshRenderer>();
34         private List<MeshRenderer> flashingRenderers = new List<MeshRenderer>
35             >();
36         private float startTime;
37         private float tickCount;
38
39         private enum OffsetType
40         {
41             Up,
42             Right,
43             Forward,
44             Back
45         }
46
47         //Info for each of the buttons
48         private class ActionHintInfo
49         {
50             public string componentName;
51             public List<MeshRenderer> renderers;
52             public Transform localTransform;
53
54             //Text hint
55             public GameObject textHintObject;
56             public Transform textStartAnchor;
57             public Transform textEndAnchor;
58             public Vector3 textEndOffsetDir;
59             public Transform canvasOffset;
60
61             public Text text;
62             public TextMesh textMesh;
63             public Canvas textCanvas;
64             public LineRenderer line;
65         }

```

```

62     public float distanceFromCenter;
63     public bool textHintActive = false;
64 }
65
66 private Dictionary<ISteamVR_Action_In_Source, ActionHintInfo>
67     actionHintInfos;
68 private Transform textHintParent;
69
70 private int colorID;
71
72 public bool initialized { get; private set; }
73 private Vector3 centerPosition = Vector3.zero;
74
75 SteamVR_Events.Action renderModelLoadedAction;
76
77     protected SteamVR_Input_Sources inputSource;
78
79     //-----
80     void Awake()
81     {
82         renderModelLoadedAction = SteamVR_Events.RenderModelLoadedAction(
83             OnRenderModelLoaded );
84         colorID = Shader.PropertyToID( "_Color" );
85     }
86
87     //-----
88     void Start()
89     {
90         player = Player.instance;
91     }
92
93     //-----
94     private void HintDebugLog( string msg )
95     {
96         if ( debugHints )
97         {
98             Debug.Log("<b>[SteamVR Interaction]</b> Hints: " + msg );
99         }
100     }
101
102
103     //-----
104     void OnEnable()
105     {
106         renderModelLoadedAction.enabled = true;
107     }
108
109
110     //-----
111     void OnDisable()
112     {
113         renderModelLoadedAction.enabled = false;

```

```

114     Clear();
115 }
116
117
118 //-----
119 private void OnParentHandInputFocusLost()
120 {
121     //Hide all the hints when the controller is no longer the primary
        attached object
122     HideAllButtonHints();
123     HideAllText();
124 }
125
126
127     public virtual void SetInputSource(SteamVR_Input_Sources
        newInputSource)
128     {
129         inputSource = newInputSource;
130         if (renderModel != null)
131             renderModel.SetInputSource(newInputSource);
132     }
133
134     //-----
135     // Gets called when the hand has been initialized and a render
        model has been set
136     //-----
137     private void OnHandInitialized(int deviceIndex)
138     {
139         //Create a new render model for the controller hints
140         renderModel = new GameObject( "SteamVR_RenderModel" ).AddComponent<
            SteamVR_RenderModel>();
141         renderModel.transform.parent = transform;
142         renderModel.transform.localPosition = Vector3.zero;
143         renderModel.transform.localRotation = Quaternion.identity;
144         renderModel.transform.localScale = Vector3.one;
145
146         renderModel.SetInputSource(inputSource);
147         renderModel.SetDeviceIndex(deviceIndex);
148
149         if ( !initialized )
150         {
151             //The controller hint render model needs to be active to get
                accurate transforms for all the individual components
152             renderModel.gameObject.SetActive( true );
153         }
154     }
155
156
157     private Dictionary<string, Transform> componentTransformMap = new
        Dictionary<string, Transform>();
158
159     //-----
160     void OnRenderModelLoaded(SteamVR_RenderModel renderModel, bool
        success)

```

```

161     {
162         //Only initialize when the render model for the controller
            hints has been loaded
163         if (renderModel == this.renderModel)
164         {
165             //Debug.Log("<b>[SteamVR Interaction]</b>
                OnRenderModelLoaded: " + this.renderModel.
                    renderModelName);
166             if (initialized)
167             {
168                 Destroy(textHintParent.gameObject);
169                 componentTransformMap.Clear();
170                 flashingRenderers.Clear();
171             }
172
173             renderModel.SetMeshRendererState(false);
174
175             StartCoroutine(DoInitialize(renderModel));
176         }
177     }
178     private IEnumerator DoInitialize(SteamVR_RenderModel renderModel)
179     {
180         while (renderModel.initializedAttachPoints == false)
181             yield return null;
182
183         textHintParent = new GameObject("Text Hints").transform;
184         textHintParent.SetParent(this.transform);
185         textHintParent.localPosition = Vector3.zero;
186         textHintParent.localRotation = Quaternion.identity;
187         textHintParent.localScale = Vector3.one;
188
189         //Get the button mask for each component of the render model
190
191         var renderModels = OpenVR.RenderModels;
192         if (renderModels != null)
193         {
194             string renderModelDebug = "";
195
196             if (debugHints)
197                 renderModelDebug = "Components for render model " +
                    renderModel.index;
198
199             for (int childIndex = 0; childIndex < renderModel.
                transform.childCount; childIndex++)
200             {
201                 Transform child = renderModel.transform.GetChild(
                    childIndex);
202
203                 if (componentTransformMap.ContainsKey(child.name))
204                 {
205                     if (debugHints)
206                         renderModelDebug += "\n\t!    Child component
                            already exists with name: " + child.name
                                ;

```

```

207         }
208         else
209             componentTransformMap.Add(child.name, child);
210
211         if (debugHints)
212             renderModelDebug += "\n\t" + child.name + ".";
213     }
214
215     //Uncomment to show the button mask for each component of
216     //the render model
217     HintDebugLog(renderModelDebug);
218 }
219
220 actionHintInfos = new Dictionary<ISteamVR_Action_In_Source,
221     ActionHintInfo>();
222
223 for (int actionIndex = 0; actionIndex < SteamVR_Input.
224     actionsNonPoseNonSkeletonIn.Length; actionIndex++)
225 {
226     ISteamVR_Action_In action = SteamVR_Input.
227         actionsNonPoseNonSkeletonIn[actionIndex];
228
229     if (action.GetActive(inputSource))
230         CreateAndAddButtonInfo(action, inputSource);
231 }
232
233 ComputeTextEndTransforms();
234
235 initialized = true;
236
237 //Set the controller hints render model to not active
238 renderModel.SetMeshRendererState(true);
239 renderModel.gameObject.SetActive(false);
240 }
241
242 //-----
243 private void CreateAndAddButtonInfo(ISteamVR_Action_In action,
244     SteamVR_Input_Sources inputSource)
245 {
246     Transform buttonTransform = null;
247     List<MeshRenderer> buttonRenderers = new List<MeshRenderer>();
248
249     StringBuilder buttonDebug = new StringBuilder();
250     buttonDebug.Append("Looking for action: ");
251
252     buttonDebug.AppendLine(action.GetShortName());
253
254     buttonDebug.Append("Action localized origin: ");
255     buttonDebug.AppendLine(action.GetLocalizedOrigin(inputSource)
256         );
257
258     string actionComponentName = action.
259         GetRenderModelComponentName(inputSource);

```

```

254         if (componentTransformMap.ContainsKey(actionComponentName))
255         {
256             buttonDebug.AppendLine(string.Format("Found component:
257                 {0} for {1}", actionComponentName, action.
                    GetShortName()));
258             Transform componentTransform = componentTransformMap[
                actionComponentName];
259
260             buttonTransform = componentTransform;
261
262             buttonDebug.AppendLine(string.Format("Found
                componentTransform: {0}. buttonTransform: {1}",
                componentTransform, buttonTransform));
263
264             buttonRenderers.AddRange(componentTransform.
                GetComponentsInChildren<MeshRenderer>());
265         }
266         else
267         {
268             buttonDebug.AppendLine(string.Format("Can't find
                component transform for action: {0}. Component name:
                \"{1}\"", action.GetShortName(), actionComponentName)
                );
269         }
270
271         buttonDebug.AppendLine(string.Format("Found {0} renderers for
                {1}", buttonRenderers.Count, action.GetShortName()));
272
273         foreach ( MeshRenderer renderer in buttonRenderers )
274         {
275             buttonDebug.Append("\t");
276             buttonDebug.AppendLine(renderer.name);
277         }
278
279         HintDebugLog( buttonDebug.ToString() );
280
281         if ( buttonTransform == null )
282         {
283             HintDebugLog( "Couldn't find buttonTransform for " + action.
                GetShortName());
284             return;
285         }
286
287         ActionHintInfo hintInfo = new ActionHintInfo();
288         actionHintInfos.Add( action, hintInfo );
289
290         hintInfo.componentName = buttonTransform.name;
291         hintInfo.renderers = buttonRenderers;
292
293         //Get the local transform for the button
294         for (int childIndex = 0; childIndex < buttonTransform.
            childCount; childIndex++)
295         {

```

```

296         Transform child = buttonTransform.GetChild(childIndex);
297         if (child.name == SteamVR_RenderModel.
            k_localTransformName)
            hintInfo.localTransform = child;
298     }
299
300     OffsetType offsetType = OffsetType.Right;
301
302     /*
303     switch ( buttonID )
304 {
305     case EVRButtonId.k_EButton_SteamVR_Trigger:
306     {
307         offsetType = OffsetType.Right;
308     }
309     break;
310     case EVRButtonId.k_EButton_ApplicationMenu:
311     {
312         offsetType = OffsetType.Right;
313     }
314     break;
315     case EVRButtonId.k_EButton_System:
316     {
317         offsetType = OffsetType.Right;
318     }
319     break;
320     case Valve.VR.EVRButtonId.k_EButton_Grip:
321     {
322         offsetType = OffsetType.Forward;
323     }
324     break;
325     case Valve.VR.EVRButtonId.k_EButton_SteamVR_Touchpad:
326     {
327         offsetType = OffsetType.Up;
328     }
329     break;
330 }
331
332     */
333
334     //Offset for the text end transform
335     switch ( offsetType )
336 {
337     case OffsetType.Forward:
338         hintInfo.textEndOffsetDir = hintInfo.localTransform.forward;
339         break;
340     case OffsetType.Back:
341         hintInfo.textEndOffsetDir = -hintInfo.localTransform.forward;
342         break;
343     case OffsetType.Right:
344         hintInfo.textEndOffsetDir = hintInfo.localTransform.right;
345         break;
346     case OffsetType.Up:
347         hintInfo.textEndOffsetDir = hintInfo.localTransform.up;
348         break;

```



```

349     }
350
351     //Create the text hint object
352     Vector3 hintStartPos = hintInfo.localTransform.position + (
353         hintInfo.localTransform.forward * 0.01f );
354     hintInfo.textHintObject = GameObject.Instantiate( textHintPrefab,
355         hintStartPos, Quaternion.identity ) as GameObject;
356     hintInfo.textHintObject.name = "Hint_" + hintInfo.componentName + "_Start";
357     hintInfo.textHintObject.transform.SetParent( textHintParent );
358     hintInfo.textHintObject.layer = gameObject.layer;
359     hintInfo.textHintObject.tag = gameObject.tag;
360
361     //Get all the relevant child objects
362     hintInfo.textStartAnchor = hintInfo.textHintObject.transform.Find( "Start" );
363     hintInfo.textEndAnchor = hintInfo.textHintObject.transform.Find( "End" );
364     hintInfo.canvasOffset = hintInfo.textHintObject.transform.Find( "CanvasOffset" );
365     hintInfo.line = hintInfo.textHintObject.transform.Find( "Line" ).
366         GetComponent<LineRenderer>();
367     hintInfo.textCanvas = hintInfo.textHintObject.
368         GetComponentInChildren<Canvas>();
369     hintInfo.text = hintInfo.textCanvas.GetComponentInChildren<Text>();
370     hintInfo.textMesh = hintInfo.textCanvas.GetComponentInChildren<
371         TextMesh>();
372
373     hintInfo.textHintObject.SetActive( false );
374
375     hintInfo.textStartAnchor.position = hintStartPos;
376
377     if ( hintInfo.text != null )
378     {
379         hintInfo.text.text = hintInfo.componentName;
380     }
381
382     if ( hintInfo.textMesh != null )
383     {
384         hintInfo.textMesh.text = hintInfo.componentName;
385     }
386
387     centerPosition += hintInfo.textStartAnchor.position;
388
389     // Scale hint components to match player size
390     hintInfo.textCanvas.transform.localScale = Vector3.Scale( hintInfo.
391         textCanvas.transform.localScale, player.transform.localScale );
392     hintInfo.textStartAnchor.transform.localScale = Vector3.Scale(
393         hintInfo.textStartAnchor.transform.localScale, player.transform.
394         localScale );
395     hintInfo.textEndAnchor.transform.localScale = Vector3.Scale(
396         hintInfo.textEndAnchor.transform.localScale, player.transform.
397         localScale );

```

```

388         hintInfo.line.transform.localScale = Vector3.Scale( hintInfo.line.
389             transform.localScale, player.transform.localScale );
390     }
391
392     //-----
393     private void ComputeTextEndTransforms()
394     {
395         //This is done as a separate step after all the ButtonHintInfos
396         //have been initialized
397         //to make the text hints fan out appropriately based on the button's
398         //position on the controller.
399
400         centerPosition /= actionHintInfos.Count;
401         float maxDistanceFromCenter = 0.0f;
402
403         foreach ( var hintInfo in actionHintInfos )
404         {
405             hintInfo.Value.distanceFromCenter = Vector3.Distance( hintInfo.
406                 Value.textStartAnchor.position, centerPosition );
407
408             if ( hintInfo.Value.distanceFromCenter > maxDistanceFromCenter )
409             {
410                 maxDistanceFromCenter = hintInfo.Value.distanceFromCenter;
411             }
412         }
413
414         foreach ( var hintInfo in actionHintInfos )
415         {
416             Vector3 centerToButton = hintInfo.Value.textStartAnchor.position
417                 - centerPosition;
418             centerToButton.Normalize();
419
420             centerToButton = Vector3.Project( centerToButton, renderModel.
421                 transform.forward );
422
423             //Spread out the text end positions based on the distance from
424             //the center
425             float t = hintInfo.Value.distanceFromCenter /
426                 maxDistanceFromCenter;
427             float scale = hintInfo.Value.distanceFromCenter * Mathf.Pow( 2,
428                 10 * ( t - 1.0f ) ) * 20.0f;
429
430             //Flip the direction of the end pos based on which hand this is
431             float endPosOffset = 0.1f;
432
433             Vector3 hintEndPos = hintInfo.Value.textStartAnchor.position + (
434                 hintInfo.Value.textEndOffsetDir * endPosOffset ) + (
435                 centerToButton * scale * 0.1f );
436
437             if ( SteamVR_Utils.IsValid(hintEndPos))
438             {
439                 hintInfo.Value.textEndAnchor.position = hintEndPos;
440             }
441         }
442     }

```

```

431         hintInfo.Value.canvasOffset.position = hintEndPos;
432     }
433     else
434     {
435         Debug.LogWarning("<b>[SteamVR Interaction]</b>
            Invalid end position for: " + hintInfo.Value.
            textStartAnchor.name, hintInfo.Value.
            textStartAnchor.gameObject);
436     }
437     hintInfo.Value.canvasOffset.localRotation = Quaternion.identity;
438 }
439 }
440
441
442 //-----
443 private void ShowButtonHint( params ISteamVR_Action_In_Source[]
    actions )
444 {
445     renderModel.gameObject.SetActive( true );
446
447     renderModel.GetComponentsInChildren<MeshRenderer>( renderers );
448     for ( int i = 0; i < renderers.Count; i++ )
449     {
450         Texture mainTexture = renderers[i].material.mainTexture;
451         renderers[i].sharedMaterial = controllerMaterial;
452         renderers[i].material.mainTexture = mainTexture;
453
454         // This is to poke unity into setting the correct render queue
            for the model
455         renderers[i].material.renderQueue = controllerMaterial.shader.
            renderQueue;
456     }
457
458     for ( int i = 0; i < actions.Length; i++ )
459     {
460         if ( actionHintInfos.ContainsKey( actions[i] ) )
461         {
462             ActionHintInfo hintInfo = actionHintInfos[actions[i]];
463             foreach ( MeshRenderer renderer in hintInfo.renderers )
464             {
465                 if ( !flashingRenderers.Contains( renderer ) )
466                 {
467                     flashingRenderers.Add( renderer );
468                 }
469             }
470         }
471     }
472
473     startTime = Time.realtimeSinceStartup;
474     tickCount = 0.0f;
475 }
476
477
478 //-----

```

```

479 private void HideAllButtonHints()
480 {
481     Clear();
482
483     if (renderModel != null && renderModel.gameObject != null)
484         renderModel.gameObject.SetActive( false );
485 }
486
487
488 //-----
489 private void HideButtonHint( params ISteamVR_Action_In_Source[]
490     actions )
491 {
492     Color baseColor = controllerMaterial.GetColor( colorID );
493     for ( int i = 0; i < actions.Length; i++ )
494     {
495         if ( actionHintInfos.ContainsKey(actions[i] ) )
496         {
497             ActionHintInfo hintInfo = actionHintInfos[actions[i]];
498             foreach ( MeshRenderer renderer in hintInfo.renderers )
499             {
500                 renderer.material.color = baseColor;
501                 flashingRenderers.Remove( renderer );
502             }
503         }
504     }
505
506     if ( flashingRenderers.Count == 0 )
507     {
508         renderModel.gameObject.SetActive( false );
509     }
510 }
511
512
513
514 //-----
515 private bool IsButtonHintActive(ISteamVR_Action_In_Source action )
516 {
517     if ( actionHintInfos.ContainsKey(action) )
518     {
519         ActionHintInfo hintInfo = actionHintInfos[action];
520         foreach ( MeshRenderer buttonRenderer in hintInfo.renderers )
521         {
522             if ( flashingRenderers.Contains( buttonRenderer ) )
523             {
524                 return true;
525             }
526         }
527     }
528
529     return false;
530 }
531

```

```

532
533 //-----
534 private IEnumerator TestButtonHints()
535 {
536     while ( true )
537     {
538         for (int actionIndex = 0; actionIndex < SteamVR_Input.
539             actionsNonPoseNonSkeletonIn.Length; actionIndex++)
540         {
541             ISteamVR_Action_In action = SteamVR_Input.
542                 actionsNonPoseNonSkeletonIn[actionIndex];
543             if (action.GetActive(inputSource))
544             {
545                 ShowButtonHint(action);
546                 yield return new WaitForSeconds(1.0f);
547             }
548             yield return null;
549         }
550     }
551
552 //-----
553 private IEnumerator TestTextHints()
554 {
555     while ( true )
556     {
557         for (int actionIndex = 0; actionIndex < SteamVR_Input.
558             actionsNonPoseNonSkeletonIn.Length; actionIndex++)
559         {
560             ISteamVR_Action_In action = SteamVR_Input.
561                 actionsNonPoseNonSkeletonIn[actionIndex];
562             if (action.GetActive(inputSource))
563             {
564                 ShowText(action, action.GetShortName());
565                 yield return new WaitForSeconds(3.0f);
566             }
567             yield return null;
568         }
569         HideAllText();
570         yield return new WaitForSeconds( 3.0f );
571     }
572
573 //-----
574 void Update()
575 {
576     if ( renderModel != null && renderModel.gameObject.
577         activeInHierarchy && flashingRenderers.Count > 0 )
578     {
579         Color baseColor = controllerMaterial.GetColor( colorID );
580

```

```

581     float flash = ( Time.realtimeSinceStartup - startTime ) * Mathf.
        PI * 2.0f;
582     flash = Mathf.Cos( flash );
583     flash = Util.RemapNumberClamped( flash, -1.0f, 1.0f, 0.0f, 1.0f )
        ;
584
585     float ticks = ( Time.realtimeSinceStartup - startTime );
586     if ( ticks - tickCount > 1.0f )
587     {
588         tickCount += 1.0f;
589         hapticFlash.Execute(0, 0.005f, 0.005f, 1, inputSource
            );
590     }
591
592     for ( int i = 0; i < flashingRenderers.Count; i++ )
593     {
594         Renderer r = flashingRenderers[i];
595         r.material.SetColor( colorID, Color.Lerp( baseColor, flashColor
            , flash ) );
596     }
597
598     if ( initialized )
599     {
600         foreach ( var hintInfo in actionHintInfos )
601         {
602             if ( hintInfo.Value.textHintActive )
603             {
604                 UpdateTextHint( hintInfo.Value );
605             }
606         }
607     }
608 }
609 }
610
611 //-----
612 private void UpdateTextHint( ActionHintInfo hintInfo )
613 {
614     Transform playerTransform = player.hmdTransform;
615     Vector3 vDir = playerTransform.position - hintInfo.canvasOffset.
        position;
616
617     Quaternion standardLookat = Quaternion.LookRotation( vDir, Vector3.
        up );
618     Quaternion upsideDownLookat = Quaternion.LookRotation( vDir,
        playerTransform.up );
619
620     float flInterp;
621     if ( playerTransform.forward.y > 0.0f )
622     {
623         flInterp = Util.RemapNumberClamped( playerTransform.forward.y,
            0.6f, 0.4f, 1.0f, 0.0f );
624     }
625     else
626

```

```

627     {
628         flInterp = Util.RemapNumberClamped( playerTransform.forward.y,
        -0.8f, -0.6f, 1.0f, 0.0f );
629     }
630
631     hintInfo.canvasOffset.rotation = Quaternion.Slerp( standardLookat,
        upsideDownLookat, flInterp );
632
633     Transform lineTransform = hintInfo.line.transform;
634
635     hintInfo.line.useWorldSpace = false;
636     hintInfo.line.SetPosition( 0, lineTransform.InverseTransformPoint(
        hintInfo.textStartAnchor.position ) );
637     hintInfo.line.SetPosition( 1, lineTransform.InverseTransformPoint(
        hintInfo.textEndAnchor.position ) );
638 }
639
640
641 //-----
642 private void Clear()
643 {
644     renderers.Clear();
645     flashingRenderers.Clear();
646 }
647
648
649 //-----
650 private void ShowText(ISteamVR_Action_In_Source action, string text,
        bool highlightButton = true )
651 {
652     if ( actionHintInfos.ContainsKey(action) )
653     {
654         ActionHintInfo hintInfo = actionHintInfos[action];
655         hintInfo.textHintObject.SetActive( true );
656         hintInfo.textHintActive = true;
657
658         if ( hintInfo.text != null )
659         {
660             hintInfo.text.text = text;
661         }
662
663         if ( hintInfo.textMesh != null )
664         {
665             hintInfo.textMesh.text = text;
666         }
667
668         UpdateTextHint( hintInfo );
669
670         if ( highlightButton )
671         {
672             ShowButtonHint(action);
673         }
674
675         renderModel.gameObject.SetActive( true );

```

```

676     }
677 }
678
679
680 //-----
681 private void HideText(ISteamVR_Action_In_Source action)
682 {
683     if ( actionHintInfos.ContainsKey(action) )
684     {
685         ActionHintInfo hintInfo = actionHintInfos[action];
686         hintInfo.textHintObject.SetActive( false );
687         hintInfo.textHintActive = false;
688
689         HideButtonHint(action);
690     }
691 }
692
693
694 //-----
695 private void HideAllText()
696 {
697     if (actionHintInfos != null)
698     {
699
700         foreach (var hintInfo in actionHintInfos)
701         {
702             hintInfo.Value.textHintObject.SetActive(false);
703             hintInfo.Value.textHintActive = false;
704         }
705
706         HideAllButtonHints();
707     }
708 }
709
710
711 //-----
712 private string GetActiveHintText(ISteamVR_Action_In_Source action )
713 {
714     if ( actionHintInfos.ContainsKey(action) )
715     {
716         ActionHintInfo hintInfo = actionHintInfos[action];
717         if ( hintInfo.textHintActive )
718         {
719             return hintInfo.text.text;
720         }
721     }
722     return string.Empty;
723 }
724
725
726 //-----
727 // These are the static functions which are used to show/hide the
728 // hints
729 //-----

```



```

729
730 //-----
731 private static ControllerButtonHints GetControllerButtonHints( Hand
    hand )
732 {
733     if ( hand != null )
734     {
735         ControllerButtonHints hints = hand.GetComponentInChildren<
            ControllerButtonHints>();
736         if ( hints != null && hints.initialized )
737         {
738             return hints;
739         }
740     }
741
742     return null;
743 }
744
745 //-----
746 public static void ShowButtonHint( Hand hand, params
    ISteamVR_Action_In_Source[] actions )
747 {
748     ControllerButtonHints hints = GetControllerButtonHints( hand );
749     if ( hints != null )
750     {
751         hints.ShowButtonHint( actions );
752     }
753 }
754
755 //-----
756 public static void HideButtonHint( Hand hand, params
    ISteamVR_Action_In_Source[] actions )
757 {
758     ControllerButtonHints hints = GetControllerButtonHints( hand );
759     if ( hints != null )
760     {
761         hints.HideButtonHint( actions );
762     }
763 }
764
765 //-----
766 public static void HideAllButtonHints( Hand hand )
767 {
768     ControllerButtonHints hints = GetControllerButtonHints( hand );
769     if ( hints != null )
770     {
771         hints.HideAllButtonHints();
772     }
773 }
774
775
776
777
778

```

```

779 //-----
780 public static bool IsButtonHintActive( Hand hand,
781                                     ISteamVR_Action_In_Source action )
782 {
783     ControllerButtonHints hints = GetControllerButtonHints( hand );
784     if ( hints != null )
785     {
786         return hints.IsButtonHintActive(action);
787     }
788     return false;
789 }
790
791 //-----
792 public static void ShowTextHint( Hand hand, ISteamVR_Action_In_Source
793                                 action, string text, bool highlightButton = true )
794 {
795     ControllerButtonHints hints = GetControllerButtonHints( hand
796                                                             );
797     if ( hints != null )
798     {
799         hints.ShowText(action, text, highlightButton );
800     }
801 }
802
803 //-----
804 public static void HideTextHint( Hand hand, ISteamVR_Action_In_Source
805                                 action)
806 {
807     ControllerButtonHints hints = GetControllerButtonHints( hand );
808     if ( hints != null )
809     {
810         hints.HideText(action);
811     }
812 }
813
814 //-----
815 public static void HideAllTextHints( Hand hand )
816 {
817     ControllerButtonHints hints = GetControllerButtonHints( hand );
818     if ( hints != null )
819     {
820         hints.HideAllText();
821     }
822 }
823
824 //-----
825 public static string GetActiveHintText( Hand hand,
826                                       ISteamVR_Action_In_Source action)
827 {

```

```

828         ControllerButtonHints hints = GetControllerButtonHints( hand );
829         if ( hints != null )
830         {
831             return hints.GetActiveHintText(action);
832         }
833
834         return string.Empty;
835     }
836 }
837 }

```

9.5.3. ControllerHoverHighlight.cs

```

1  //===== Copyright (c) Valve Corporation, All rights reserved.
   //=====
2  //
3  // Purpose: Highlights the controller when hovering over interactables
4  //
5  //
   =====
6
7  using UnityEngine;
8  using System.Collections;
9
10 namespace Valve.VR.InteractionSystem
11 {
12     //
   -----
13
14     public class ControllerHoverHighlight : MonoBehaviour
15     {
16         public Material highLightMaterial;
17         public bool fireHapticsOnHightlight = true;
18
19         protected Hand hand;
20
21         protected RenderModel renderModel;
22
23         protected SteamVR_Events.Action renderModelLoadedAction;
24
25         protected void Awake()
26         {
27             hand = GetComponentInParent<Hand>();
28         }
29
30         protected void OnHandInitialized(int deviceIndex)
31         {
32             GameObject renderModelGameObject = GameObject.Instantiate(
33                 hand.renderModelPrefab);
34             renderModelGameObject.transform.parent = this.transform;
35             renderModelGameObject.transform.localPosition = Vector3.zero;

```

```

34         renderModelGameObject.transform.localRotation = Quaternion.
            identity;
35         renderModelGameObject.transform.localScale = hand.
            renderModelPrefab.transform.localScale;
36
37
38         renderModel = renderModelGameObject.GetComponent<RenderModel>();
39
40         renderModel.SetInputSource(hand.handType);
41         renderModel.OnHandInitialized(deviceIndex);
42         renderModel.SetMaterial(highLightMaterial);
43
44         hand.SetHoverRenderModel(renderModel);
45         renderModel.onControllerLoaded +=
            RenderModel_onControllerLoaded;
46         renderModel.Hide();
47     }
48
49     private void RenderModel_onControllerLoaded()
50     {
51         renderModel.Hide();
52     }
53
54
55     //-----
56     protected void OnParentHandHoverBegin(Interactable other)
57     {
58         if (!this.isActiveAndEnabled)
59         {
60             return;
61         }
62
63         if (other.transform.parent != transform.parent)
64         {
65             ShowHighlight();
66         }
67     }
68
69
70     //-----
71     private void OnParentHandHoverEnd(Interactable other)
72     {
73         HideHighlight();
74     }
75
76
77     //-----
78     private void OnParentHandInputFocusAcquired()
79     {
80         if (!this.isActiveAndEnabled)
81         {
82             return;
83         }

```

```
84
85         if (hand.hoveringInteractable && hand.hoveringInteractable.
            transform.parent != transform.parent)
86         {
87             ShowHighlight();
88         }
89     }
90
91
92     //-----
93     private void OnParentHandInputFocusLost()
94     {
95         HideHighlight();
96     }
97
98
99     //-----
100     public void ShowHighlight()
101     {
102         if (renderModel == null)
103         {
104             return;
105         }
106
107         if (fireHapticsOnHightlight)
108         {
109             hand.TriggerHapticPulse(500);
110         }
111
112         renderModel.Show();
113     }
114
115
116     //-----
117     public void HideHighlight()
118     {
119         if (renderModel == null)
120         {
121             return;
122         }
123
124         if (fireHapticsOnHightlight)
125         {
126             hand.TriggerHapticPulse(300);
127         }
128
129         renderModel.Hide();
130     }
131 }
132 }
```

9.5.4. Hand.cs

```

1 //===== Copyright (c) Valve Corporation, All rights reserved.
  =====
2 //
3 // Purpose: The hands used by the player in the vr interaction system
4 //
5 //
  =====
6
7 using UnityEngine;
8 using System;
9 using System.Collections;
10 using System.Collections.Generic;
11 using System.Collections.ObjectModel;
12 using UnityEngine.Events;
13 using System.Threading;
14
15 namespace Valve.VR.InteractionSystem
16 {
17     //
18     -----
19
20     // Links with an appropriate SteamVR controller and facilitates
21     // interactions with objects in the virtual world.
22     //
23     -----
24
25     public class Hand : MonoBehaviour
26     {
27         // The flags used to determine how an object is attached to the
28         // hand.
29         [Flags]
30         public enum AttachmentFlags
31         {
32             SnapOnAttach = 1 << 0, // The object should snap to the
33             position of the specified attachment point on the hand.
34             DetachOthers = 1 << 1, // Other objects attached to this hand
35             will be detached.
36             DetachFromOtherHand = 1 << 2, // This object will be detached
37             from the other hand.
38             ParentToHand = 1 << 3, // The object will be parented to the
39             hand.
40             VelocityMovement = 1 << 4, // The object will attempt to move
41             to match the position and rotation of the hand.
42             TurnOnKinematic = 1 << 5, // The object will not respond to
43             external physics.
44             TurnOffGravity = 1 << 6, // The object will not respond to
45             external physics.
46             AllowSidegrade = 1 << 7, // The object is able to switch from
47             a pinch grab to a grip grab. Decreases likelihood of a
48             good throw but also decreases likelihood of accidental
49             drop
50         };
51     };

```

```

36
37     public const AttachmentFlags defaultAttachmentFlags =
38         AttachmentFlags.ParentToHand |
39
40         AttachmentFlags
41             .DetachOthers
42             |
43         AttachmentFlags
44             .DetachFromOtherHand
45             |
46         AttachmentFlags
47             .TurnOnKinematic
48             |
49         AttachmentFlags
50             .SnapOnAttach
51             ;
52
53     public Hand otherHand;
54     public SteamVR_Input_Sources handType;
55
56     public SteamVR_Behaviour_Pose trackedObject;
57
58     public SteamVR_Action_Boolean grabPinchAction = SteamVR_Input.
59         GetAction<SteamVR_Action_Boolean>("GrabPinch");
60
61     public SteamVR_Action_Boolean grabGripAction = SteamVR_Input.
62         GetAction<SteamVR_Action_Boolean>("GrabGrip");
63
64     public SteamVR_Action_Vibration hapticAction = SteamVR_Input.
65         GetAction<SteamVR_Action_Vibration>("Haptic");
66
67     public SteamVR_Action_Boolean uiInteractAction = SteamVR_Input.
68         GetAction<SteamVR_Action_Boolean>("InteractUI");
69
70     public bool useHoverSphere = true;
71     public Transform hoverSphereTransform;
72     public float hoverSphereRadius = 0.05f;
73     public LayerMask hoverLayerMask = -1;
74     public float hoverUpdateInterval = 0.1f;
75
76     public bool useControllerHoverComponent = true;
77     public string controllerHoverComponent = "tip";
78     public float controllerHoverRadius = 0.075f;
79
80     public bool useFingerJointHover = true;
81     public SteamVR_Skeleton_JointIndexEnum fingerJointHover =
82         SteamVR_Skeleton_JointIndexEnum.indexTip;
83     public float fingerJointHoverRadius = 0.025f;
84
85     [Tooltip("A transform on the hand to center attached objects on")]
86     ]

```

```

71     public Transform objectAttachmentPoint;
72
73     public Camera noSteamVRFallbackCamera;
74     public float noSteamVRFallbackMaxDistanceNoItem = 10.0f;
75     public float noSteamVRFallbackMaxDistanceWithItem = 0.5f;
76     private float noSteamVRFallbackInteractorDistance = -1.0f;
77
78     public GameObject renderModelPrefab;
79     protected List<RenderModel> renderModels = new List<RenderModel>
80         >();
81     protected RenderModel mainRenderModel;
82     protected RenderModel hoverhighlightRenderModel;
83
84     public bool showDebugText = false;
85     public bool spewDebugText = false;
86     public bool showDebugInteractables = false;
87
88     public struct AttachedObject
89     {
90         public GameObject attachedObject;
91         public Interactable interactable;
92         public Rigidbody attachedRigidbody;
93         public CollisionDetectionMode collisionDetectionMode;
94         public bool attachedRigidbodyWasKinematic;
95         public bool attachedRigidbodyUsedGravity;
96         public GameObject originalParent;
97         public bool isParentedToHand;
98         public GrabTypes grabbedWithType;
99         public AttachmentFlags attachmentFlags;
100        public Vector3 initialPositionalOffset;
101        public Quaternion initialRotationalOffset;
102        public Transform attachedOffsetTransform;
103        public Transform handAttachmentPointTransform;
104        public Vector3 easeSourcePosition;
105        public Quaternion easeSourceRotation;
106        public float attachTime;
107
108        public bool HasAttachFlag(AttachmentFlags flag)
109        {
110            return (attachmentFlags & flag) == flag;
111        }
112    }
113
114    private List<AttachedObject> attachedObjects = new List<
115        AttachedObject>();
116
117    public ReadOnlyCollection<AttachedObject> AttachedObjects
118    {
119        get { return attachedObjects.AsReadOnly(); }
120    }
121
122    public bool hoverLocked { get; private set; }
123
124    private Interactable _hoveringInteractable;

```



```

123
124     private TextMesh debugText;
125     private int prevOverlappingColliders = 0;
126
127     private const int ColliderArraySize = 16;
128     private Collider[] overlappingColliders;
129
130     private Player playerInstance;
131
132     private GameObject applicationLostFocusObject;
133
134     private SteamVR_Events.Action inputFocusAction;
135
136     public bool isActive
137     {
138         get
139         {
140             if (trackedObject != null)
141                 return trackedObject.isActive;
142
143             return this.gameObject.activeInHierarchy;
144         }
145     }
146
147     public bool isPoseValid
148     {
149         get
150         {
151             return trackedObject.isValid;
152         }
153     }
154
155
156     //-----
157     // The Interactable object this Hand is currently hovering over
158     //-----
159     public Interactable hoveringInteractable
160     {
161         get { return _hoveringInteractable; }
162         set
163         {
164             if (_hoveringInteractable != value)
165             {
166                 if (_hoveringInteractable != null)
167                 {
168                     if (spewDebugText)
169                         HandDebugLog("HoverEnd " +
170                                     _hoveringInteractable.gameObject);
171                     _hoveringInteractable.SendMessage("OnHandHoverEnd",
172                                                         this, SendMessageOptions.
173                                                         DontRequireReceiver);
174
175                     //Note: The _hoveringInteractable can change
176                         after sending the OnHandHoverEnd message so

```

```

        we need to check it again before broadcasting
        this message
173     if (_hoveringInteractable != null)
174     {
175         this.BroadcastMessage("OnParentHandHoverEnd",
            _hoveringInteractable,
            SendMessageOptions.DontRequireReceiver);
        // let objects attached to the hand know
        that a hover has ended
176     }
177 }
178
179 _hoveringInteractable = value;
180
181 if (_hoveringInteractable != null)
182 {
183     if (spewDebugText)
184         HandDebugLog("HoverBegin " +
            _hoveringInteractable.gameObject);
185     _hoveringInteractable.SendMessage("
        OnHandHoverBegin", this, SendMessageOptions.
        DontRequireReceiver);
186
187     //Note: The _hoveringInteractable can change
        after sending the OnHandHoverBegin message so
        we need to check it again before
        broadcasting this message
188     if (_hoveringInteractable != null)
189     {
190         this.BroadcastMessage("OnParentHandHoverBegin
            ", _hoveringInteractable,
            SendMessageOptions.DontRequireReceiver);
        // let objects attached to the hand know
        that a hover has begun
191     }
192 }
193 }
194 }
195 }
196
197
198 //-----
199 // Active GameObject attached to this Hand
200 //-----
201 public GameObject currentAttachedObject
202 {
203     get
204     {
205         CleanupAttachedObjectStack();
206
207         if (attachedObjects.Count > 0)
208         {
209             return attachedObjects[attachedObjects.Count - 1].
                attachedObject;

```

```
210         }
211
212         return null;
213     }
214 }
215
216 public AttachedObject? currentAttachedObjectInfo
217 {
218     get
219     {
220         CleanUpAttachedObjectStack();
221
222         if (attachedObjects.Count > 0)
223         {
224             return attachedObjects[attachedObjects.Count - 1];
225         }
226
227         return null;
228     }
229 }
230
231 public SteamVR_Behaviour_Skeleton skeleton
232 {
233     get
234     {
235         if (mainRenderModel != null)
236             return mainRenderModel.GetSkeleton();
237
238         return null;
239     }
240 }
241
242 public void ShowController(bool permanent = false)
243 {
244     if (mainRenderModel != null)
245         mainRenderModel.SetControllerVisibility(true, permanent);
246
247     if (hoverhighlightRenderModel != null)
248         hoverhighlightRenderModel.SetControllerVisibility(true,
249             permanent);
250 }
251
252 public void HideController(bool permanent = false)
253 {
254     if (mainRenderModel != null)
255         mainRenderModel.SetControllerVisibility(false, permanent);
256     ;
257
258     if (hoverhighlightRenderModel != null)
259         hoverhighlightRenderModel.SetControllerVisibility(false,
260             permanent);
261 }
262
263 public void ShowSkeleton(bool permanent = false)
```

```

261     {
262         if (mainRenderModel != null)
263             mainRenderModel.SetHandVisibility(true, permanent);
264
265         if (hoverhighlightRenderModel != null)
266             hoverhighlightRenderModel.SetHandVisibility(true,
267                 permanent);
268     }
269
270     public void HideSkeleton(bool permanent = false)
271     {
272         if (mainRenderModel != null)
273             mainRenderModel.SetHandVisibility(false, permanent);
274
275         if (hoverhighlightRenderModel != null)
276             hoverhighlightRenderModel.SetHandVisibility(false,
277                 permanent);
278     }
279
280     public bool HasSkeleton()
281     {
282         return mainRenderModel != null && mainRenderModel.GetSkeleton
283             () != null;
284     }
285
286     public void Show()
287     {
288         SetVisibility(true);
289     }
290
291     public void Hide()
292     {
293         SetVisibility(false);
294     }
295
296     public void SetVisibility(bool visible)
297     {
298         if (mainRenderModel != null)
299             mainRenderModel.SetVisibility(visible);
300     }
301
302     public void SetSkeletonRangeOfMotion(EVRSkeletalMotionRange
303         newRangeOfMotion, float blendOverSeconds = 0.1f)
304     {
305         for (int renderModelIndex = 0; renderModelIndex <
306             renderModels.Count; renderModelIndex++)
307         {
308             renderModels[renderModelIndex].SetSkeletonRangeOfMotion(
309                 newRangeOfMotion, blendOverSeconds);
310         }
311     }
312
313     public void SetTemporarySkeletonRangeOfMotion(
314         SkeletalMotionRangeChange temporaryRangeOfMotionChange, float

```

```

308         blendOverSeconds = 0.1f)
309     {
310         for (int renderModelIndex = 0; renderModelIndex <
311             renderModels.Count; renderModelIndex++)
312         {
313             renderModels[renderModelIndex].
314                 SetTemporarySkeletonRangeOfMotion(
315                     temporaryRangeOfMotionChange, blendOverSeconds);
316         }
317     }
318
319     public void ResetTemporarySkeletonRangeOfMotion(float
320         blendOverSeconds = 0.1f)
321     {
322         for (int renderModelIndex = 0; renderModelIndex <
323             renderModels.Count; renderModelIndex++)
324         {
325             renderModels[renderModelIndex].
326                 ResetTemporarySkeletonRangeOfMotion(blendOverSeconds)
327                 ;
328         }
329     }
330
331     public void SetAnimationState(int stateValue)
332     {
333         for (int renderModelIndex = 0; renderModelIndex <
334             renderModels.Count; renderModelIndex++)
335         {
336             renderModels[renderModelIndex].SetAnimationState(
337                 stateValue);
338         }
339     }
340
341     public void StopAnimation()
342     {
343         for (int renderModelIndex = 0; renderModelIndex <
344             renderModels.Count; renderModelIndex++)
345         {
346             renderModels[renderModelIndex].StopAnimation();
347         }
348     }
349
350     //-----
351     // Attach a GameObject to this GameObject
352     //
353     // objectToAttach - The GameObject to attach
354     // flags - The flags to use for attaching the object
355     // attachmentPoint - Name of the GameObject in the hierarchy of
356     // this Hand which should act as the attachment point for this
357     // GameObject
358     //-----
359     public void AttachObject(GameObject objectToAttach, GrabTypes
360         grabbedWithType, AttachmentFlags flags =

```

```

348         defaultAttachmentFlags, Transform attachmentOffset = null)
349     {
350         AttachedObject attachedObject = new AttachedObject();
351         attachedObject.attachmentFlags = flags;
352         attachedObject.attachedOffsetTransform = attachmentOffset;
353         attachedObject.attachTime = Time.time;
354
355         if (flags == 0)
356         {
357             flags = defaultAttachmentFlags;
358         }
359
360         //Make sure top object on stack is non-null
361         CleanupAttachedObjectStack();
362
363         //Detach the object if it is already attached so that it can
364         //get re-attached at the top of the stack
365         if (ObjectIsAttached(objectToAttach))
366             DetachObject(objectToAttach);
367
368         //Detach from the other hand if requested
369         if (attachedObject.HasAttachFlag(AttachmentFlags.DetachOthers))
370         {
371             if (otherHand != null)
372                 otherHand.DetachObject(objectToAttach);
373         }
374
375         if (attachedObject.HasAttachFlag(AttachmentFlags.DetachOthers))
376         {
377             //Detach all the objects from the stack
378             while (attachedObjects.Count > 0)
379             {
380                 DetachObject(attachedObjects[0].attachedObject);
381             }
382         }
383
384         if (currentAttachedObject)
385         {
386             currentAttachedObject.SendMessage("OnHandFocusLost", this,
387                 SendMessageOptions.DontRequireReceiver);
388         }
389
390         attachedObject.attachedObject = objectToAttach;
391         attachedObject.interactable = objectToAttach.GetComponent<
392             Interactable>();
393         attachedObject.handAttachmentPointTransform = this.transform;
394
395         if (attachedObject.interactable != null)
396         {
397             if (attachedObject.interactable.attachEaseIn)
398             {

```

```

395         attachedObject.easeSourcePosition = attachedObject.
396             attachedObject.transform.position;
397         attachedObject.easeSourceRotation = attachedObject.
398             attachedObject.transform.rotation;
399         attachedObject.interactable.snapAttachEaseInCompleted
400             = false;
401     }
402
403     if (attachedObject.interactable.
404         useHandObjectAttachmentPoint)
405         attachedObject.handAttachmentPointTransform =
406             objectAttachmentPoint;
407
408     if (attachedObject.interactable.hideHandOnAttach)
409         Hide();
410
411     if (attachedObject.interactable.hideSkeletonOnAttach &&
412         mainRenderModel != null && mainRenderModel.
413         displayHandByDefault)
414         HideSkeleton();
415
416     if (attachedObject.interactable.hideControllerOnAttach &&
417         mainRenderModel != null && mainRenderModel.
418         displayControllerByDefault)
419         HideController();
420
421     if (attachedObject.interactable.handAnimationOnPickup !=
422         0)
423         SetAnimationState(attachedObject.interactable.
424             handAnimationOnPickup);
425
426     if (attachedObject.interactable.setRangeOfMotionOnPickup
427         != SkeletalMotionRangeChange.None)
428         SetTemporarySkeletonRangeOfMotion(attachedObject.
429             interactable.setRangeOfMotionOnPickup);
430
431     }
432
433     attachedObject.originalParent = objectToAttach.transform.
434         parent != null ? objectToAttach.transform.parent.
435         gameObject : null;
436
437     attachedObject.attachedRigidbody = objectToAttach.
438         GetComponent<Rigidbody>();
439     if (attachedObject.attachedRigidbody != null)
440     {
441         if (attachedObject.interactable.attachedToHand != null)
442             //already attached to another hand
443         {
444             //if it was attached to another hand, get the flags
445             from that hand
446
447             for (int attachedIndex = 0; attachedIndex <
448                 attachedObject.interactable.attachedToHand.

```

```

430         attachedObjects.Count; attachedIndex++)
431     {
432         AttachedObject attachedObjectInList =
433             attachedObject.interactable.attachedToHand.
434             attachedObjects[attachedIndex];
435         if (attachedObjectInList.interactable ==
436             attachedObject.interactable)
437         {
438             attachedObject.attachedRigidbodyWasKinematic
439                 = attachedObjectInList.
440                 attachedRigidbodyWasKinematic;
441             attachedObject.attachedRigidbodyUsedGravity =
442                 attachedObjectInList.
443                 attachedRigidbodyUsedGravity;
444             attachedObject.originalParent =
445                 attachedObjectInList.originalParent;
446         }
447     }
448     else
449     {
450         attachedObject.attachedRigidbodyWasKinematic =
451             attachedObject.attachedRigidbody.isKinematic;
452         attachedObject.attachedRigidbodyUsedGravity =
453             attachedObject.attachedRigidbody.useGravity;
454     }
455 }
456
457 attachedObject.grabbedWithType = grabbedWithType;
458
459 if (attachedObject.HasAttachFlag(AttachmentFlags.ParentToHand))
460 {
461     //Parent the object to the hand
462     objectToAttach.transform.parent = this.transform;
463     attachedObject.isParentedToHand = true;
464 }
465 else
466 {
467     attachedObject.isParentedToHand = false;
468 }
469
470 if (attachedObject.HasAttachFlag(AttachmentFlags.SnapOnAttach))
471 {
472     if (attachedObject.interactable != null && attachedObject
473         .interactable.skeletonPoser != null && HasSkeleton())
474     {
475         SteamVR_Skeleton_PoseSnapshot pose = attachedObject.
476             interactable.skeletonPoser.GetBlendedPose(
477                 skeleton);
478
479         //snap the object to the center of the attach point

```



```

467         objectToAttach.transform.position = this.transform.
           TransformPoint(pose.position);
468         objectToAttach.transform.rotation = this.transform.
           rotation * pose.rotation;
469
470         attachedObject.initialPositionalOffset =
           attachedObject.handAttachmentPointTransform.
           InverseTransformPoint(objectToAttach.transform.
           position);
471         attachedObject.initialRotationalOffset = Quaternion.
           Inverse(attachedObject.
           handAttachmentPointTransform.rotation) *
           objectToAttach.transform.rotation;
472     }
473     else
474     {
475         if (attachmentOffset != null)
476         {
477             //offset the object from the hand by the
           positional and rotational difference between
           the offset transform and the attached object
478             Quaternion rotDiff = Quaternion.Inverse(
           attachmentOffset.transform.rotation) *
           objectToAttach.transform.rotation;
479             objectToAttach.transform.rotation =
           attachedObject.handAttachmentPointTransform.
           rotation * rotDiff;
480
481             Vector3 posDiff = objectToAttach.transform.
           position - attachmentOffset.transform.
           position;
482             objectToAttach.transform.position =
           attachedObject.handAttachmentPointTransform.
           position + posDiff;
483         }
484         else
485         {
486             //snap the object to the center of the attach
           point
487             objectToAttach.transform.rotation =
           attachedObject.handAttachmentPointTransform.
           rotation;
488             objectToAttach.transform.position =
           attachedObject.handAttachmentPointTransform.
           position;
489         }
490
491         Transform followPoint = objectToAttach.transform;
492
493         attachedObject.initialPositionalOffset =
           attachedObject.handAttachmentPointTransform.
           InverseTransformPoint(followPoint.position);
494         attachedObject.initialRotationalOffset = Quaternion.
           Inverse(attachedObject.

```

```

        handAttachmentPointTransform.rotation) *
        followPoint.rotation;
495     }
496 }
497 else
498 {
499     if (attachedObject.interactable != null && attachedObject
500         .interactable.skeletonPoser != null && HasSkeleton())
501     {
502         attachedObject.initialPositionalOffset =
503             attachedObject.handAttachmentPointTransform.
504             InverseTransformPoint(objectToAttach.transform.
505             position);
506         attachedObject.initialRotationalOffset = Quaternion.
507             Inverse(attachedObject.
508             handAttachmentPointTransform.rotation) *
509             objectToAttach.transform.rotation;
510     }
511     else
512     {
513         if (attachmentOffset != null)
514         {
515             //get the initial positional and rotational
516             //offsets between the hand and the offset
517             //transform
518             Quaternion rotDiff = Quaternion.Inverse(
519                 attachmentOffset.transform.rotation) *
520                 objectToAttach.transform.rotation;
521             Quaternion targetRotation = attachedObject.
522                 handAttachmentPointTransform.rotation *
523                 rotDiff;
524             Quaternion rotationPositionBy = targetRotation *
525                 Quaternion.Inverse(objectToAttach.transform.
526                 rotation);
527             Vector3 posDiff = (rotationPositionBy *
528                 objectToAttach.transform.position) - (
529                 rotationPositionBy * attachmentOffset.
530                 transform.position);
531             attachedObject.initialPositionalOffset =
532                 attachedObject.handAttachmentPointTransform.
533                 InverseTransformPoint(attachedObject.
534                 handAttachmentPointTransform.position +
535                 posDiff);
536             attachedObject.initialRotationalOffset =
537                 Quaternion.Inverse(attachedObject.
538                 handAttachmentPointTransform.rotation) * (
539                 attachedObject.handAttachmentPointTransform.
540                 rotation * rotDiff);
541         }
542     }
543 }
544 else
545 {

```

```

520         attachedObject.initialPositionalOffset =
521             attachedObject.handAttachmentPointTransform.
522             InverseTransformPoint(objectToAttach.
523             transform.position);
524         attachedObject.initialRotationalOffset =
525             Quaternion.Inverse(attachedObject.
526             handAttachmentPointTransform.rotation) *
527             objectToAttach.transform.rotation;
528     }
529 }
530
531 if (attachedObject.HasAttachFlag(AttachmentFlags.
532     TurnOnKinematic))
533 {
534     if (attachedObject.attachedRigidbody != null)
535     {
536         attachedObject.collisionDetectionMode =
537             attachedObject.attachedRigidbody.
538             collisionDetectionMode;
539         if (attachedObject.collisionDetectionMode ==
540             CollisionDetectionMode.Continuous)
541             attachedObject.attachedRigidbody.
542             collisionDetectionMode =
543             CollisionDetectionMode.Discrete;
544
545         attachedObject.attachedRigidbody.isKinematic = true;
546     }
547 }
548
549 if (attachedObject.HasAttachFlag(AttachmentFlags.
550     TurnOffGravity))
551 {
552     if (attachedObject.attachedRigidbody != null)
553     {
554         attachedObject.attachedRigidbody.useGravity = false;
555     }
556 }
557
558 if (attachedObject.interactable != null && attachedObject.
559     interactable.attachEaseIn)
560 {
561     attachedObject.attachedObject.transform.position =
562         attachedObject.easeSourcePosition;
563     attachedObject.attachedObject.transform.rotation =
564         attachedObject.easeSourceRotation;
565 }
566
567 attachedObjects.Add(attachedObject);
568
569 UpdateHovering();

```

```

558         if (spewDebugText)
559             HandDebugLog("AttachObject " + objectToAttach);
560         objectToAttach.SendMessage("OnAttachedToHand", this,
561                                     SendMessageOptions.DontRequireReceiver);
562     }
563     public bool ObjectIsAttached(GameObject go)
564     {
565         for (int attachedIndex = 0; attachedIndex < attachedObjects.
566             Count; attachedIndex++)
567         {
568             if (attachedObjects[attachedIndex].attachedObject == go)
569                 return true;
570         }
571         return false;
572     }
573     public void ForceHoverUnlock()
574     {
575         hoverLocked = false;
576     }
577
578     //-----
579     // Detach this GameObject from the attached object stack of this
580     // Hand
581     //
582     // objectToDetach - The GameObject to detach from this Hand
583     //-----
584     public void DetachObject(GameObject objectToDetach, bool
585         restoreOriginalParent = true)
586     {
587         int index = attachedObjects.FindIndex(l => l.attachedObject
588             == objectToDetach);
589         if (index != -1)
590         {
591             if (spewDebugText)
592                 HandDebugLog("DetachObject " + objectToDetach);
593
594             GameObject prevTopObject = currentAttachedObject;
595
596             if (attachedObjects[index].interactable != null)
597             {
598                 if (attachedObjects[index].interactable.
599                     hideHandOnAttach)
600                     Show();
601
602                 if (attachedObjects[index].interactable.
603                     hideSkeletonOnAttach && mainRenderModel != null
604                     && mainRenderModel.displayHandByDefault)
605                     ShowSkeleton();

```

```

603         if (attachedObjects[index].interactable.
604             hideControllerOnAttach && mainRenderModel != null
605             && mainRenderModel.displayControllerByDefault)
606             ShowController();
607
608         if (attachedObjects[index].interactable.
609             handAnimationOnPickup != 0)
610             StopAnimation();
611
612         if (attachedObjects[index].interactable.
613             setRangeOfMotionOnPickup !=
614             SkeletalMotionRangeChange.None)
615             ResetTemporarySkeletonRangeOfMotion();
616     }
617
618     Transform parentTransform = null;
619     if (attachedObjects[index].isParentedToHand)
620     {
621         if (restoreOriginalParent && (attachedObjects[index].
622             originalParent != null))
623         {
624             parentTransform = attachedObjects[index].
625                 originalParent.transform;
626         }
627
628         if (attachedObjects[index].attachedObject != null)
629         {
630             attachedObjects[index].attachedObject.transform.
631                 parent = parentTransform;
632         }
633     }
634
635     if (attachedObjects[index].HasAttachFlag(AttachmentFlags.
636         TurnOnKinematic))
637     {
638         if (attachedObjects[index].attachedRigidbody != null)
639         {
640             attachedObjects[index].attachedRigidbody.
641                 isKinematic = attachedObjects[index].
642                 attachedRigidbodyWasKinematic;
643             attachedObjects[index].attachedRigidbody.
644                 collisionDetectionMode = attachedObjects[
645                 index].collisionDetectionMode;
646         }
647     }
648
649     if (attachedObjects[index].HasAttachFlag(AttachmentFlags.
650         TurnOffGravity))
651     {
652         if (attachedObjects[index].attachedObject != null)
653         {
654             if (attachedObjects[index].attachedRigidbody !=
655                 null)

```

```

641         attachedObjects[index].attachedRigidbody.
            useGravity = attachedObjects[index].
            attachedRigidbodyUsedGravity;
642     }
643 }
644
645 if (attachedObjects[index].interactable != null &&
    attachedObjects[index].interactable.
    handFollowTransform && HasSkeleton())
646 {
647     skeleton.transform.localPosition = Vector3.zero;
648     skeleton.transform.localRotation = Quaternion.
        identity;
649 }
650
651 if (attachedObjects[index].attachedObject != null)
652 {
653     if (attachedObjects[index].interactable == null || (
        attachedObjects[index].interactable != null &&
        attachedObjects[index].interactable.isDestroying
        == false))
654         attachedObjects[index].attachedObject.SetActive(
            true);
655
656     attachedObjects[index].attachedObject.SendMessage("
        OnDetachedFromHand", this, SendMessageOptions.
        DontRequireReceiver);
657 }
658
659 attachedObjects.RemoveAt(index);
660
661 CleanUpAttachedObjectStack();
662
663 GameObject newTopObject = currentAttachedObject;
664
665 hoverLocked = false;
666
667
668 //Give focus to the top most object on the stack if it
    changed
669 if (newTopObject != null && newTopObject != prevTopObject
    )
670 {
671     newTopObject.SetActive(true);
672     newTopObject.SendMessage("OnHandFocusAcquired", this,
        SendMessageOptions.DontRequireReceiver);
673 }
674 }
675
676 CleanUpAttachedObjectStack();
677
678 if (mainRenderModel != null)
679     mainRenderModel.MatchHandToTransform(mainRenderModel.
        transform);

```

```

680         if (hoverhighlightRenderModel != null)
681             hoverhighlightRenderModel.MatchHandToTransform(
682                 hoverhighlightRenderModel.transform);
683     }
684
685     //-----
686     // Get the world velocity of the VR Hand.
687     //-----
688     public Vector3 GetTrackedObjectVelocity(float timeOffset = 0)
689     {
690         if (trackedObject == null)
691         {
692             Vector3 velocityTarget, angularTarget;
693             GetUpdatedAttachedVelocities(currentAttachedObjectInfo.
694                 Value, out velocityTarget, out angularTarget);
695             return velocityTarget;
696         }
697
698         if (isActive)
699         {
700             if (timeOffset == 0)
701                 return Player.instance.trackingOriginTransform.
702                     TransformVector(trackedObject.GetVelocity());
703             else
704             {
705                 Vector3 velocity;
706                 Vector3 angularVelocity;
707
708                 trackedObject.GetVelocitiesAtTimeOffset(timeOffset,
709                     out velocity, out angularVelocity);
710                 return Player.instance.trackingOriginTransform.
711                     TransformVector(velocity);
712             }
713         }
714
715         return Vector3.zero;
716     }
717
718     //-----
719     // Get the world space angular velocity of the VR Hand.
720     //-----
721     public Vector3 GetTrackedObjectAngularVelocity(float timeOffset =
722         0)
723     {
724         if (trackedObject == null)
725         {
726             Vector3 velocityTarget, angularTarget;
727             GetUpdatedAttachedVelocities(currentAttachedObjectInfo.
728                 Value, out velocityTarget, out angularTarget);
729             return angularTarget;
730         }

```

```

727         if (isActive)
728         {
729             if (timeOffset == 0)
730                 return Player.instance.trackingOriginTransform.
                    TransformDirection(trackedObject.
                        GetAngularVelocity());
731             else
732             {
733                 Vector3 velocity;
734                 Vector3 angularVelocity;
735
736                 trackedObject.GetVelocitiesAtTimeOffset(timeOffset,
                    out velocity, out angularVelocity);
737                 return Player.instance.trackingOriginTransform.
                    TransformDirection(angularVelocity);
738             }
739         }
740
741         return Vector3.zero;
742     }
743
744     public void GetEstimatedPeakVelocities(out Vector3 velocity, out
        Vector3 angularVelocity)
745     {
746         trackedObject.GetEstimatedPeakVelocities(out velocity, out
            angularVelocity);
747         velocity = Player.instance.trackingOriginTransform.
            TransformVector(velocity);
748         angularVelocity = Player.instance.trackingOriginTransform.
            TransformDirection(angularVelocity);
749     }
750
751
752     //-----
753     private void CleanUpAttachedObjectStack()
754     {
755         attachedObjects.RemoveAll(l => l.attachedObject == null);
756     }
757
758
759     //-----
760     protected virtual void Awake()
761     {
762         inputFocusAction = SteamVR_Events.InputFocusAction(
            OnInputFocus);
763
764         if (hoverSphereTransform == null)
765             hoverSphereTransform = this.transform;
766
767         if (objectAttachmentPoint == null)
768             objectAttachmentPoint = this.transform;
769
770         applicationLostFocusObject = new GameObject("
            _application_lost_focus");

```



```

771         applicationLostFocusObject.transform.parent = transform;
772         applicationLostFocusObject.SetActive(false);
773
774         if (trackedObject == null)
775         {
776             trackedObject = this.gameObject.GetComponent<
777                 SteamVR_Behaviour_Pose>();
778
779             if (trackedObject != null)
780                 trackedObject.onTransformUpdatedEvent +=
781                     OnTransformUpdated;
782         }
783     }
784
785     protected virtual void OnDestroy()
786     {
787         if (trackedObject != null)
788         {
789             trackedObject.onTransformUpdatedEvent -=
790                 OnTransformUpdated;
791         }
792     }
793
794     protected virtual void OnTransformUpdated(SteamVR_Behaviour_Pose
795         updatedPose, SteamVR_Input_Sources updatedSource)
796     {
797         HandFollowUpdate();
798     }
799
800     //-----
801     protected virtual IEnumerator Start()
802     {
803         // save off player instance
804         playerInstance = Player.instance;
805         if (!playerInstance)
806         {
807             Debug.LogError("<b>[SteamVR Interaction]</b> No player
808                 instance found in Hand Start()");
809         }
810
811         // allocate array for colliders
812         overlappingColliders = new Collider[ColliderArraySize];
813
814         // We are a "no SteamVR fallback hand" if we have this camera
815         set
816         // we'll use the right mouse to look around and left mouse to
817         interact
818         // - don't need to find the device
819         if (noSteamVRFallbackCamera)
820         {
821             yield break;
822         }

```

```

817         //Debug.Log( "<b>[SteamVR Interaction]</b> Hand -
            initializing connection routine" );
818
819         while (true)
820         {
821             if (isPoseValid)
822             {
823                 InitController();
824                 break;
825             }
826
827             yield return null;
828         }
829     }
830
831
832     //-----
833     protected virtual void UpdateHovering()
834     {
835         if ((noSteamVRFallbackCamera == null) && (isActive == false))
836         {
837             return;
838         }
839
840         if (hoverLocked)
841             return;
842
843         if (applicationLostFocusObject.activeSelf)
844             return;
845
846         float closestDistance = float.MaxValue;
847         Interactable closestInteractable = null;
848
849         if (useHoverSphere)
850         {
851             float scaledHoverRadius = hoverSphereRadius * Mathf.Abs(
                SteamVR_Utils.GetLossyScale(hoverSphereTransform));
852             CheckHoveringForTransform(hoverSphereTransform.position,
                scaledHoverRadius, ref closestDistance, ref
                closestInteractable, Color.green);
853         }
854
855         if (useControllerHoverComponent && mainRenderModel != null &&
            mainRenderModel.IsControllerVisibile())
856         {
857             float scaledHoverRadius = controllerHoverRadius * Mathf.
                Abs(SteamVR_Utils.GetLossyScale(this.transform));
858             CheckHoveringForTransform(mainRenderModel.
                GetControllerPosition(controllerHoverComponent),
                scaledHoverRadius / 2f, ref closestDistance, ref
                closestInteractable, Color.blue);
859         }
860

```

```

861         if (useFingerJointHover && mainRenderModel != null &&
862             mainRenderModel.IsHandVisibile())
863         {
864             float scaledHoverRadius = fingerJointHoverRadius * Mathf.
                Abs(SteamVR_Utils.GetLossyScale(this.transform));
            CheckHoveringForTransform(mainRenderModel.GetBonePosition
                ((int)fingerJointHover), scaledHoverRadius / 2f, ref
                closestDistance, ref closestInteractable, Color.
                yellow);
865         }
866
867         // Hover on this one
868         hoveringInteractable = closestInteractable;
869     }
870
871     protected virtual bool CheckHoveringForTransform(Vector3
        hoverPosition, float hoverRadius, ref float closestDistance,
        ref Interactable closestInteractable, Color debugColor)
872     {
873         bool foundCloser = false;
874
875         // null out old vals
876         for (int i = 0; i < overlappingColliders.Length; ++i)
877         {
878             overlappingColliders[i] = null;
879         }
880
881         int numColliding = Physics.OverlapSphereNonAlloc(
            hoverPosition, hoverRadius, overlappingColliders,
            hoverLayerMask.value);
882
883         if (numColliding == ColliderArraySize)
884             Debug.LogWarning("<b>[SteamVR Interaction]</b> This hand
                is overlapping the max number of colliders: " +
                ColliderArraySize + ". Some collisions may be missed.
                Increase ColliderArraySize on Hand.cs");
885
886         // DebugVar
887         int iActualColliderCount = 0;
888
889         // Pick the closest hovering
890         for (int colliderIndex = 0; colliderIndex <
            overlappingColliders.Length; colliderIndex++)
891         {
892             Collider collider = overlappingColliders[colliderIndex];
893
894             if (collider == null)
895                 continue;
896
897             Interactable contacting = collider.GetComponentInParent<
                Interactable>();
898
899             // Yeah, it's null, skip
900             if (contacting == null)

```

```

901         continue;
902
903     // Ignore this collider for hovering
904     IgnoreHovering ignore = collider.GetComponent<
        IgnoreHovering>();
905     if (ignore != null)
906     {
907         if (ignore.onlyIgnoreHand == null || ignore.
            onlyIgnoreHand == this)
908         {
909             continue;
910         }
911     }
912
913     // Can't hover over the object if it's attached
914     bool hoveringOverAttached = false;
915     for (int attachedIndex = 0; attachedIndex <
        attachedObjects.Count; attachedIndex++)
916     {
917         if (attachedObjects[attachedIndex].attachedObject ==
            contacting.gameObject)
918         {
919             hoveringOverAttached = true;
920             break;
921         }
922     }
923     if (hoveringOverAttached)
924         continue;
925
926     // Occupied by another hand, so we can't touch it
927     if (otherHand && otherHand.hoveringInteractable ==
        contacting)
928         continue;
929
930     // Best candidate so far...
931     float distance = Vector3.Distance(contacting.transform.
        position, hoverPosition);
932     if (distance < closestDistance)
933     {
934         closestDistance = distance;
935         closestInteractable = contacting;
936         foundCloser = true;
937     }
938     iActualColliderCount++;
939 }
940
941 if (showDebugInteractables && foundCloser)
942 {
943     Debug.DrawLine(hoverPosition, closestInteractable.
        transform.position, debugColor, .05f, false);
944 }
945
946 if (iActualColliderCount > 0 && iActualColliderCount !=
    prevOverlappingColliders)

```

```

947         {
948             prevOverlappingColliders = iActualColliderCount;
949
950             if (spewDebugText)
951                 HandDebugLog("Found " + iActualColliderCount + "
952                             overlapping colliders.");
953         }
954         return foundCloser;
955     }
956
957     //-----
958     protected virtual void UpdateNoSteamVRFallback()
959     {
960         if (noSteamVRFallbackCamera)
961         {
962             Ray ray = noSteamVRFallbackCamera.ScreenPointToRay(Input.
963                 mousePosition);
964
965             if (attachedObjects.Count > 0)
966             {
967                 // Holding down the mouse:
968                 // move around a fixed distance from the camera
969                 transform.position = ray.origin +
970                     noSteamVRFallbackInteractorDistance * ray.
971                     direction;
972             }
973             else
974             {
975                 // Not holding down the mouse:
976                 // cast out a ray to see what we should mouse over
977
978                 // Don't want to hit the hand and anything underneath
979                 // it
980                 // So move it back behind the camera when we do the
981                 // raycast
982                 Vector3 oldPosition = transform.position;
983                 transform.position = noSteamVRFallbackCamera.
984                     transform.forward * (-1000.0f);
985
986                 RaycastHit raycastHit;
987                 if (Physics.Raycast(ray, out raycastHit,
988                     noSteamVRFallbackMaxDistanceNoItem))
989                 {
990                     transform.position = raycastHit.point;
991
992                     // Remember this distance in case we click and
993                     // drag the mouse
994                     noSteamVRFallbackInteractorDistance = Mathf.Min(
995                         noSteamVRFallbackMaxDistanceNoItem,
996                         raycastHit.distance);
997                 }
998             }
999             else if (noSteamVRFallbackInteractorDistance > 0.0f)

```

```

990         {
991             // Move it around at the distance we last had a
992             hit
993             transform.position = ray.origin + Mathf.Min(
994                 noSteamVRFallbackMaxDistanceNoItem,
995                 noSteamVRFallbackInteractorDistance) * ray.
996                 direction;
997         }
998     else
999     {
1000         // Didn't hit, just leave it where it was
1001         transform.position = oldPosition;
1002     }
1003 }
1004 }
1005
1006 //-----
1007 private void UpdateDebugText()
1008 {
1009     if (showDebugText)
1010     {
1011         if (debugText == null)
1012         {
1013             debugText = new GameObject("_debug_text").
1014                 AddComponent<TextMesh>();
1015             debugText.fontSize = 120;
1016             debugText.characterSize = 0.001f;
1017             debugText.transform.parent = transform;
1018
1019             debugText.transform.localRotation = Quaternion.Euler
1020                 (90.0f, 0.0f, 0.0f);
1021         }
1022
1023         if (handType == SteamVR_Input_Sources.RightHand)
1024         {
1025             debugText.transform.localPosition = new Vector3(-0.05f,
1026                 0.0f, 0.0f);
1027             debugText.alignment = TextAlignment.Right;
1028             debugText.anchor = TextAnchor.UpperRight;
1029         }
1030         else
1031         {
1032             debugText.transform.localPosition = new Vector3(0.05f,
1033                 0.0f, 0.0f);
1034             debugText.alignment = TextAlignment.Left;
1035             debugText.anchor = TextAnchor.UpperLeft;
1036         }
1037
1038         debugText.text = string.Format(
1039             "Hovering: {0}\n" +
1040             "Hover Lock: {1}\n" +
1041             "Attached: {2}\n" +

```

```

1036         "Total Attached: {3}\n" +
1037         "Type: {4}\n",
1038         (hoveringInteractable ? hoveringInteractable.
            gameObject.name : "null"),
1039         hoverLocked,
1040         (currentAttachedObject ? currentAttachedObject.name :
            "null"),
1041         attachedObjects.Count,
1042         handType.ToString());
1043     }
1044     else
1045     {
1046         if (debugText != null)
1047         {
1048             Destroy(debugText.gameObject);
1049         }
1050     }
1051 }
1052
1053
1054 //-----
1055 protected virtual void OnEnable()
1056 {
1057     inputFocusAction.enabled = true;
1058
1059     // Stagger updates between hands
1060     float hoverUpdateBegin = ((otherHand != null) && (otherHand.
        GetInstanceID() < GetInstanceID())) ? (0.5f *
        hoverUpdateInterval) : (0.0f);
1061     InvokeRepeating("UpdateHovering", hoverUpdateBegin,
        hoverUpdateInterval);
1062     InvokeRepeating("UpdateDebugText", hoverUpdateBegin,
        hoverUpdateInterval);
1063 }
1064
1065
1066 //-----
1067 protected virtual void OnDisable()
1068 {
1069     inputFocusAction.enabled = false;
1070
1071     CancelInvoke();
1072 }
1073
1074
1075 //-----
1076 protected virtual void Update()
1077 {
1078     UpdateNoSteamVRFallback();
1079
1080     GameObject attachedObject = currentAttachedObject;
1081     if (attachedObject != null)
1082     {

```

```

1083         attachedObject.SendMessage("HandAttachedUpdate", this,
1084                                     SendMessageOptions.DontRequireReceiver);
1085     }
1086     if (hoveringInteractable)
1087     {
1088         hoveringInteractable.SendMessage("HandHoverUpdate", this,
1089                                         SendMessageOptions.DontRequireReceiver);
1090     }
1091 }
1092 /// <summary>
1093 /// Returns true when the hand is currently hovering over the
1094 /// interactable passed in
1095 /// </summary>
1096 public bool IsStillHovering(Interactable interactable)
1097 {
1098     return hoveringInteractable == interactable;
1099 }
1100 protected virtual void HandFollowUpdate()
1101 {
1102     GameObject attachedObject = currentAttachedObject;
1103     if (attachedObject != null)
1104     {
1105         if (currentAttachedObjectInfo.Value.interactable != null)
1106         {
1107             SteamVR_Skeleton_PoseSnapshot pose = null;
1108
1109             if (currentAttachedObjectInfo.Value.interactable.
1110                 skeletonPoser != null && HasSkeleton())
1111             {
1112                 pose = currentAttachedObjectInfo.Value.
1113                     interactable.skeletonPoser.GetBlendedPose(
1114                         skeleton);
1115             }
1116
1117             if (currentAttachedObjectInfo.Value.interactable.
1118                 handFollowTransform)
1119             {
1120                 Quaternion targetHandRotation;
1121                 Vector3 targetHandPosition;
1122
1123                 if (pose == null)
1124                 {
1125                     Quaternion offset = Quaternion.Inverse(this.
1126                         transform.rotation) *
1127                         currentAttachedObjectInfo.Value.
1128                         handAttachmentPointTransform.rotation;
1129                     targetHandRotation =
1130                         currentAttachedObjectInfo.Value.
1131                         interactable.transform.rotation *
1132                         Quaternion.Inverse(offset);
1133                 }
1134             }
1135         }
1136     }

```



```

1124         Vector3 worldOffset = (this.transform.
            position - currentAttachedObjectInfo.
            Value.handAttachmentPointTransform.
            position);
1125         Quaternion rotationDiff = mainRenderModel.
            GetHandRotation() * Quaternion.Inverse(
            this.transform.rotation);
1126         Vector3 localOffset = rotationDiff *
            worldOffset;
1127         targetHandPosition =
            currentAttachedObjectInfo.Value.
            interactable.transform.position +
            localOffset;
1128     }
1129     else
1130     {
1131         Transform objectT = currentAttachedObjectInfo
            .Value.attachedObject.transform;
1132         Vector3 oldItemPos = objectT.position;
1133         Quaternion oldItemRot = objectT.transform.
            rotation;
1134         objectT.position = TargetItemPosition(
            currentAttachedObjectInfo.Value);
1135         objectT.rotation = TargetItemRotation(
            currentAttachedObjectInfo.Value);
1136         Vector3 localSkelePos = objectT.
            InverseTransformPoint(transform.position)
            ;
1137         Quaternion localSkeleRot = Quaternion.Inverse
            (objectT.rotation) * transform.rotation;
1138         objectT.position = oldItemPos;
1139         objectT.rotation = oldItemRot;
1140
1141         targetHandPosition = objectT.TransformPoint(
            localSkelePos);
1142         targetHandRotation = objectT.rotation *
            localSkeleRot;
1143     }
1144
1145     if (mainRenderModel != null)
1146         mainRenderModel.SetHandRotation(
            targetHandRotation);
1147     if (hoverhighlightRenderModel != null)
1148         hoverhighlightRenderModel.SetHandRotation(
            targetHandRotation);
1149
1150     if (mainRenderModel != null)
1151         mainRenderModel.SetHandPosition(
            targetHandPosition);
1152     if (hoverhighlightRenderModel != null)
1153         hoverhighlightRenderModel.SetHandPosition(
            targetHandPosition);
1154 }
1155 }

```

```

1156     }
1157 }
1158
1159 protected virtual void FixedUpdate()
1160 {
1161     if (currentAttachedObject != null)
1162     {
1163         AttachedObject attachedInfo = currentAttachedObjectInfo.
            Value;
1164         if (attachedInfo.attachedObject != null)
1165         {
1166             if (attachedInfo.HasAttachFlag(AttachmentFlags.
                VelocityMovement))
1167             {
1168                 if (attachedInfo.interactable.attachEaseIn ==
                    false || attachedInfo.interactable.
                        snapAttachEaseInCompleted)
1169                     UpdateAttachedVelocity(attachedInfo);
1170
1171                 /*if (attachedInfo.interactable.
                    handFollowTransformPosition)
1172                 {
1173                     skeleton.transform.position =
                        TargetSkeletonPosition(attachedInfo);
1174                     skeleton.transform.rotation = attachedInfo.
                        attachedObject.transform.rotation *
                        attachedInfo.skeletonLockRotation;
1175                 }*/
1176             }else
1177             {
1178                 if (attachedInfo.HasAttachFlag(AttachmentFlags.
                    ParentToHand))
1179                 {
1180                     attachedInfo.attachedObject.transform.
                        position = TargetItemPosition(
                            attachedInfo);
1181                     attachedInfo.attachedObject.transform.
                        rotation = TargetItemRotation(
                            attachedInfo);
1182                 }
1183             }
1184
1185             if (attachedInfo.interactable.attachEaseIn)
1186             {
1187                 float t = Util.RemapNumberClamped(Time.time,
                    attachedInfo.attachTime, attachedInfo.
                        attachTime + attachedInfo.interactable.
                            snapAttachEaseInTime, 0.0f, 1.0f);
1189                 if (t < 1.0f)
1190                 {
1191                     if (attachedInfo.HasAttachFlag(
                        AttachmentFlags.VelocityMovement))
1192                     {

```

```

1193         attachedInfo.attachedRigidbody.velocity =
1194             Vector3.zero;
1195         attachedInfo.attachedRigidbody.
1196             angularVelocity = Vector3.zero;
1197     }
1198     t = attachedInfo.interactable.
1199         snapAttachEaseInCurve.Evaluate(t);
1200     attachedInfo.attachedObject.transform.
1201         position = Vector3.Lerp(attachedInfo.
1202             easeSourcePosition, TargetItemPosition(
1203                 attachedInfo), t);
1204     attachedInfo.attachedObject.transform.
1205         rotation = Quaternion.Lerp(attachedInfo.
1206             easeSourceRotation, TargetItemRotation(
1207                 attachedInfo), t);
1208 }
1209 else if (!attachedInfo.interactable.
1210     snapAttachEaseInCompleted)
1211 {
1212     attachedInfo.interactable.gameObject.
1213         SendMessage("
1214             OnThrowableAttachEaseInCompleted", this,
1215             SendMessageOptions.DontRequireReceiver);
1216     attachedInfo.interactable.
1217         snapAttachEaseInCompleted = true;
1218 }
1219 }
1220 }
1221 }
1222
1223 protected const float MaxVelocityChange = 10f;
1224 protected const float VelocityMagic = 6000f;
1225 protected const float AngularVelocityMagic = 50f;
1226 protected const float MaxAngularVelocityChange = 20f;
1227
1228 protected void UpdateAttachedVelocity(AttachedObject
1229     attachedObjectInfo)
1230 {
1231     Vector3 velocityTarget, angularTarget;
1232     bool success = GetUpdatedAttachedVelocities(
1233         attachedObjectInfo, out velocityTarget, out angularTarget
1234     );
1235     if (success)
1236     {
1237         float scale = SteamVR_Utils.GetLossyScale(
1238             currentAttachedObjectInfo.Value.
1239             handAttachmentPointTransform);
1240         float maxAngularVelocityChange = MaxAngularVelocityChange
1241             * scale;
1242         float maxVelocityChange = MaxVelocityChange * scale;
1243         attachedObjectInfo.attachedRigidbody.velocity = Vector3.
1244             MoveTowards(attachedObjectInfo.attachedRigidbody.

```

```

1226         velocity, velocityTarget, maxVelocityChange);
1227         attachedObjectInfo.attachedRigidbody.angularVelocity =
1228         Vector3.MoveTowards(attachedObjectInfo.
1229         attachedRigidbody.angularVelocity, angularTarget,
1230         maxAngularVelocityChange);
1227     }
1228 }
1229
1230 protected Vector3 TargetItemPosition(AttachedObject
1231 attachedObject)
1232 {
1233     if (attachedObject.interactable != null && attachedObject.
1234     interactable.skeletonPoser != null && HasSkeleton())
1235     {
1236         Vector3 tp = attachedObject.handAttachmentPointTransform.
1237         InverseTransformPoint(transform.TransformPoint(
1238         attachedObject.interactable.skeletonPoser.
1239         GetBlendedPose(skeleton).position));
1240         //tp.x *= -1;
1241         return currentAttachedObjectInfo.Value.
1242         handAttachmentPointTransform.TransformPoint(tp);
1243     }
1244     else
1245     {
1246         return currentAttachedObjectInfo.Value.
1247         handAttachmentPointTransform.TransformPoint(
1248         attachedObject.initialPositionalOffset);
1249     }
1250 }
1251
1252 protected Quaternion TargetItemRotation(AttachedObject
1253 attachedObject)
1254 {
1255     if (attachedObject.interactable != null && attachedObject.
1256     interactable.skeletonPoser != null && HasSkeleton())
1257     {
1258         Quaternion tr = Quaternion.Inverse(attachedObject.
1259         handAttachmentPointTransform.rotation) * (transform.
1260         rotation * attachedObject.interactable.skeletonPoser.
1261         GetBlendedPose(skeleton).rotation);
1262         return currentAttachedObjectInfo.Value.
1263         handAttachmentPointTransform.rotation * tr;
1264     }
1265     else
1266     {
1267         return currentAttachedObjectInfo.Value.
1268         handAttachmentPointTransform.rotation *
1269         attachedObject.initialRotationalOffset;
1270     }
1271 }
1272
1273 protected bool GetUpdatedAttachedVelocities(AttachedObject
1274 attachedObjectInfo, out Vector3 velocityTarget, out Vector3
1275 angularTarget)

```

```

1258     {
1259         bool realNumbers = false;
1260
1261         float velocityMagic = VelocityMagic;
1262         float angularVelocityMagic = AngularVelocityMagic;
1263
1264         Vector3 targetItemPosition = TargetItemPosition(
1265             attachedObjectInfo);
1266         Vector3 positionDelta = (targetItemPosition -
1267             attachedObjectInfo.attachedRigidbody.position);
1268         velocityTarget = (positionDelta * velocityMagic * Time.
1269             deltaTime);
1270
1271         if (float.IsNaN(velocityTarget.x) == false && float.
1272             IsInfinity(velocityTarget.x) == false)
1273         {
1274             if (noSteamVRFallbackCamera)
1275                 velocityTarget /= 10; //hacky fix for fallback
1276
1277             realNumbers = true;
1278         }
1279         else
1280             velocityTarget = Vector3.zero;
1281
1282         Quaternion targetItemRotation = TargetItemRotation(
1283             attachedObjectInfo);
1284         Quaternion rotationDelta = targetItemRotation * Quaternion.
1285             Inverse(attachedObjectInfo.attachedObject.transform.
1286                 rotation);
1287
1288         float angle;
1289         Vector3 axis;
1290         rotationDelta.ToAngleAxis(out angle, out axis);
1291
1292         if (angle > 180)
1293             angle -= 360;
1294
1295         if (angle != 0 && float.IsNaN(axis.x) == false && float.
1296             IsInfinity(axis.x) == false)
1297         {
1298             angularTarget = angle * axis * angularVelocityMagic *
1299                 Time.deltaTime;
1300
1301             if (noSteamVRFallbackCamera)
1302                 angularTarget /= 10; //hacky fix for fallback
1303
1304             realNumbers &= true;
1305         }
1306         else
1307             angularTarget = Vector3.zero;

```

```

1303         return realNumbers;
1304     }
1305
1306
1307     //-----
1308     protected virtual void OnInputFocus(bool hasFocus)
1309     {
1310         if (hasFocus)
1311         {
1312             DetachObject(applicationLostFocusObject, true);
1313             applicationLostFocusObject.SetActive(false);
1314             UpdateHovering();
1315             BroadcastMessage("OnParentHandInputFocusAcquired",
1316                             SendMessageOptions.DontRequireReceiver);
1317         }
1318         else
1319         {
1320             applicationLostFocusObject.SetActive(true);
1321             AttachObject(applicationLostFocusObject, GrabTypes.
1322                           Scripted, AttachmentFlags.ParentToHand);
1323             BroadcastMessage("OnParentHandInputFocusLost",
1324                             SendMessageOptions.DontRequireReceiver);
1325         }
1326     }
1327
1328     //-----
1329     protected virtual void OnDrawGizmos()
1330     {
1331         if (useHoverSphere && hoverSphereTransform != null)
1332         {
1333             Gizmos.color = Color.green;
1334             float scaledHoverRadius = hoverSphereRadius * Mathf.Abs(
1335                                     SteamVR_Utils.GetLossyScale(hoverSphereTransform));
1336             Gizmos.DrawWireSphere(hoverSphereTransform.position,
1337                                   scaledHoverRadius/2);
1338         }
1339
1340         if (useControllerHoverComponent && mainRenderModel != null &&
1341             mainRenderModel.IsControllerVisibile())
1342         {
1343             Gizmos.color = Color.blue;
1344             float scaledHoverRadius = controllerHoverRadius * Mathf.
1345                                     Abs(SteamVR_Utils.GetLossyScale(this.transform));
1346             Gizmos.DrawWireSphere(mainRenderModel.
1347                                   GetControllerPosition(controllerHoverComponent),
1348                                   scaledHoverRadius/2);
1349         }
1350
1351         if (useFingerJointHover && mainRenderModel != null &&
1352             mainRenderModel.IsHandVisibile())
1353         {
1354             Gizmos.color = Color.yellow;
1355             float scaledHoverRadius = fingerJointHoverRadius * Mathf.
1356                                     Abs(SteamVR_Utils.GetLossyScale(this.transform));

```

```

1346         Gizmos.DrawWireSphere(mainRenderModel.GetBonePosition((
1347             int)fingerJointHover), scaledHoverRadius/2);
1348     }
1349 }
1350
1351 //-----
1352 private void HandDebugLog(string msg)
1353 {
1354     if (spewDebugText)
1355     {
1356         Debug.Log("<b>[SteamVR Interaction]</b> Hand (" + this.
1357             name + "): " + msg);
1358     }
1359 }
1360
1361 //-----
1362 // Continue to hover over this object indefinitely, whether or
1363 // not the Hand moves out of its interaction trigger volume.
1364 //
1365 // interactable - The Interactable to hover over indefinitely.
1366 //-----
1367 public void HoverLock(Interactable interactable)
1368 {
1369     if (spewDebugText)
1370         HandDebugLog("HoverLock " + interactable);
1371     hoverLocked = true;
1372     hoveringInteractable = interactable;
1373 }
1374
1375 //-----
1376 // Stop hovering over this object indefinitely.
1377 //
1378 // interactable - The hover-locked Interactable to stop hovering
1379 // over indefinitely.
1380 //-----
1381 public void HoverUnlock(Interactable interactable)
1382 {
1383     if (spewDebugText)
1384         HandDebugLog("HoverUnlock " + interactable);
1385
1386     if (hoveringInteractable == interactable)
1387     {
1388         hoverLocked = false;
1389     }
1390 }
1391
1392 public void TriggerHapticPulse(ushort microsecondsDuration)
1393 {
1394     float seconds = (float)microsecondsDuration / 1000000f;
1395     hapticAction.Execute(0, seconds, 1f / seconds, 1, handType);

```

```

1396
1397     public void TriggerHapticPulse(float duration, float frequency,
1398                                   float amplitude)
1399     {
1400         hapticAction.Execute(0, duration, frequency, amplitude,
1401                               handType);
1402     }
1403
1404     public void ShowGrabHint()
1405     {
1406         ControllerButtonHints.ShowButtonHint(this, grabGripAction);
1407         //todo: assess
1408     }
1409
1410     public void HideGrabHint()
1411     {
1412         ControllerButtonHints.HideButtonHint(this, grabGripAction);
1413         //todo: assess
1414     }
1415
1416     public void ShowGrabHint(string text)
1417     {
1418         ControllerButtonHints.ShowTextHint(this, grabGripAction, text
1419                                             );
1420     }
1421
1422     public GrabTypes GetGrabStarting(GrabTypes explicitType =
1423                                     GrabTypes.None)
1424     {
1425         if (explicitType != GrabTypes.None)
1426         {
1427             if (noSteamVRFallbackCamera)
1428             {
1429                 if (Input.GetMouseButtonDown(0))
1430                     return explicitType;
1431             }
1432
1433             if (explicitType == GrabTypes.Pinch && grabPinchAction.
1434                 GetStateDown(handType))
1435                 return GrabTypes.Pinch;
1436             if (explicitType == GrabTypes.Grip && grabGripAction.
1437                 GetStateDown(handType))
1438                 return GrabTypes.Grip;
1439         }
1440         else
1441         {
1442             if (noSteamVRFallbackCamera)
1443             {
1444                 if (Input.GetMouseButtonDown(0))
1445                     return GrabTypes.Grip;
1446             }
1447
1448             if (grabPinchAction.GetStateDown(handType))
1449                 return GrabTypes.Pinch;
1450         }
1451     }

```



```

1442         if (grabGripAction.GetStateDown(handType))
1443             return GrabTypes.Grip;
1444     }
1445
1446     return GrabTypes.None;
1447 }
1448
1449 public GrabTypes GetGrabEnding(GrabTypes explicitType = GrabTypes
1450     .None)
1451 {
1452     if (explicitType != GrabTypes.None)
1453     {
1454         if (noSteamVRFallbackCamera)
1455         {
1456             if (Input.GetMouseButtonUp(0))
1457                 return explicitType;
1458         }
1459
1460         if (explicitType == GrabTypes.Pinch && grabPinchAction.
1461             GetStateUp(handType))
1462             return GrabTypes.Pinch;
1463         if (explicitType == GrabTypes.Grip && grabGripAction.
1464             GetStateUp(handType))
1465             return GrabTypes.Grip;
1466     }
1467     else
1468     {
1469         if (noSteamVRFallbackCamera)
1470         {
1471             if (Input.GetMouseButtonUp(0))
1472                 return GrabTypes.Grip;
1473         }
1474
1475         if (grabPinchAction.GetStateUp(handType))
1476             return GrabTypes.Pinch;
1477         if (grabGripAction.GetStateUp(handType))
1478             return GrabTypes.Grip;
1479     }
1480
1481     return GrabTypes.None;
1482 }
1483
1484 public bool IsGrabEnding(GameObject attachedObject)
1485 {
1486     for (int attachedObjectIndex = 0; attachedObjectIndex <
1487         attachedObjects.Count; attachedObjectIndex++)
1488     {
1489         if (attachedObjects[attachedObjectIndex].attachedObject
1490             == attachedObject)
1491         {
1492             return IsGrabbingWithType(attachedObjects[
1493                 attachedObjectIndex].grabbedWithType) == false;
1494         }
1495     }
1496 }

```

```

1490         return false;
1491     }
1492
1493
1494     public bool IsGrabbingWithType(GrabTypes type)
1495     {
1496         if (noSteamVRFallbackCamera)
1497         {
1498             if (Input.GetMouseButton(0))
1499                 return true;
1500         }
1501
1502         switch (type)
1503         {
1504             case GrabTypes.Pinch:
1505                 return grabPinchAction.GetState(handType);
1506
1507             case GrabTypes.Grip:
1508                 return grabGripAction.GetState(handType);
1509
1510             default:
1511                 return false;
1512         }
1513     }
1514
1515     public bool IsGrabbingWithOppositeType(GrabTypes type)
1516     {
1517         if (noSteamVRFallbackCamera)
1518         {
1519             if (Input.GetMouseButton(0))
1520                 return true;
1521         }
1522
1523         switch (type)
1524         {
1525             case GrabTypes.Pinch:
1526                 return grabGripAction.GetState(handType);
1527
1528             case GrabTypes.Grip:
1529                 return grabPinchAction.GetState(handType);
1530
1531             default:
1532                 return false;
1533         }
1534     }
1535
1536     public GrabTypes GetBestGrabbingType()
1537     {
1538         return GetBestGrabbingType(GrabTypes.None);
1539     }
1540
1541     public GrabTypes GetBestGrabbingType(GrabTypes preferred, bool
forcePreference = false)
1542     {

```

```

1543         if (noSteamVRFallbackCamera)
1544         {
1545             if (Input.GetMouseButton(0))
1546                 return preferred;
1547         }
1548
1549         if (preferred == GrabTypes.Pinch)
1550         {
1551             if (grabPinchAction.GetState(handType))
1552                 return GrabTypes.Pinch;
1553             else if (forcePreference)
1554                 return GrabTypes.None;
1555         }
1556         if (preferred == GrabTypes.Grip)
1557         {
1558             if (grabGripAction.GetState(handType))
1559                 return GrabTypes.Grip;
1560             else if (forcePreference)
1561                 return GrabTypes.None;
1562         }
1563
1564         if (grabPinchAction.GetState(handType))
1565             return GrabTypes.Pinch;
1566         if (grabGripAction.GetState(handType))
1567             return GrabTypes.Grip;
1568
1569         return GrabTypes.None;
1570     }
1571
1572
1573     //-----
1574     private void InitController()
1575     {
1576         if (spewDebugText)
1577             HandDebugLog("Hand " + name + " connected with type " +
1578                 handType.ToString());
1579
1580         bool hadOldRendermodel = mainRenderModel != null;
1581         EVRSkeletalMotionRange oldRM_rom = EVRSkeletalMotionRange.
1582             WithController;
1583         if(hadOldRendermodel)
1584             oldRM_rom = mainRenderModel.GetSkeletonRangeOfMotion;
1585
1586         foreach (RenderModel r in renderModels)
1587         {
1588             if (r != null)
1589                 Destroy(r.gameObject);
1590         }
1591
1592         renderModels.Clear();
1593
1594         GameObject renderModelInstance = GameObject.Instantiate(
1595             renderModelPrefab);

```

```

1594         renderModelInstance.layer = gameObject.layer;
1595         renderModelInstance.tag = gameObject.tag;
1596         renderModelInstance.transform.parent = this.transform;
1597         renderModelInstance.transform.localPosition = Vector3.zero;
1598         renderModelInstance.transform.localRotation = Quaternion.
            identity;
1599         renderModelInstance.transform.localScale = renderModelPrefab.
            transform.localScale;
1600
1601         //TriggerHapticPulse(800); //pulse on controller init
1602
1603         int deviceIndex = trackedObject.GetDeviceIndex();
1604
1605         mainRenderModel = renderModelInstance.GetComponent<
            RenderModel>();
1606         renderModels.Add(mainRenderModel);
1607
1608         if (hadOldRendermodel)
1609             mainRenderModel.SetSkeletonRangeOfMotion(oldRM_rom);
1610
1611         this.BroadcastMessage("SetInputSource", handType,
            SendMessageOptions.DontRequireReceiver); // let child
            objects know we've initialized
1612         this.BroadcastMessage("OnHandInitialized", deviceIndex,
            SendMessageOptions.DontRequireReceiver); // let child
            objects know we've initialized
1613     }
1614
1615     public void SetRenderModel(GameObject prefab)
1616     {
1617         renderModelPrefab = prefab;
1618
1619         if (mainRenderModel != null && isPoseValid)
1620             InitController();
1621     }
1622
1623     public void SetHoverRenderModel(RenderModel hoverRenderModel)
1624     {
1625         hoverhighlightRenderModel = hoverRenderModel;
1626         renderModels.Add(hoverRenderModel);
1627     }
1628
1629     public int GetDeviceIndex()
1630     {
1631         return trackedObject.GetDeviceIndex();
1632     }
1633 }
1634
1635
1636 [System.Serializable]
1637 public class HandEvent : UnityEvent<Hand> { }
1638
1639
1640 #if UNITY_EDITOR

```

```

1641 // -----
1642 [UnityEditor.CustomEditor(typeof(Hand))]
1643 public class HandEditor : UnityEditor.Editor
1644 {
1645     //-----
1646     // Custom Inspector GUI allows us to click from within the UI
1647     //-----
1648     public override void OnInspectorGUI()
1649     {
1650         DrawDefaultInspector();
1651
1652         /*
1653         Hand hand = (Hand)target;
1654
1655         if (hand.otherHand)
1656         {
1657             if (hand.otherHand.otherHand != hand)
1658             {
1659                 UnityEditor.EditorGUILayout.HelpBox("The otherHand of
1660                 this Hand's otherHand is not this Hand.",
1661                 MessageType.Warning);
1662             }
1663
1664             if (hand.handType == SteamVR_Input_Sources.LeftHand &&
1665                 hand.otherHand && hand.otherHand.handType !=
1666                 SteamVR_Input_Sources.RightHand)
1667             {
1668                 UnityEditor.EditorGUILayout.HelpBox("This is a left
1669                 Hand but otherHand is not a right Hand.",
1670                 MessageType.Warning);
1671             }
1672
1673             if (hand.handType == SteamVR_Input_Sources.RightHand &&
1674                 hand.otherHand && hand.otherHand.handType !=
1675                 SteamVR_Input_Sources.LeftHand)
1676             {
1677                 UnityEditor.EditorGUILayout.HelpBox("This is a right
1678                 Hand but otherHand is not a left Hand.",
1679                 MessageType.Warning);
1680             }
1681
1682             if (hand.handType == SteamVR_Input_Sources.Any && hand.
1683                 otherHand && hand.otherHand.handType !=
1684                 SteamVR_Input_Sources.Any)
1685             {
1686                 UnityEditor.EditorGUILayout.HelpBox("This is an any-
1687                 handed Hand but otherHand is not an any-handed
1688                 Hand.", MessageType.Warning);
1689             }
1690         }
1691         */ //removing for now because it conflicts with other input
1692         sources (trackers and such)

```

```

1678     }
1679 }
1680 #endif
1681 }

```

9.5.5. IgnoreTeleportTrace.cs

```

1  //===== Copyright (c) Valve Corporation, All rights reserved.
   =====
2  //
3  // Purpose: Allows the teleport arc trace to pass through any colliders
   on this
4  //      object
5  //
6  //
   =====
7
8  using UnityEngine;
9  using System.Collections;
10
11 namespace Valve.VR.InteractionSystem
12 {
13     //
   -----
14     public class IgnoreTeleportTrace : MonoBehaviour
15     {
16     }
17 }

```

9.5.6. InputModule.cs

```

1  //===== Copyright (c) Valve Corporation, All rights reserved.
   =====
2  //
3  // Purpose: Makes the hand act as an input module for Unity's event
   system
4  //
5  //
   =====
6
7  using UnityEngine;
8  using System.Collections;
9  using UnityEngine.EventSystems;
10
11 namespace Valve.VR.InteractionSystem
12 {

```

```

13  // -----
14  public class InputModule : BaseInputModule
15  {
16      private GameObject submitObject;
17
18      //-----
19      private static InputModule _instance;
20      public static InputModule instance
21      {
22          get
23          {
24              if ( _instance == null )
25                  _instance = GameObject.FindObjectOfType<InputModule>();
26
27              return _instance;
28          }
29      }
30
31
32      //-----
33      public override bool ShouldActivateModule()
34      {
35          if ( !base.ShouldActivateModule() )
36              return false;
37
38          return submitObject != null;
39      }
40
41
42      //-----
43      public void HoverBegin( GameObject gameObject )
44      {
45          PointerEventData pointerEventData = new PointerEventData(
46              eventSystem );
47          ExecuteEvents.Execute( gameObject, pointerEventData, ExecuteEvents.
48              pointerEnterHandler );
49
50
51      //-----
52      public void HoverEnd( GameObject gameObject )
53      {
54          PointerEventData pointerEventData = new PointerEventData(
55              eventSystem );
56          pointerEventData.selectedObject = null;
57          ExecuteEvents.Execute( gameObject, pointerEventData, ExecuteEvents.
58              pointerExitHandler );
59
60      //-----
61      public void Submit( GameObject gameObject )

```

```

61     {
62         submitObject = gameObject;
63     }
64
65
66     //-----
67     public override void Process()
68     {
69         if ( submitObject )
70         {
71             BaseEventData data = GetBaseEventData();
72             data.selectedObject = submitObject;
73             ExecuteEvents.Execute( submitObject, data, ExecuteEvents.
                submitHandler );
74
75             submitObject = null;
76         }
77     }
78 }
79 }

```

9.5.7. Interactable.cs

```

1  //===== Copyright (c) Valve Corporation, All rights reserved.
   //=====
2  //
3  // Purpose: This object will get hover events and can be attached to the
   hands
4  //
5  //
   =====
6
7  using UnityEngine;
8  using UnityEngine.Events;
9  using System.Collections;
10 using System.Collections.Generic;
11
12 namespace Valve.VR.InteractionSystem
13 {
14     //
   -----
15     public class Interactable : MonoBehaviour
16     {
17         [Tooltip("Activates an action set on attach and deactivates on
            detach")]
18         public SteamVR_ActionSet activateActionSetOnAttach;
19
20         [Tooltip("Hide the whole hand on attachment and show on detach")]
21         public bool hideHandOnAttach = true;
22

```



```

23     [Tooltip("Hide the skeleton part of the hand on attachment and
24         show on detach")]
25     public bool hideSkeletonOnAttach = false;
26
27     [Tooltip("Hide the controller part of the hand on attachment and
28         show on detach")]
29     public bool hideControllerOnAttach = false;
30
31     [Tooltip("The integer in the animator to trigger on pickup. 0 for
32         none")]
33     public int handAnimationOnPickup = 0;
34
35     [Tooltip("The range of motion to set on the skeleton. None for no
36         change.")]
37     public SkeletalMotionRangeChange setRangeOfMotionOnPickup =
38         SkeletalMotionRangeChange.None;
39
40     public delegate void OnAttachedToHandDelegate(Hand hand);
41     public delegate void OnDetachedFromHandDelegate(Hand hand);
42
43     public event OnAttachedToHandDelegate onAttachedToHand;
44     public event OnDetachedFromHandDelegate onDetachedFromHand;
45
46     [Tooltip("Specify whether you want to snap to the hand's object
47         attachment point, or just the raw hand")]
48     public bool useHandObjectAttachmentPoint = true;
49
50     public bool attachEaseIn = false;
51     [HideInInspector]
52     public AnimationCurve snapAttachEaseInCurve = AnimationCurve.
53         EaseInOut(0.0f, 0.0f, 1.0f, 1.0f);
54     public float snapAttachEaseInTime = 0.15f;
55
56     public bool snapAttachEaseInCompleted = false;
57
58     // [Tooltip("The skeleton pose to apply when grabbing. Can only
59         set this or handFollowTransform.")]
60     [HideInInspector]
61     public SteamVR_Skeleton_Poser skeletonPoser;
62
63     [Tooltip("Should the rendered hand lock on to and follow the
64         object")]
65     public bool handFollowTransform = true;
66
67     [Tooltip("Set whether or not you want this interactible to
68         highlight when hovering over it")]
69     public bool highlightOnHover = true;
70     protected MeshRenderer[] highlightRenderers;
71     protected MeshRenderer[] existingRenderers;
72     protected GameObject highlightHolder;
73     protected SkinnedMeshRenderer[] highlightSkinnedRenderers;

```

```

67     protected SkinnedMeshRenderer[] existingSkinnedRenderers;
68     protected static Material highlightMat;
69     [Tooltip("An array of child gameObjects to not render a highlight
    for. Things like transparent parts, vfx, etc.")]
70     public GameObject[] hideHighlight;
71
72
73     [System.NonSerialized]
74     public Hand attachedToHand;
75
76     [System.NonSerialized]
77     public Hand hoveringHand;
78
79     public bool isDestroying { get; protected set; }
80     public bool isHovering { get; protected set; }
81     public bool wasHovering { get; protected set; }
82
83
84     private void Awake()
85     {
86         skeletonPoser = GetComponent<SteamVR_Skeleton_Poser>();
87     }
88
89     protected virtual void Start()
90     {
91         highlightMat = (Material)Resources.Load("
    SteamVR_HoverHighlight", typeof(Material));
92
93         if (highlightMat == null)
94             Debug.LogError("<b>[SteamVR Interaction]</b> Hover
    Highlight Material is missing. Please create a
    material named 'SteamVR_HoverHighlight' and place it
    in a Resources folder");
95
96         if (skeletonPoser != null)
97         {
98             if (useHandObjectAttachmentPoint)
99             {
100                 //Debug.LogWarning("<b>[SteamVR Interaction]</b>
    SkeletonPose and useHandObjectAttachmentPoint
    both set at the same time. Ignoring
    useHandObjectAttachmentPoint.");
101                 useHandObjectAttachmentPoint = false;
102             }
103         }
104     }
105
106     protected virtual bool ShouldIgnoreHighlight(Component component)
107     {
108         return ShouldIgnore(component.gameObject);
109     }
110
111     protected virtual bool ShouldIgnore(GameObject check)
112     {

```

```

113         for (int ignoreIndex = 0; ignoreIndex < hideHighlight.Length;
114             ignoreIndex++)
115         {
116             if (check == hideHighlight[ignoreIndex])
117                 return true;
118         }
119         return false;
120     }
121
122     protected virtual void CreateHighlightRenderers()
123     {
124         existingSkinnedRenderers = this.GetComponentsInChildren<
125             SkinnedMeshRenderer>(true);
126         highlightHolder = new GameObject("Highlighter");
127         highlightSkinnedRenderers = new SkinnedMeshRenderer[
128             existingSkinnedRenderers.Length];
129
130         for (int skinnedIndex = 0; skinnedIndex <
131             existingSkinnedRenderers.Length; skinnedIndex++)
132         {
133             SkinnedMeshRenderer existingSkinned =
134                 existingSkinnedRenderers[skinnedIndex];
135
136             if (ShouldIgnoreHighlight(existingSkinned))
137                 continue;
138
139             GameObject newSkinnedHolder = new GameObject("
140                 SkinnedHolder");
141             newSkinnedHolder.transform.parent = highlightHolder.
142                 transform;
143             SkinnedMeshRenderer newSkinned = newSkinnedHolder.
144                 AddComponent<SkinnedMeshRenderer>();
145             Material[] materials = new Material[existingSkinned.
146                 sharedMaterials.Length];
147             for (int materialIndex = 0; materialIndex < materials.
148                 Length; materialIndex++)
149             {
150                 materials[materialIndex] = highlightMat;
151             }
152
153             newSkinned.sharedMaterials = materials;
154             newSkinned.sharedMesh = existingSkinned.sharedMesh;
155             newSkinned.rootBone = existingSkinned.rootBone;
156             newSkinned.updateWhenOffscreen = existingSkinned.
157                 updateWhenOffscreen;
158             newSkinned.bones = existingSkinned.bones;
159
160             highlightSkinnedRenderers[skinnedIndex] = newSkinned;
161         }
162
163         MeshFilter[] existingFilters = this.GetComponentsInChildren<
164             MeshFilter>(true);
165         existingRenderers = new MeshRenderer[existingFilters.Length];

```

```

155         highlightRenderers = new MeshRenderer[existingFilters.Length
156         ];
157         for (int filterIndex = 0; filterIndex < existingFilters.
158             Length; filterIndex++)
159         {
160             MeshFilter existingFilter = existingFilters[filterIndex];
161             MeshRenderer existingRenderer = existingFilter.
162                 GetComponent<MeshRenderer>();
163
164             if (existingFilter == null || existingRenderer == null ||
165                 ShouldIgnoreHighlight(existingFilter))
166                 continue;
167
168             GameObject newFilterHolder = new GameObject("FilterHolder
169                 ");
170             newFilterHolder.transform.parent = highlightHolder.
171                 transform;
172             MeshFilter newFilter = newFilterHolder.AddComponent<
173                 MeshFilter>();
174             newFilter.sharedMesh = existingFilter.sharedMesh;
175             MeshRenderer newRenderer = newFilterHolder.AddComponent<
176                 MeshRenderer>();
177
178             Material[] materials = new Material[existingRenderer.
179                 sharedMaterials.Length];
180             for (int materialIndex = 0; materialIndex < materials.
181                 Length; materialIndex++)
182             {
183                 materials[materialIndex] = highlightMat;
184             }
185             newRenderer.sharedMaterials = materials;
186
187             highlightRenderers[filterIndex] = newRenderer;
188             existingRenderers[filterIndex] = existingRenderer;
189         }
190     }
191
192     protected virtual void UpdateHighlightRenderers()
193     {
194         if (highlightHolder == null)
195             return;
196
197         for (int skinnedIndex = 0; skinnedIndex <
198             existingSkinnedRenderers.Length; skinnedIndex++)
199         {
200             SkinnedMeshRenderer existingSkinned =
201                 existingSkinnedRenderers[skinnedIndex];
202             SkinnedMeshRenderer highlightSkinned =
203                 highlightSkinnedRenderers[skinnedIndex];
204
205             if (existingSkinned != null && highlightSkinned != null
206                 && attachedToHand == false)
207             {

```

```

195         highlightSkinned.transform.position = existingSkinned
196             .transform.position;
197         highlightSkinned.transform.rotation = existingSkinned
198             .transform.rotation;
199         highlightSkinned.transform.localScale =
200             existingSkinned.transform.lossyScale;
201         highlightSkinned.localBounds = existingSkinned.
202             localBounds;
203         highlightSkinned.enabled = isHovering &&
204             existingSkinned.enabled && existingSkinned.
205             gameObject.activeInHierarchy;
206
207         int blendShapeCount = existingSkinned.sharedMesh.
208             blendShapeCount;
209         for (int blendShapeIndex = 0; blendShapeIndex <
210             blendShapeCount; blendShapeIndex++)
211         {
212             highlightSkinned.SetBlendShapeWeight(
213                 blendShapeIndex, existingSkinned.
214                     GetBlendShapeWeight(blendShapeIndex));
215         }
216     }
217     else if (highlightSkinned != null)
218         highlightSkinned.enabled = false;
219 }
220
221 for (int rendererIndex = 0; rendererIndex <
222     highlightRenderers.Length; rendererIndex++)
223 {
224     MeshRenderer existingRenderer = existingRenderers[
225         rendererIndex];
226     MeshRenderer highlightRenderer = highlightRenderers[
227         rendererIndex];
228
229     if (existingRenderer != null && highlightRenderer != null
230         && attachedToHand == false)
231     {
232         highlightRenderer.transform.position =
233             existingRenderer.transform.position;
234         highlightRenderer.transform.rotation =
235             existingRenderer.transform.rotation;
236         highlightRenderer.transform.localScale =
237             existingRenderer.transform.lossyScale;
238         highlightRenderer.enabled = isHovering &&
239             existingRenderer.enabled && existingRenderer.
240                 gameObject.activeInHierarchy;
241     }
242     else if (highlightRenderer != null)
243         highlightRenderer.enabled = false;
244 }
245 }
246
247 /// <summary>

```

```

230     /// Called when a Hand starts hovering over this object
231     /// </summary>
232     protected virtual void OnHandHoverBegin(Hand hand)
233     {
234         wasHovering = isHovering;
235         isHovering = true;
236
237         hoveringHand = hand;
238
239         if (highlightOnHover == true)
240         {
241             CreateHighlightRenderers();
242             UpdateHighlightRenderers();
243         }
244     }
245
246
247     /// <summary>
248     /// Called when a Hand stops hovering over this object
249     /// </summary>
250     private void OnHandHoverEnd(Hand hand)
251     {
252         wasHovering = isHovering;
253         isHovering = false;
254
255         if (highlightOnHover && highlightHolder != null)
256             Destroy(highlightHolder);
257     }
258
259     protected virtual void Update()
260     {
261         if (highlightOnHover)
262         {
263             UpdateHighlightRenderers();
264
265             if (isHovering == false && highlightHolder != null)
266                 Destroy(highlightHolder);
267         }
268     }
269
270
271     protected float blendToPoseTime = 0.1f;
272     protected float releasePoseBlendTime = 0.2f;
273
274     protected virtual void OnAttachedToHand(Hand hand)
275     {
276         if (activateActionSetOnAttach != null)
277             activateActionSetOnAttach.Activate(hand.handType);
278
279         if (onAttachedToHand != null)
280         {
281             onAttachedToHand.Invoke(hand);
282         }
283     }

```

```

284         if (skeletonPoser != null && hand.skeleton != null)
285         {
286             hand.skeleton.BlendToPoser(skeletonPoser, blendToPoseTime
287                                     );
288         }
289         attachedToHand = hand;
290     }
291
292     protected virtual void OnDetachedFromHand(Hand hand)
293     {
294         if (activateActionSetOnAttach != null)
295         {
296             if (hand.otherHand == null || hand.otherHand.
297                 currentAttachedObjectInfo.HasValue == false ||
298                 (hand.otherHand.currentAttachedObjectInfo.Value.
299                     interactable != null &&
300                     hand.otherHand.currentAttachedObjectInfo.Value.
301                         interactable.activateActionSetOnAttach != this.
302                             activateActionSetOnAttach))
303             {
304                 activateActionSetOnAttach.Deactivate(hand.handType);
305             }
306         }
307
308         if (onDetachedFromHand != null)
309         {
310             onDetachedFromHand.Invoke(hand);
311         }
312
313         if (skeletonPoser != null)
314         {
315             if (hand.skeleton != null)
316                 hand.skeleton.BlendToSkeleton(releasePoseBlendTime);
317         }
318
319         attachedToHand = null;
320     }
321
322     protected virtual void OnDestroy()
323     {
324         isDestroying = true;
325
326         if (attachedToHand != null)
327         {
328             attachedToHand.DetachObject(this.gameObject, false);
329             attachedToHand.skeleton.BlendToSkeleton(0.1f);
330         }
331
332         if (highlightHolder != null)
333             Destroy(highlightHolder);
334     }

```

```

333
334
335     protected virtual void OnDisable()
336     {
337         isDestroying = true;
338
339         if (attachedToHand != null)
340         {
341             attachedToHand.ForceHoverUnlock();
342         }
343
344         if (highlightHolder != null)
345             Destroy(highlightHolder);
346     }
347 }
348

```

9.5.8. Player.cs

```

1  //===== Copyright (c) Valve Corporation, All rights reserved.
   //=====
2  //
3  // Purpose: Player interface used to query HMD transforms and VR hands
4  //
5  //
   =====
6
7  using UnityEngine;
8  using System.Collections;
9  using System.Collections.Generic;
10
11 namespace Valve.VR.InteractionSystem
12 {
13     //
   -----
14     // Singleton representing the local VR player/user, with methods for
   // getting
15     // the player's hands, head, tracking origin, and guesses for various
   // properties.
16     //
   -----
17     public class Player : MonoBehaviour
18     {
19         [Tooltip( "Virtual transform corresponding to the meatspace tracking
   // origin. Devices are tracked relative to this." )]
20         public Transform trackingOriginTransform;
21
22         [Tooltip( "List of possible transforms for the head/HMD, including
   // the no-SteamVR fallback camera." )]

```



```

23     public Transform[] hmdTransforms;
24
25     [Tooltip( "List of possible Hands, including no-SteamVR fallback
                Hands." )]
26     public Hand[] hands;
27
28     [Tooltip( "Reference to the physics collider that follows the player'
                s HMD position." )]
29     public Collider headCollider;
30
31     [Tooltip( "These objects are enabled when SteamVR is available" )]
32     public GameObject rigSteamVR;
33
34     [Tooltip( "These objects are enabled when SteamVR is not available,
                or when the user toggles out of VR" )]
35     public GameObject rig2DFallback;
36
37     [Tooltip( "The audio listener for this player" )]
38     public Transform audioListener;
39
40         [Tooltip("This action lets you know when the player has placed
                the headset on their head")]
41         public SteamVR_Action_Boolean headsetOnHead = SteamVR_Input.
            GetBooleanAction("HeadsetOnHead");
42
43     public bool allowToggleTo2D = true;
44
45
46     //-----
47     // Singleton instance of the Player. Only one can exist at a time.
48     //-----
49     private static Player _instance;
50     public static Player instance
51     {
52         get
53         {
54             if ( _instance == null )
55             {
56                 _instance = FindObjectOfType<Player>();
57             }
58             return _instance;
59         }
60     }
61
62
63     //-----
64     // Get the number of active Hands.
65     //-----
66     public int handCount
67     {
68         get
69         {
70             int count = 0;
71             for ( int i = 0; i < hands.Length; i++ )

```

```

72         {
73             if ( hands[i].gameObject.activeInHierarchy )
74             {
75                 count++;
76             }
77         }
78         return count;
79     }
80 }
81
82
83 //-----
84 // Get the i-th active Hand.
85 //
86 // i - Zero-based index of the active Hand to get
87 //-----
88 public Hand GetHand( int i )
89 {
90     for ( int j = 0; j < hands.Length; j++ )
91     {
92         if ( !hands[j].gameObject.activeInHierarchy )
93         {
94             continue;
95         }
96
97         if ( i > 0 )
98         {
99             i--;
100             continue;
101         }
102
103         return hands[j];
104     }
105
106     return null;
107 }
108
109
110 //-----
111 public Hand leftHand
112 {
113     get
114     {
115         for ( int j = 0; j < hands.Length; j++ )
116         {
117             if ( !hands[j].gameObject.activeInHierarchy )
118             {
119                 continue;
120             }
121
122             if ( hands[j].handType != SteamVR_Input_Sources.LeftHand )
123             {
124                 continue;
125             }

```

```

126         return hands[j];
127     }
128
129     return null;
130 }
131
132 }
133
134
135 //-----
136 public Hand rightHand
137 {
138     get
139     {
140         for ( int j = 0; j < hands.Length; j++ )
141         {
142             if ( !hands[j].gameObject.activeInHierarchy )
143             {
144                 continue;
145             }
146
147             if ( hands[j].handType != SteamVR_Input_Sources.RightHand )
148             {
149                 continue;
150             }
151
152             return hands[j];
153         }
154
155         return null;
156     }
157 }
158
159 //-----
160 // Get Player scale. Assumes it is scaled equally on all axes.
161 //-----
162
163 public float scale
164 {
165     get
166     {
167         return transform.lossyScale.x;
168     }
169 }
170
171
172 //-----
173 // Get the HMD transform. This might return the fallback camera
174 // transform if SteamVR is unavailable or disabled.
175 //-----
176 public Transform hmdTransform
177 {
178     get
179     {

```

```

179         if (hmdTransforms != null)
180         {
181             for (int i = 0; i < hmdTransforms.Length; i++)
182             {
183                 if (hmdTransforms[i].gameObject.activeInHierarchy
184                     )
185                     return hmdTransforms[i];
186             }
187         }
188         return null;
189     }
190
191
192     //-----
193     // Height of the eyes above the ground - useful for estimating player
194     // height.
195     //-----
196     public float eyeHeight
197     {
198         get
199         {
200             Transform hmd = hmdTransform;
201             if ( hmd )
202             {
203                 Vector3 eyeOffset = Vector3.Project( hmd.position -
204                     trackingOriginTransform.position, trackingOriginTransform.
205                     up );
206                 return eyeOffset.magnitude / trackingOriginTransform.lossyScale
207                     .x;
208             }
209             return 0.0f;
210         }
211     }
212
213     //-----
214     // Guess for the world-space position of the player's feet, directly
215     // beneath the HMD.
216     //-----
217     public Vector3 feetPositionGuess
218     {
219         get
220         {
221             Transform hmd = hmdTransform;
222             if ( hmd )
223             {
224                 return trackingOriginTransform.position + Vector3.
225                     ProjectOnPlane( hmd.position - trackingOriginTransform.
226                     position, trackingOriginTransform.up );
227             }
228             return trackingOriginTransform.position;
229         }
230     }
231
232

```

```

225
226
227 //-----
228 // Guess for the world-space direction of the player's hips/torso.
    This is effectively just the gaze direction projected onto the
    floor plane.
229 //-----
230 public Vector3 bodyDirectionGuess
231 {
232     get
233     {
234         Transform hmd = hmdTransform;
235         if ( hmd )
236         {
237             Vector3 direction = Vector3.ProjectOnPlane( hmd.forward,
                trackingOriginTransform.up );
238             if ( Vector3.Dot( hmd.up, trackingOriginTransform.up ) < 0.0f )
239             {
240                 // The HMD is upside-down. Either
241                 // -The player is bending over backwards
242                 // -The player is bent over looking through their legs
243                 direction = -direction;
244             }
245             return direction;
246         }
247         return trackingOriginTransform.forward;
248     }
249 }
250
251
252 //-----
253 private void Awake()
254 {
255     if ( trackingOriginTransform == null )
256     {
257         trackingOriginTransform = this.transform;
258     }
259 }
260
261
262 //-----
263 private IEnumerator Start()
264 {
265     _instance = this;
266
267     while (SteamVR.initializedState == SteamVR.InitializedStates.
        None || SteamVR.initializedState == SteamVR.
        InitializedStates.Initializing)
        yield return null;
268
269     if ( SteamVR.instance != null )
270     {
271         ActivateRig( rigSteamVR );
272     }
273

```

```

274         else
275         {
276             #if !HIDE_DEBUG_UI
277                 ActivateRig( rig2DFallback );
278             #endif
279         }
280     }
281
282     protected virtual void Update()
283     {
284         if (SteamVR.initializedState != SteamVR.InitializedStates.
                InitializeSuccess)
285             return;
286
287         if (headsetOnHead != null)
288         {
289             if (headsetOnHead.GetStateDown(SteamVR_Input_Sources.Head
                ))
290             {
291                 Debug.Log("<b>SteamVR Interaction System</b> Headset
                    placed on head");
292             }
293             else if (headsetOnHead.GetStateUp(SteamVR_Input_Sources.
                Head))
294             {
295                 Debug.Log("<b>SteamVR Interaction System</b> Headset
                    removed");
296             }
297         }
298     }
299
300     //-----
301     void OnDrawGizmos()
302     {
303         if ( this != instance )
304         {
305             return;
306         }
307
308         //NOTE: These gizmo icons don't work in the plugin since the icons
                need to exist in a specific "Gizmos"
309         //     folder in your Asset tree. These icons are included under
                Core/Icons. Moving them into a
310         //     "Gizmos" folder should make them work again.
311
312         Gizmos.color = Color.white;
313         Gizmos.DrawIcon( feetPositionGuess, "vr_interaction_system_feet.png
                " );
314
315         Gizmos.color = Color.cyan;
316         Gizmos.DrawLine( feetPositionGuess, feetPositionGuess +
                trackingOriginTransform.up * eyeHeight );
317
318         // Body direction arrow

```

```

319     Gizmos.color = Color.blue;
320     Vector3 bodyDirection = bodyDirectionGuess;
321     Vector3 bodyDirectionTangent = Vector3.Cross(
322         trackingOriginTransform.up, bodyDirection );
323     Vector3 startForward = feetPositionGuess + trackingOriginTransform.
324         up * eyeHeight * 0.75f;
325     Vector3 endForward = startForward + bodyDirection * 0.33f;
326     Gizmos.DrawLine( startForward, endForward );
327     Gizmos.DrawLine( endForward, endForward - 0.033f * ( bodyDirection
328         + bodyDirectionTangent ) );
329     Gizmos.DrawLine( endForward, endForward - 0.033f * ( bodyDirection
330         - bodyDirectionTangent ) );
331
332     Gizmos.color = Color.red;
333     int count = handCount;
334     for ( int i = 0; i < count; i++ )
335     {
336         Hand hand = GetHand( i );
337
338         if ( hand.handType == SteamVR_Input_Sources.LeftHand )
339         {
340             Gizmos.DrawIcon( hand.transform.position, "
341                 vr_interaction_system_left_hand.png" );
342         }
343         else if ( hand.handType == SteamVR_Input_Sources.RightHand )
344         {
345             Gizmos.DrawIcon( hand.transform.position, "
346                 vr_interaction_system_right_hand.png" );
347         }
348         else
349         {
350             /*
351             Hand.HandType guessHandType = hand.currentHandType;
352
353             if ( guessHandType == Hand.HandType.Left )
354             {
355                 Gizmos.DrawIcon( hand.transform.position, "
356                     vr_interaction_system_left_hand_question.png" );
357             }
358             else if ( guessHandType == Hand.HandType.Right )
359             {
360                 Gizmos.DrawIcon( hand.transform.position, "
361                     vr_interaction_system_right_hand_question.png" );
362             }
363             else
364             {
365                 Gizmos.DrawIcon( hand.transform.position, "
366                     vr_interaction_system_unknown_hand.png" );
367             }
368             */
369         }
370     }
371 }

```

```

364
365 //-----
366 public void Draw2DDebug()
367 {
368     if ( !allowToggleTo2D )
369         return;
370
371     if ( !SteamVR.active )
372         return;
373
374     int width = 100;
375     int height = 25;
376     int left = Screen.width / 2 - width / 2;
377     int top = Screen.height - height - 10;
378
379     string text = ( rigSteamVR.activeSelf ) ? "2D Debug" : "VR";
380
381     if ( GUI.Button( new Rect( left, top, width, height ), text ) )
382     {
383         if ( rigSteamVR.activeSelf )
384         {
385             ActivateRig( rig2DFallback );
386         }
387         else
388         {
389             ActivateRig( rigSteamVR );
390         }
391     }
392 }
393
394
395 //-----
396 private void ActivateRig( GameObject rig )
397 {
398     rigSteamVR.SetActive( rig == rigSteamVR );
399     rig2DFallback.SetActive( rig == rig2DFallback );
400
401     if ( audioListener )
402     {
403         audioListener.transform.parent = hmdTransform;
404         audioListener.transform.localPosition = Vector3.zero;
405         audioListener.transform.localRotation = Quaternion.identity;
406     }
407 }
408
409
410 //-----
411 public void PlayerShotSelf()
412 {
413     //Do something appropriate here
414 }
415 }
416 }

```


9.5.9. SteamVR_ActivateActionSetOnLoad.cs

```

1  //===== Copyright (c) Valve Corporation, All rights reserved.
   //=====
2
3  using UnityEngine;
4  using System.Collections;
5
6  namespace Valve.VR
7  {
8      /// <summary>
9      /// Automatically activates an action set on Start() and deactivates
10     the set on OnDestroy(). Optionally deactivating all other sets as
11     well.
12     /// </summary>
13     public class SteamVR_ActivateActionSetOnLoad : MonoBehaviour
14     {
15         public SteamVR_ActionSet actionSet = SteamVR_Input.GetActionSet("
            default");
16
17         public SteamVR_Input_Sources forSources = SteamVR_Input_Sources.
            Any;
18
19         public bool disableAllOtherActionSets = false;
20
21         public bool activateOnStart = true;
22         public bool deactivateOnDestroy = true;
23
24         private void Start()
25         {
26             if (actionSet != null && activateOnStart)
27             {
28                 //Debug.Log(string.Format("[SteamVR] Activating {0}
                action set.", actionSet.fullPath));
29                 actionSet.Activate(forSources, 0,
                    disableAllOtherActionSets);
30             }
31         }
32
33         private void OnDestroy()
34         {
35             if (actionSet != null && deactivateOnDestroy)
36             {
37                 //Debug.Log(string.Format("[SteamVR] Deactivating {0}
                action set.", actionSet.fullPath));
38                 actionSet.Deactivate(forSources);
39             }
40         }
41     }

```

9.5.10. SteamVR_ Behaviour.cs

```

1  using System;
2  using System.Collections;
3  using System.Collections.Generic;
4  using System.Linq;
5  using System.Text;
6  using UnityEngine;
7
8  #if UNITY_2017_2_OR_NEWER
9      using UnityEngine.XR;
10 #else
11 using XRSettings = UnityEngine.VR.VRSettings;
12 using XRDevice = UnityEngine.VR.VRDevice;
13 #endif
14
15 namespace Valve.VR
16 {
17     public class SteamVR_Behaviour : MonoBehaviour
18     {
19         private const string openVRDeviceName = "OpenVR";
20         public static bool forcingInitialization = false;
21
22         private static SteamVR_Behaviour _instance;
23         public static SteamVR_Behaviour instance
24         {
25             get
26             {
27                 if (_instance == null)
28                 {
29                     Initialize(false);
30                 }
31
32                 return _instance;
33             }
34         }
35
36         public bool initializeSteamVROnAwake = true;
37
38         public bool doNotDestroy = true;
39
40         [HideInInspector]
41         public SteamVR_Render steamvr_render;
42
43
44         private static bool initializing = false;
45         public static void Initialize(bool forceUnityVRToOpenVR = false)
46         {
47             if (_instance == null && initializing == false)
48             {
49                 initializing = true;
50                 GameObject steamVRObject = null;
51
52                 if (forceUnityVRToOpenVR)

```

```

53         forcingInitialization = true;
54
55         SteamVR_Render renderInstance = GameObject.
56             FindObjectOfType<SteamVR_Render>();
57         if (renderInstance != null)
58             steamVRObject = renderInstance.gameObject;
59
60         SteamVR_Behaviour behaviourInstance = GameObject.
61             FindObjectOfType<SteamVR_Behaviour>();
62         if (behaviourInstance != null)
63             steamVRObject = behaviourInstance.gameObject;
64
65         if (steamVRObject == null)
66         {
67             GameObject objectInstance = new GameObject("[SteamVR]
68                 ");
69             _instance = objectInstance.AddComponent<
70                 SteamVR_Behaviour>();
71             _instance.steamvr_render = objectInstance.
72                 AddComponent<SteamVR_Render>();
73         }
74         else
75         {
76             behaviourInstance = steamVRObject.GetComponent<
77                 SteamVR_Behaviour>();
78             if (behaviourInstance == null)
79                 behaviourInstance = steamVRObject.AddComponent<
80                     SteamVR_Behaviour>();
81
82             if (renderInstance != null)
83                 behaviourInstance.steamvr_render = renderInstance
84                     ;
85             else
86             {
87                 behaviourInstance.steamvr_render = steamVRObject.
88                     GetComponent<SteamVR_Render>();
89                 if (behaviourInstance.steamvr_render == null)
90                     behaviourInstance.steamvr_render =
91                         steamVRObject.AddComponent<SteamVR_Render
92                             >();
93             }
94
95             _instance = behaviourInstance;
96         }
97
98         if (behaviourInstance != null && behaviourInstance.
99             doNotDestroy)
100             GameObject.DontDestroyOnLoad(behaviourInstance.
101                 transform.root.gameObject);
102
103         initializing = false;
104     }
105 }

```

```

94     protected void Awake()
95     {
96         if (initializeSteamVROnAwake && forcingInitialization ==
97             false)
98             InitializeSteamVR();
99     }
100     public void InitializeSteamVR(bool forceUnityVRToOpenVR = false)
101     {
102         if (forceUnityVRToOpenVR)
103         {
104             forcingInitialization = true;
105
106             if (initializeCoroutine != null)
107                 StopCoroutine(initializeCoroutine);
108
109             if (XRSettings.loadedDeviceName == openVRDeviceName)
110                 EnableOpenVR();
111             else
112                 initializeCoroutine = StartCoroutine(
113                     DoInitializeSteamVR(forceUnityVRToOpenVR));
114         }
115         else
116         {
117             SteamVR.Initialize(false);
118         }
119     }
120     private Coroutine initializeCoroutine;
121
122     #if UNITY_2018_3_OR_NEWER
123     private bool loadedOpenVRDeviceSuccess = false;
124     private IEnumerator DoInitializeSteamVR(bool forceUnityVRToOpenVR
125         = false)
126     {
127         XRDevice.deviceLoaded += XRDevice_deviceLoaded;
128         XRSettings.LoadDeviceByName(openVRDeviceName);
129         while (loadedOpenVRDeviceSuccess == false)
130         {
131             yield return null;
132         }
133         XRDevice.deviceLoaded -= XRDevice_deviceLoaded;
134         EnableOpenVR();
135     }
136     private void XRDevice_deviceLoaded(string deviceName)
137     {
138         if (deviceName == openVRDeviceName)
139         {
140             loadedOpenVRDeviceSuccess = true;
141         }
142         else
143         {

```

```

144         Debug.LogError("<b>[SteamVR]</b> Tried to async load: " +
145             openVRDeviceName + ". Loaded: " + deviceName);
146         loadedOpenVRDeviceSuccess = true; //try anyway
147     }
148 #else
149     private IEnumerator DoInitializeSteamVR(bool forceUnityVRToOpenVR
150         = false)
151     {
152         XRSettings.LoadDeviceByName(openVRDeviceName);
153         yield return null;
154         EnableOpenVR();
155     }
156 #endif
157
158     private void EnableOpenVR()
159     {
160         XRSettings.enabled = true;
161         SteamVR.Initialize(false);
162         initializeCoroutine = null;
163         forcingInitialization = false;
164     }
165
166 #if UNITY_2017_1_OR_NEWER
167     protected void OnEnable()
168     {
169         Application.onBeforeRender += OnBeforeRender;
170         SteamVR_Events.System(EVREventType.VREvent_Quit).Listen(
171             OnQuit);
172     }
173     protected void OnDisable()
174     {
175         Application.onBeforeRender -= OnBeforeRender;
176         SteamVR_Events.System(EVREventType.VREvent_Quit).Remove(
177             OnQuit);
178     }
179     protected void OnBeforeRender()
180     {
181         PreCull();
182     }
183 #else
184     protected void OnEnable()
185     {
186         Camera.onPreCull += OnCameraPreCull;
187         SteamVR_Events.System(EVREventType.VREvent_Quit).Listen(
188             OnQuit);
189     }
190     protected void OnDisable()
191     {
192         Camera.onPreCull -= OnCameraPreCull;
193         SteamVR_Events.System(EVREventType.VREvent_Quit).Remove(
194             OnQuit);
195     }
196     protected void OnCameraPreCull(Camera cam)

```

```

192     {
193         if (!cam.stereoEnabled)
194             return;
195
196         PreCull();
197     }
198 #endif
199
200     protected static int lastFrameCount = -1;
201     protected void PreCull()
202     {
203         // Only update poses on the first camera per frame.
204         if (Time.frameCount != lastFrameCount)
205         {
206             lastFrameCount = Time.frameCount;
207
208             SteamVR_Input.OnPreCull();
209         }
210     }
211
212     protected void FixedUpdate()
213     {
214         SteamVR_Input.FixedUpdate();
215     }
216
217     protected void LateUpdate()
218     {
219         SteamVR_Input.LateUpdate();
220     }
221
222     protected void Update()
223     {
224         SteamVR_Input.Update();
225     }
226
227     protected void OnQuit(VREvent_t vrEvent)
228     {
229 #if UNITY_EDITOR
230         UnityEditor.EditorApplication.isPlaying = false;
231 #else
232         Application.Quit();
233 #endif
234     }
235 }
236

```

9.5.11. SteamVR_Behaviour_Pose.cs

```

1 //===== Copyright (c) Valve Corporation, All rights reserved.
2 //=====
3 using System;

```

```

4  using System.Threading;
5  using UnityEngine;
6  using UnityEngine.Events;
7  using Valve.VR;
8
9  namespace Valve.VR
10 {
11     /// <summary>
12     /// This component simplifies the use of Pose actions. Adding it to a
13     /// gameobject will auto set that transform's position and rotation
14     /// every update to match the pose.
15     /// Advanced velocity estimation is handled through a buffer of the
16     /// last 30 updates.
17     /// </summary>
18     public class SteamVR_Behaviour_Pose : MonoBehaviour
19     {
20         public SteamVR_Action_Pose poseAction = SteamVR_Input.GetAction<
21             SteamVR_Action_Pose>("Pose");
22
23         [Tooltip("The device this action should apply to. Any if the
24             action is not device specific.")]
25         public SteamVR_Input_Sources inputSource;
26
27         [Tooltip("If not set, relative to parent")]
28         public Transform origin;
29
30         /// <summary>Returns whether or not the current pose is in a
31         /// valid state</summary>
32         public bool isValid { get { return poseAction[inputSource].
33             poseIsValid; } }
34
35         /// <summary>Returns whether or not the pose action is bound and
36         /// able to be updated</summary>
37         public bool isActive { get { return poseAction[inputSource].
38             active; } }
39
40         /// <summary>This Unity event will fire whenever the position or
41         /// rotation of this transform is updated.</summary>
42         public SteamVR_Behaviour_PoseEvent onTransformUpdated;
43
44         /// <summary>This Unity event will fire whenever the position or
45         /// rotation of this transform is changed.</summary>
46         public SteamVR_Behaviour_PoseEvent onTransformChanged;
47
48         /// <summary>This Unity event will fire whenever the device is
49         /// connected or disconnected</summary>
50         public SteamVR_Behaviour_Pose_ConnectedChangedEvent
51             onConnectedChanged;
52
53         /// <summary>This Unity event will fire whenever the device's
54         /// tracking state changes</summary>
55         public SteamVR_Behaviour_Pose_TrackingChangedEvent
56             onTrackingChanged;

```

```

43
44     /// <summary>This Unity event will fire whenever the device's
45     deviceIndex changes</summary>
46     public SteamVR_Behaviour_Pose_DeviceIndexChangedEvent
47         onDeviceIndexChanged;
48
49     /// <summary>This C# event will fire whenever the position or
50     rotation of this transform is updated.</summary>
51     public UpdateHandler onTransformUpdatedEvent;
52
53     /// <summary>This C# event will fire whenever the position or
54     rotation of this transform is changed.</summary>
55     public ChangeHandler onTransformChangedEvent;
56
57     /// <summary>This C# event will fire whenever the device is
58     connected or disconnected</summary>
59     public DeviceConnectedChangeHandler onConnectedChangedEvent;
60
61     /// <summary>This C# event will fire whenever the device's
62     tracking state changes</summary>
63     public TrackingChangeHandler onTrackingChangedEvent;
64
65     /// <summary>This C# event will fire whenever the device's
66     deviceIndex changes</summary>
67     public DeviceIndexChangedHandler onDeviceIndexChangedEvent;
68
69     [Tooltip("Can be disabled to stop broadcasting bound device
70     status changes")]
71     public bool broadcastDeviceChanges = true;
72
73     protected int deviceIndex = -1;
74
75     protected SteamVR_HistoryBuffer historyBuffer = new
76         SteamVR_HistoryBuffer(30);
77
78     protected virtual void Start()
79     {
80         if (poseAction == null)
81         {
82             Debug.LogError("<b>[SteamVR]</b> No pose action set for
83             this component");
84             return;
85         }
86
87         CheckDeviceIndex();
88
89         if (origin == null)
90             origin = this.transform.parent;
91     }
92
93     protected virtual void OnEnable()

```



```

87         {
88             SteamVR.Initialize();
89
90             if (poseAction != null)
91             {
92                 poseAction[inputSource].onUpdate +=
93                     SteamVR_Behaviour_Pose_OnUpdate;
94                 poseAction[inputSource].onDeviceConnectedChanged +=
95                     OnDeviceConnectedChanged;
96                 poseAction[inputSource].onTrackingChanged +=
97                     OnTrackingChanged;
98                 poseAction[inputSource].onChange +=
99                     SteamVR_Behaviour_Pose_OnChange;
100             }
101         }
102
103     protected virtual void OnDisable()
104     {
105         if (poseAction != null)
106         {
107             poseAction[inputSource].onUpdate -=
108                 SteamVR_Behaviour_Pose_OnUpdate;
109             poseAction[inputSource].onDeviceConnectedChanged -=
110                 OnDeviceConnectedChanged;
111             poseAction[inputSource].onTrackingChanged -=
112                 OnTrackingChanged;
113             poseAction[inputSource].onChange -=
114                 SteamVR_Behaviour_Pose_OnChange;
115         }
116
117         historyBuffer.Clear();
118     }
119
120     private void SteamVR_Behaviour_Pose_OnUpdate(SteamVR_Action_Pose
121         fromAction, SteamVR_Input_Sources fromSource)
122     {
123         UpdateHistoryBuffer();
124
125         UpdateTransform();
126
127         if (onTransformUpdated != null)
128             onTransformUpdated.Invoke(this, inputSource);
129         if (onTransformUpdatedEvent != null)
130             onTransformUpdatedEvent.Invoke(this, inputSource);
131     }
132
133     protected virtual void UpdateTransform()
134     {
135         CheckDeviceIndex();
136
137         if (origin != null)
138         {
139             transform.position = origin.transform.TransformPoint(
140                 poseAction[inputSource].localPosition);

```

```

130         transform.rotation = origin.rotation * poseAction[
131             inputSource].localRotation;
132     }
133     else
134     {
135         transform.localPosition = poseAction[inputSource].
136             localPosition;
137         transform.localRotation = poseAction[inputSource].
138             localRotation;
139     }
140 }
141
142 private void SteamVR_Behaviour_Pose_OnChange(SteamVR_Action_Pose
143     fromAction, SteamVR_Input_Sources fromSource)
144 {
145     if (onTransformChanged != null)
146         onTransformChanged.Invoke(this, fromSource);
147     if (onTransformChangedEvent != null)
148         onTransformChangedEvent.Invoke(this, fromSource);
149 }
150
151 protected virtual void OnDeviceConnectedChanged(
152     SteamVR_Action_Pose changedAction, SteamVR_Input_Sources
153     changedSource, bool connected)
154 {
155     CheckDeviceIndex();
156
157     if (onConnectedChanged != null)
158         onConnectedChanged.Invoke(this, inputSource, connected);
159     if (onConnectedChangedEvent != null)
160         onConnectedChangedEvent.Invoke(this, inputSource,
161             connected);
162 }
163
164 protected virtual void OnTrackingChanged(SteamVR_Action_Pose
165     changedAction, SteamVR_Input_Sources changedSource,
166     ETrackingResult trackingChanged)
167 {
168     if (onTrackingChanged != null)
169         onTrackingChanged.Invoke(this, inputSource,
170             trackingChanged);
171     if (onTrackingChangedEvent != null)
172         onTrackingChangedEvent.Invoke(this, inputSource,
173             trackingChanged);
174 }
175
176 protected virtual void CheckDeviceIndex()
177 {
178     if (poseAction[inputSource].active && poseAction[inputSource]
179         .deviceIsConnected)
180     {
181         int currentDeviceIndex = (int)poseAction[inputSource].
182             trackedDeviceIndex;
183     }
184 }

```

```

171         if (deviceIndex != currentDeviceIndex)
172         {
173             deviceIndex = currentDeviceIndex;
174
175             if (broadcastDeviceChanges)
176             {
177                 this.gameObject.BroadcastMessage("SetInputSource"
178                     , inputSource, SendMessageOptions.
179                     DontRequireReceiver);
180
181                 this.gameObject.BroadcastMessage("SetDeviceIndex"
182                     , deviceIndex, SendMessageOptions.
183                     DontRequireReceiver);
184             }
185
186             if (onDeviceIndexChanged != null)
187                 onDeviceIndexChanged.Invoke(this, inputSource,
188                     deviceIndex);
189             if (onDeviceIndexChangedEvent != null)
190                 onDeviceIndexChangedEvent.Invoke(this,
191                     inputSource, deviceIndex);
192         }
193     }
194
195     /// <summary>
196     /// Returns the device index for the device bound to the pose.
197     /// </summary>
198     public int GetDeviceIndex()
199     {
200         if (deviceIndex == -1)
201             CheckDeviceIndex();
202
203         return deviceIndex;
204     }
205
206     /// <summary>Returns the current velocity of the pose (as of the
207     last update)</summary>
208     public Vector3 GetVelocity()
209     {
210         return poseAction[inputSource].velocity;
211     }
212
213     /// <summary>Returns the current angular velocity of the pose (as
214     of the last update)</summary>
215     public Vector3 GetAngularVelocity()
216     {
217         return poseAction[inputSource].angularVelocity;
218     }
219
220     /// <summary>Returns the velocities of the pose at the time
221     specified. Can predict in the future or return past values.</
222     summary>
223     public bool GetVelocitiesAtTimeOffset(float secondsFromNow, out
224         Vector3 velocity, out Vector3 angularVelocity)

```

```

214     {
215         return poseAction[inputSource].GetVelocitiesAtTimeOffset(
216             secondsFromNow, out velocity, out angularVelocity);
217     }
218
219     /// <summary>Uses previously recorded values to find the peak
220     speed of the pose and returns the corresponding velocity and
221     angular velocity</summary>
222     public void GetEstimatedPeakVelocities(out Vector3 velocity, out
223         Vector3 angularVelocity)
224     {
225         int top = historyBuffer.GetTopVelocity(10, 1);
226
227         historyBuffer.GetAverageVelocities(out velocity, out
228             angularVelocity, 2, top);
229     }
230
231     protected int lastFrameUpdated;
232     protected void UpdateHistoryBuffer()
233     {
234         int currentFrame = Time.frameCount;
235         if (lastFrameUpdated != currentFrame)
236         {
237             historyBuffer.Update(poseAction[inputSource].
238                 localPosition, poseAction[inputSource].localRotation,
239                 poseAction[inputSource].velocity, poseAction[
240                     inputSource].angularVelocity);
241             lastFrameUpdated = currentFrame;
242         }
243     }
244
245     /// <summary>
246     /// Gets the localized name of the device that the action
247     /// corresponds to.
248     /// </summary>
249     /// <param name="localizedParts">
250     /// <list type="bullet">
251     /// <item><description>VRInputString_Hand - Which hand the origin
252     is in. E.g. "Left Hand"</description></item>
253     /// <item><description>VRInputString_ControllerType - What kind
254     of controller the user has in that hand.E.g. "Vive Controller
255     "</description></item>
256     /// <item><description>VRInputString_InputSource - What part of
257     that controller is the origin. E.g. "Trackpad"</description
258     ></item>
259     /// <item><description>VRInputString_All - All of the above. E.g.
260     "Left Hand Vive Controller Trackpad"</description></item>
261     /// </list>
262     /// </param>
263     public string GetLocalizedName(params EVRInputStringBits []
264         localizedParts)
265     {
266         if (poseAction != null)

```

```

251         return poseAction.GetLocalizedOriginPart(inputSource,
252             localizedParts);
253     return null;
254 }
255
256 public delegate void ActiveChangeHandler(SteamVR_Behaviour_Pose
257     fromAction, SteamVR_Input_Sources fromSource, bool active);
258 public delegate void ChangeHandler(SteamVR_Behaviour_Pose
259     fromAction, SteamVR_Input_Sources fromSource);
260 public delegate void UpdateHandler(SteamVR_Behaviour_Pose
261     fromAction, SteamVR_Input_Sources fromSource);
262 public delegate void TrackingChangeHandler(SteamVR_Behaviour_Pose
263     fromAction, SteamVR_Input_Sources fromSource,
264     ETrackingResult trackingState);
265 public delegate void ValidPoseChangeHandler(
266     SteamVR_Behaviour_Pose fromAction, SteamVR_Input_Sources
267     fromSource, bool validPose);
268 public delegate void DeviceConnectedChangeHandler(
269     SteamVR_Behaviour_Pose fromAction, SteamVR_Input_Sources
270     fromSource, bool deviceConnected);
271 public delegate void DeviceIndexChangedHandler(
272     SteamVR_Behaviour_Pose fromAction, SteamVR_Input_Sources
273     fromSource, int newDeviceIndex);
274 }
275 }

```

9.5.12. SteamVR_Fade.cs

```

1  //===== Copyright (c) Valve Corporation, All rights reserved.
2  //=====
3  //
4  // Purpose: CameraFade script adapted to work with SteamVR.
5  //
6  // Usage: Add to your top level SteamVR_Camera (the one with
7  //         ApplyDistortion
8  //         checked) and drag a reference to this component into
9  //         SteamVR_Camera
10 //
11 //         RenderComponents list. Then call the static helper function
12 //         SteamVR_Fade.Start with the desired color and duration.
13 //         Use a duration of zero to set the start color.
14 //
15 // Example: Fade down from black over one second.
16 //         SteamVR_Fade.Start(Color.black, 0);
17 //         SteamVR_Fade.Start(Color.clear, 1);
18 //
19 // Note: This component is provided to fade out a single camera layer's
20 //         scene view. If instead you want to fade the entire view, use:
21 //         SteamVR_Fade.View(Color.black, 1);
22 //         (Does not affect the game view, however.)
23 //
24 //
25 //=====

```

```

21
22 using UnityEngine;
23 using Valve.VR;
24
25 namespace Valve.VR
26 {
27     public class SteamVR_Fade : MonoBehaviour
28     {
29         private Color currentColor = new Color(0, 0, 0, 0); // default
30         private Color targetColor = new Color(0, 0, 0, 0); // default
31         private Color deltaColor = new Color(0, 0, 0, 0); // the delta-
32         color is basically the "speed / second" at which the current
33         color should change
34         private bool fadeOverlay = false;
35
36         static public void Start(Color newColor, float duration, bool
37         fadeOverlay = false)
38         {
39             SteamVR_Events.Fade.Send(newColor, duration, fadeOverlay);
40         }
41
42         static public void View(Color newColor, float duration)
43         {
44             var compositor = OpenVR.Compositor;
45             if (compositor != null)
46                 compositor.FadeToColor(duration, newColor.r, newColor.g,
47                 newColor.b, newColor.a, false);
48         }
49
50 #if TEST_FADE_VIEW
51     void Update()
52     {
53         if (Input.GetKeyDown(KeyCode.Space))
54         {
55             SteamVR_Fade.View(Color.black, 0);
56             SteamVR_Fade.View(Color.clear, 1);
57         }
58     }
59 #endif
60
61     public void OnStartFade(Color newColor, float duration, bool
62     fadeOverlay)
63     {
64         if (duration > 0.0f)
65         {
66             targetColor = newColor;
67             deltaColor = (targetColor - currentColor) / duration;
68         }
69         else
70         {
71             currentColor = newColor;

```

```

67         }
68     }
69
70     static Material fadeMaterial = null;
71     static int fadeMaterialColorID = -1;
72
73     void OnEnable()
74     {
75         if (fadeMaterial == null)
76         {
77             fadeMaterial = new Material(Shader.Find("Custom/
78                 SteamVR_Fade"));
79             fadeMaterialColorID = Shader.PropertyToID("fadeColor");
80         }
81
82         SteamVR_Events.Fade.Listen(OnStartFade);
83         SteamVR_Events.FadeReady.Send();
84     }
85
86     void OnDisable()
87     {
88         SteamVR_Events.Fade.Remove(OnStartFade);
89     }
90
91     void OnPostRender()
92     {
93         if (currentColor != targetColor)
94         {
95             // if the difference between the current alpha and the
96             // desired alpha is smaller than delta-alpha * deltaTime
97             // , then we're pretty much done fading:
98             if (Mathf.Abs(currentColor.a - targetColor.a) < Mathf.Abs
99                 (deltaColor.a) * Time.deltaTime)
100             {
101                 currentColor = targetColor;
102                 deltaColor = new Color(0, 0, 0, 0);
103             }
104             else
105             {
106                 currentColor += deltaColor * Time.deltaTime;
107             }
108
109             if (fadeOverlay)
110             {
111                 var overlay = SteamVR_Overlay.instance;
112                 if (overlay != null)
113                 {
114                     overlay.alpha = 1.0f - currentColor.a;
115                 }
116             }
117
118             if (currentColor.a > 0 && fadeMaterial)
119             {

```

```

117         fadeMaterial.SetColor(fadeMaterialColorID, currentColor);
118         fadeMaterial.SetPass(0);
119         GL.Begin(GL.QUADS);
120
121         GL.Vertex3(-1, -1, 0);
122         GL.Vertex3(1, -1, 0);
123         GL.Vertex3(1, 1, 0);
124         GL.Vertex3(-1, 1, 0);
125         GL.End();
126     }
127 }
128 }
129 }

```

9.5.13. SteamVR_Overlay.cs

```

1  //===== Copyright (c) Valve Corporation, All rights reserved.
2  //=====
3  // Purpose: Displays 2d content on a large virtual screen.
4  //
5  //
6  //=====
7
8  using UnityEngine;
9  using System.Collections;
10 using Valve.VR;
11
12 namespace Valve.VR
13 {
14     public class SteamVR_Overlay : MonoBehaviour
15     {
16         public Texture texture;
17         public bool curved = true;
18         public bool antialias = true;
19         public bool highquality = true;
20
21         [Tooltip("Size of overlay view.")]
22         public float scale = 3.0f;
23
24         [Tooltip("Distance from surface.")]
25         public float distance = 1.25f;
26
27         [Tooltip("Opacity"), Range(0.0f, 1.0f)]
28         public float alpha = 1.0f;
29
30         public Vector4 uvOffset = new Vector4(0, 0, 1, 1);
31         public Vector2 mouseScale = new Vector2(1, 1);
32         public Vector2 curvedRange = new Vector2(1, 2);

```



```

33     public VROverlayInputMethod inputMethod = VROverlayInputMethod.
34         None;
35
36     static public SteamVR_Overlay instance { get; private set; }
37
38     static public string key { get { return "unity:" + Application.
39         companyName + "." + Application.productName; } }
40
41     private ulong handle = OpenVR.k_ulOverlayHandleInvalid;
42
43     void OnEnable()
44     {
45         var overlay = OpenVR.Overlay;
46         if (overlay != null)
47         {
48             var error = overlay.CreateOverlay(key, gameObject.name,
49                 ref handle);
50             if (error != EVROverlayError.None)
51             {
52                 Debug.Log("<b>[SteamVR]</b> " + overlay.
53                     GetOverlayErrorNameFromEnum(error));
54                 enabled = false;
55                 return;
56             }
57         }
58
59         SteamVR_Overlay.instance = this;
60     }
61
62     void OnDisable()
63     {
64         if (handle != OpenVR.k_ulOverlayHandleInvalid)
65         {
66             var overlay = OpenVR.Overlay;
67             if (overlay != null)
68             {
69                 overlay.DestroyOverlay(handle);
70             }
71
72             handle = OpenVR.k_ulOverlayHandleInvalid;
73         }
74
75         SteamVR_Overlay.instance = null;
76     }
77
78     public void UpdateOverlay()
79     {
80         var overlay = OpenVR.Overlay;
81         if (overlay == null)
82             return;
83
84         if (texture != null)
85         {
86             var error = overlay.ShowOverlay(handle);

```

```

83         if (error == EVROverlayError.InvalidHandle || error ==
84             EVROverlayError.UnknownOverlay)
85         {
86             if (overlay.FindOverlay(key, ref handle) !=
87                 EVROverlayError.None)
88                 return;
89         }
90
91         var tex = new Texture_t();
92         tex.handle = texture.GetNativeTexturePtr();
93         tex.eType = SteamVR.instance.textureType;
94         tex.eColorSpace = EColorSpace.Auto;
95         overlay.SetOverlayTexture(handle, ref tex);
96
97         overlay.SetOverlayAlpha(handle, alpha);
98         overlay.SetOverlayWidthInMeters(handle, scale);
99         overlay.SetOverlayAutoCurveDistanceRangeInMeters(handle,
100             curvedRange.x, curvedRange.y);
101
102         var textureBounds = new VRTextureBounds_t();
103         textureBounds.uMin = (0 + uvOffset.x) * uvOffset.z;
104         textureBounds.vMin = (1 + uvOffset.y) * uvOffset.w;
105         textureBounds.uMax = (1 + uvOffset.x) * uvOffset.z;
106         textureBounds.vMax = (0 + uvOffset.y) * uvOffset.w;
107         overlay.SetOverlayTextureBounds(handle, ref textureBounds
108             );
109
110         var vecMouseScale = new HmdVector2_t();
111         vecMouseScale.v0 = mouseScale.x;
112         vecMouseScale.v1 = mouseScale.y;
113         overlay.SetOverlayMouseScale(handle, ref vecMouseScale);
114
115         var vrcam = SteamVR_Render.Top();
116         if (vrcam != null && vrcam.origin != null)
117         {
118             var offset = new SteamVR_Utils.RigidTransform(vrcam.
119                 origin, transform);
120             offset.pos.x /= vrcam.origin.localScale.x;
121             offset.pos.y /= vrcam.origin.localScale.y;
122             offset.pos.z /= vrcam.origin.localScale.z;
123
124             offset.pos.z += distance;
125
126             var t = offset.ToHmdMatrix34();
127             overlay.SetOverlayTransformAbsolute(handle, SteamVR.
128                 settings.trackingSpace, ref t);
129         }
130
131         overlay.SetOverlayInputMethod(handle, inputMethod);
132
133         if (curved || antialias)
134             highquality = true;
135
136         if (highquality)

```

```

131         {
132             overlay.SetHighQualityOverlay(handle);
133             overlay.SetOverlayFlag(handle, VROverlayFlags.Curved,
134                                     curved);
135             overlay.SetOverlayFlag(handle, VROverlayFlags.RGSS4X,
136                                     antialias);
137         }
138         else if (overlay.GetHighQualityOverlay() == handle)
139         {
140             overlay.SetHighQualityOverlay(OpenVR.
141                                     k_ulOverlayHandleInvalid);
142         }
143     }
144     else
145     {
146         overlay.HideOverlay(handle);
147     }
148 }
149
150 public bool PollNextEvent(ref VREvent_t pEvent)
151 {
152     var overlay = OpenVR.Overlay;
153     if (overlay == null)
154         return false;
155
156     var size = (uint)System.Runtime.InteropServices.Marshal.
157         SizeOf(typeof(Valve.VR.VREvent_t));
158     return overlay.PollNextOverlayEvent(handle, ref pEvent, size)
159         ;
160 }
161
162 public struct IntersectionResults
163 {
164     public Vector3 point;
165     public Vector3 normal;
166     public Vector2 UVs;
167     public float distance;
168 }
169
170 public bool ComputeIntersection(Vector3 source, Vector3 direction
171 , ref IntersectionResults results)
172 {
173     var overlay = OpenVR.Overlay;
174     if (overlay == null)
175         return false;
176
177     var input = new VROverlayIntersectionParams_t();
178     input.eOrigin = SteamVR.settings.trackingSpace;
179     input.vSource.v0 = source.x;
180     input.vSource.v1 = source.y;
181     input.vSource.v2 = -source.z;
182     input.vDirection.v0 = direction.x;
183     input.vDirection.v1 = direction.y;
184     input.vDirection.v2 = -direction.z;

```

```

179         var output = new VROverlayIntersectionResults_t();
180         if (!overlay.ComputeOverlayIntersection(handle, ref input,
181             ref output))
182             return false;
183
184         results.point = new Vector3(output.vPoint.v0, output.vPoint.
185             v1, -output.vPoint.v2);
186         results.normal = new Vector3(output.vNormal.v0, output.
187             vNormal.v1, -output.vNormal.v2);
188         results.UVs = new Vector2(output.vUVs.v0, output.vUVs.v1);
189         results.distance = output.fDistance;
190         return true;
191     }
192 }
193 }

```

9.5.14. SteamVR_Render.cs

```

1  //===== Copyright (c) Valve Corporation, All rights reserved.
2  //=====
3  //
4  // Purpose: Handles rendering of all SteamVR_Cameras
5  //
6  //=====
7
8  using UnityEngine;
9  using System.Collections;
10 using Valve.VR;
11
12 namespace Valve.VR
13 {
14     public class SteamVR_Render : MonoBehaviour
15     {
16         public SteamVR_ExternalCamera externalCamera;
17         public string externalCameraConfigPath = "externalcamera.cfg";
18
19         public static EVREye eye { get; private set; }
20
21         public static SteamVR_Render instance { get { return
22             SteamVR_Behaviour.instance.steamvr_render; } }
23
24         static private bool isQuitting;
25         void OnApplicationQuit()
26         {
27             isQuitting = true;
28             SteamVR.SafeDispose();
29         }
30     }
31 }

```

```
30     static public void Add(SteamVR_Camera vrcam)
31     {
32         if (!isQuitting)
33             instance.AddInternal(vrcam);
34     }
35
36     static public void Remove(SteamVR_Camera vrcam)
37     {
38         if (!isQuitting && instance != null)
39             instance.RemoveInternal(vrcam);
40     }
41
42     static public SteamVR_Camera Top()
43     {
44         if (!isQuitting)
45             return instance.TopInternal();
46
47         return null;
48     }
49
50     private SteamVR_Camera[] cameras = new SteamVR_Camera[0];
51
52     void AddInternal(SteamVR_Camera vrcam)
53     {
54         var camera = vrcam.GetComponent<Camera>();
55         var length = cameras.Length;
56         var sorted = new SteamVR_Camera[length + 1];
57         int insert = 0;
58         for (int i = 0; i < length; i++)
59         {
60             var c = cameras[i].GetComponent<Camera>();
61             if (i == insert && c.depth > camera.depth)
62                 sorted[insert++] = vrcam;
63
64             sorted[insert++] = cameras[i];
65         }
66         if (insert == length)
67             sorted[insert] = vrcam;
68
69         cameras = sorted;
70     }
71
72     void RemoveInternal(SteamVR_Camera vrcam)
73     {
74         var length = cameras.Length;
75         int count = 0;
76         for (int i = 0; i < length; i++)
77         {
78             var c = cameras[i];
79             if (c == vrcam)
80                 ++count;
81         }
82         if (count == 0)
83             return;
```

```

84
85     var sorted = new SteamVR_Camera[length - count];
86     int insert = 0;
87     for (int i = 0; i < length; i++)
88     {
89         var c = cameras[i];
90         if (c != vrcam)
91             sorted[insert++] = c;
92     }
93
94     cameras = sorted;
95 }
96
97 SteamVR_Camera TopInternal()
98 {
99     if (cameras.Length > 0)
100         return cameras[cameras.Length - 1];
101
102     return null;
103 }
104
105 public TrackedDevicePose_t[] poses = new TrackedDevicePose_t[
106     OpenVR.k_unMaxTrackedDeviceCount];
107 public TrackedDevicePose_t[] gamePoses = new TrackedDevicePose_t
108     [0];
109
110 static private bool _pauseRendering;
111 static public bool pauseRendering
112 {
113     get { return _pauseRendering; }
114     set
115     {
116         _pauseRendering = value;
117
118         var compositor = OpenVR.Compositor;
119         if (compositor != null)
120             compositor.SuspendRendering(value);
121     }
122 }
123
124 private WaitForEndOfFrame waitForEndOfFrame = new
125     WaitForEndOfFrame();
126
127 private IEnumerator RenderLoop()
128 {
129     while (Application.isPlaying)
130     {
131         yield return waitForEndOfFrame;
132
133         if (pauseRendering)
134             continue;
135
136         var compositor = OpenVR.Compositor;
137         if (compositor != null)

```

```

135         {
136             if (!compositor.CanRenderScene())
137                 continue;
138
139             compositor.SetTrackingSpace(SteamVR.settings.
                trackingSpace);
140         }
141
142         var overlay = SteamVR_Overlay.instance;
143         if (overlay != null)
144             overlay.UpdateOverlay();
145
146         RenderExternalCamera();
147     }
148 }
149
150 void RenderExternalCamera()
151 {
152     if (externalCamera == null)
153         return;
154
155     if (!externalCamera.gameObject.activeInHierarchy)
156         return;
157
158     var frameSkip = (int)Mathf.Max(externalCamera.config.
        frameSkip, 0.0f);
159     if (Time.frameCount % (frameSkip + 1) != 0)
160         return;
161
162     // Keep external camera relative to the most relevant vr
        camera.
163     externalCamera.AttachToCamera(TopInternal());
164
165     externalCamera.RenderNear();
166     externalCamera.RenderFar();
167 }
168
169 float sceneResolutionScale = 1.0f, timeScale = 1.0f;
170
171 private void OnInputFocus(bool hasFocus)
172 {
173     if (SteamVR.active == false)
174         return;
175
176     if (hasFocus)
177     {
178         if (SteamVR.settings.pauseGameWhenDashboardVisible)
179         {
180             Time.timeScale = timeScale;
181         }
182
183         SteamVR_Camera.sceneResolutionScale =
            sceneResolutionScale;
184     }

```

```

185         else
186         {
187             if (SteamVR.settings.pauseGameWhenDashboardVisible)
188             {
189                 timeScale = Time.timeScale;
190                 Time.timeScale = 0.0f;
191             }
192
193             sceneResolutionScale = SteamVR_Camera.
194                 sceneResolutionScale;
195             SteamVR_Camera.sceneResolutionScale = 0.5f;
196         }
197     }
198
199     private string GetScreenshotFilename(uint screenshotHandle,
200         EVRScreenshotPropertyFileNames screenshotPropertyFilename)
201     {
202         var error = EVRScreenshotError.None;
203         var capacity = OpenVR.Screenshots.
204             GetScreenshotPropertyFilename(screenshotHandle,
205                 screenshotPropertyFilename, null, 0, ref error);
206         if (error != EVRScreenshotError.None && error !=
207             EVRScreenshotError.BufferTooSmall)
208             return null;
209         if (capacity > 1)
210         {
211             var result = new System.Text.StringBuilder((int)capacity)
212                 ;
213             OpenVR.Screenshots.GetScreenshotPropertyFilename(
214                 screenshotHandle, screenshotPropertyFilename, result,
215                 capacity, ref error);
216             if (error != EVRScreenshotError.None)
217                 return null;
218             return result.ToString();
219         }
220         return null;
221     }
222
223     private void OnRequestScreenshot(VREvent_t vrEvent)
224     {
225         var screenshotHandle = vrEvent.data.screenshot.handle;
226         var screenshotType = (EVRScreenshotType)vrEvent.data.
227             screenshot.type;
228
229         if (screenshotType == EVRScreenshotType.StereoPanorama)
230         {
231             string previewFilename = GetScreenshotFilename(
232                 screenshotHandle, EVRScreenshotPropertyFileNames.
233                 Preview);
234             string VRFilename = GetScreenshotFilename(
235                 screenshotHandle, EVRScreenshotPropertyFileNames.VR);
236
237             if (previewFilename == null || VRFilename == null)
238                 return;

```



```

227         // Do the stereo panorama screenshot
228         // Figure out where the view is
229         GameObject screenshotPosition = new GameObject("
230             screenshotPosition");
231         screenshotPosition.transform.position = SteamVR_Render.
232             Top().transform.position;
233         screenshotPosition.transform.rotation = SteamVR_Render.
234             Top().transform.rotation;
235         screenshotPosition.transform.localScale = SteamVR_Render.
236             Top().transform.lossyScale;
237         SteamVR_Utils.TakeStereoScreenshot(screenshotHandle,
238             screenshotPosition, 32, 0.064f, ref previewFilename,
239             ref VRFilename);
240
241         // and submit it
242         OpenVR.Screenshots.SubmitScreenshot(screenshotHandle,
243             screenshotType, previewFilename, VRFilename);
244     }
245 }
246
247 private EVRScreenshotType[] screenshotTypes = new
248     EVRScreenshotType[] { EVRScreenshotType.StereoPanorama };
249
250 private void OnEnable()
251 {
252     StartCoroutine(RenderLoop());
253     SteamVR_Events.InputFocus.Listen(OnInputFocus);
254     SteamVR_Events.System(EVREventType.VREvent_RequestScreenshot)
255         .Listen(OnRequestScreenshot);
256
257 #if UNITY_2017_1_OR_NEWER
258     Application.onBeforeRender += OnBeforeRender;
259 #else
260     Camera.onPreCull += OnCameraPreCull;
261 #endif
262
263     if (SteamVR.initializedState == SteamVR.InitializedStates.
264         InitializeSuccess)
265         OpenVR.Screenshots.HookScreenshot(screenshotTypes);
266     else
267         SteamVR_Events.Initialized.AddListener(
268             OnSteamVRInitialized);
269 }
270
271 private void OnSteamVRInitialized(bool success)
272 {
273     if (success)
274         OpenVR.Screenshots.HookScreenshot(screenshotTypes);
275 }
276
277 private void OnDisable()
278 {
279     StopAllCoroutines();

```

```

270         SteamVR_Events.InputFocus.Remove(OnInputFocus);
271         SteamVR_Events.System(EVREventType.VREvent_RequestScreenshot)
            .Remove(OnRequestScreenshot);
272
273     #if UNITY_2017_1_OR_NEWER
274         Application.onBeforeRender -= OnBeforeRender;
275     #else
276         Camera.onPreCull -= OnCameraPreCull;
277     #endif
278
279     if (SteamVR.initializedState != SteamVR.InitializedStates.
        InitializeSuccess)
280         SteamVR_Events.Initialized.RemoveListener(
            OnSteamVRInitialized);
281 }
282
283 private void Awake()
284 {
285     if (externalCamera == null && System.IO.File.Exists(
        externalCameraConfigPath))
286     {
287         var prefab = Resources.Load<GameObject>("
            SteamVR_ExternalCamera");
288         var instance = Instantiate(prefab);
289         instance.gameObject.name = "External Camera";
290
291         externalCamera = instance.transform.GetChild(0).
            GetComponent<SteamVR_ExternalCamera>();
292         externalCamera.configPath = externalCameraConfigPath;
293         externalCamera.ReadConfig();
294     }
295 }
296
297 public void UpdatePoses()
298 {
299     var compositor = OpenVR.Compositor;
300     if (compositor != null)
301     {
302         compositor.GetLastPoses(poses, gamePoses);
303         SteamVR_Events.NewPoses.Send(poses);
304         SteamVR_Events.NewPosesApplied.Send();
305     }
306 }
307
308 #if UNITY_2017_1_OR_NEWER
309     void OnBeforeRender()
310     {
311         if (SteamVR.active == false)
312             return;
313
314         if (SteamVR.settings.IsPoseUpdateMode(SteamVR.UpdateModes.
            OnPreCull))
315         {
316             UpdatePoses();

```

```

317         }
318     }
319 #else
320     void OnCameraPreCull(Camera cam)
321     {
322         if (SteamVR.active == false)
323             return;
324
325 #if UNITY_2017_1_OR_NEWER
326     if (cam.cameraType != CameraType.VR)
327         return;
328 #else
329         //custom code
330         if (!cam.stereoEnabled) //if not main camera (stereoEnabled
                                //isn't perfect, but it is the fast/easiest way to check
                                //this in Unity 5.4)
331         {
332             return;
333         }
334 #endif
335         // Only update poses on the first camera per frame.
336         if (Time.frameCount != lastFrameCount)
337         {
338             lastFrameCount = Time.frameCount;
339
340             if (SteamVR.settings.IsPoseUpdateMode(SteamVR.UpdateModes
341                 .OnPreCull))
342             {
343                 UpdatePoses();
344             }
345         }
346         static int lastFrameCount = -1;
347 #endif
348
349     void Update()
350     {
351         if (SteamVR.active == false)
352             return;
353
354         UpdatePoses();
355
356         // Dispatch any OpenVR events.
357         var system = OpenVR.System;
358         if (system != null)
359         {
360             var vrEvent = new VREvent_t();
361             var size = (uint)System.Runtime.InteropServices.Marshal
362                 .SizeOf(typeof(VREvent_t));
363             for (int i = 0; i < 64; i++)
364             {
365                 if (!system.PollNextEvent(ref vrEvent, size))
366                     break;

```

```

367         switch ((EVREventType)vrEvent.eventType)
368         {
369             case EVREventType.VREvent_InputFocusCaptured: //
370                 another app has taken focus (likely dashboard
371                 )
372                 if (vrEvent.data.process.oldPid == 0)
373                 {
374                     SteamVR_Events.InputFocus.Send(false);
375                 }
376                 break;
377             case EVREventType.VREvent_InputFocusReleased: //
378                 that app has released input focus
379                 if (vrEvent.data.process.pid == 0)
380                 {
381                     SteamVR_Events.InputFocus.Send(true);
382                 }
383                 break;
384             case EVREventType.VREvent_ShowRenderModels:
385                 SteamVR_Events.HideRenderModels.Send(false);
386                 break;
387             case EVREventType.VREvent_HideRenderModels:
388                 SteamVR_Events.HideRenderModels.Send(true);
389                 break;
390             default:
391                 SteamVR_Events.System((EVREventType)vrEvent.
392                     eventType).Send(vrEvent);
393                 break;
394         }
395     }
396 }
397
398 // Ensure various settings to minimize latency.
399 Application.targetFrameRate = -1;
400 Application.runInBackground = true; // don't require
401     companion window focus
402 QualitySettings.maxQueuedFrames = -1;
403 QualitySettings.vSyncCount = 0; // this applies to the
404     companion window
405
406 if (SteamVR.settings.lockPhysicsUpdateRateToRenderFrequency
407     && Time.timeScale > 0.0f)
408 {
409     var vr = SteamVR.instance;
410     if (vr != null)
411     {
412         var timing = new Compositor_FrameTiming();
413         timing.m_nSize = (uint)System.Runtime.InteropServices.
414             Marshal.SizeOf(typeof(Compositor_FrameTiming));
415         vr.compositor.GetFrameTiming(ref timing, 0);
416
417         Time.fixedDeltaTime = Time.timeScale / vr.
418             hmd_DisplayFrequency;
419     }
420 }

```

```

412     }
413 }
414 }

```

9.5.15. Teleport.cs

```

1  //===== Copyright (c) Valve Corporation, All rights reserved.
   //=====
2  //
3  // Purpose: Handles all the teleport logic
4  //
5  //
   //=====
6
7  using UnityEngine;
8  using UnityEngine.Events;
9  using System.Collections;
10
11 namespace Valve.VR.InteractionSystem
12 {
13     //
   -----
14     public class Teleport : MonoBehaviour
15     {
16         public SteamVR_Action_Boolean teleportAction = SteamVR_Input.
            GetAction<SteamVR_Action_Boolean>("Teleport");
17
18         public LayerMask traceLayerMask;
19         public LayerMask floorFixupTraceLayerMask;
20         public float floorFixupMaximumTraceDistance = 1.0f;
21         public Material areaVisibleMaterial;
22         public Material areaLockedMaterial;
23         public Material areaHighlightedMaterial;
24         public Material pointVisibleMaterial;
25         public Material pointLockedMaterial;
26         public Material pointHighlightedMaterial;
27         public Transform destinationReticuleTransform;
28         public Transform invalidReticuleTransform;
29         public GameObject playAreaPreviewCorner;
30         public GameObject playAreaPreviewSide;
31         public Color pointerValidColor;
32         public Color pointerInvalidColor;
33         public Color pointerLockedColor;
34         public bool showPlayAreaMarker = true;
35
36         public float teleportFadeTime = 0.1f;
37         public float meshFadeTime = 0.2f;
38
39         public float arcDistance = 10.0f;
40

```

```

41 [Header( "Effects" )]
42 public Transform onActivateObjectTransform;
43 public Transform onDeactivateObjectTransform;
44 public float activateObjectTime = 1.0f;
45 public float deactivateObjectTime = 1.0f;
46
47 [Header( "Audio Sources" )]
48 public AudioSource pointerAudioSource;
49 public AudioSource loopingAudioSource;
50 public AudioSource headAudioSource;
51 public AudioSource reticleAudioSource;
52
53 [Header( "Sounds" )]
54 public AudioClip teleportSound;
55 public AudioClip pointerStartSound;
56 public AudioClip pointerLoopSound;
57 public AudioClip pointerStopSound;
58 public AudioClip goodHighlightSound;
59 public AudioClip badHighlightSound;
60
61 [Header( "Debug" )]
62 public bool debugFloor = false;
63 public bool showOffsetReticle = false;
64 public Transform offsetReticleTransform;
65 public MeshRenderer floorDebugSphere;
66 public LineRenderer floorDebugLine;
67
68 private LineRenderer pointerLineRenderer;
69 private GameObject teleportPointerObject;
70 private Transform pointerStartTransform;
71 private Hand pointerHand = null;
72 private Player player = null;
73 private TeleportArc teleportArc = null;
74
75 private bool visible = false;
76
77 private TeleportMarkerBase[] teleportMarkers;
78 private TeleportMarkerBase pointedAtTeleportMarker;
79 private TeleportMarkerBase teleportingToMarker;
80 private Vector3 pointedAtPosition;
81 private Vector3 prevPointedAtPosition;
82 private bool teleporting = false;
83 private float currentFadeTime = 0.0f;
84
85 private float meshAlphaPercent = 1.0f;
86 private float pointerShowStartTime = 0.0f;
87 private float pointerHideStartTime = 0.0f;
88 private bool meshFading = false;
89 private float fullTintAlpha;
90
91 private float invalidReticleMinScale = 0.2f;
92 private float invalidReticleMaxScale = 1.0f;
93 private float invalidReticleMinScaleDistance = 0.4f;
94 private float invalidReticleMaxScaleDistance = 2.0f;

```

```

95     private Vector3 invalidReticleScale = Vector3.one;
96     private Quaternion invalidReticleTargetRotation = Quaternion.identity
97         ;
98
99     private Transform playAreaPreviewTransform;
100    private Transform[] playAreaPreviewCorners;
101    private Transform[] playAreaPreviewSides;
102
103
104    private float loopingAudioMaxVolume = 0.0f;
105
106    private Coroutine hintCoroutine = null;
107
108    private bool originalHoverLockState = false;
109    private Interactable originalHoveringInteractable = null;
110    private AllowTeleportWhileAttachedToHand allowTeleportWhileAttached =
111        null;
112
113    private Vector3 startingFeetOffset = Vector3.zero;
114    private bool movedFeetFarEnough = false;
115
116    SteamVR_Events.Action chaperoneInfoInitializedAction;
117
118    // Events
119
120    public static SteamVR_Events.Event< float > ChangeScene = new
121        SteamVR_Events.Event< float >();
122    public static SteamVR_Events.Action< float > ChangeSceneAction(
123        UnityAction< float > action ) { return new SteamVR_Events.Action<
124            float >( ChangeScene, action ); }
125
126    public static SteamVR_Events.Event< TeleportMarkerBase > Player = new
127        SteamVR_Events.Event< TeleportMarkerBase >();
128    public static SteamVR_Events.Action< TeleportMarkerBase >
129        PlayerAction( UnityAction< TeleportMarkerBase > action ) { return
130            new SteamVR_Events.Action< TeleportMarkerBase >( Player, action
131            ); }
132
133    public static SteamVR_Events.Event< TeleportMarkerBase > PlayerPre =
134        new SteamVR_Events.Event< TeleportMarkerBase >();
135    public static SteamVR_Events.Action< TeleportMarkerBase >
136        PlayerPreAction( UnityAction< TeleportMarkerBase > action ) {
137        return new SteamVR_Events.Action< TeleportMarkerBase >( PlayerPre
138        , action ); }
139
140    //-----
141    private static Teleport _instance;
142    public static Teleport instance
143    {
144        get
145        {
146            if ( _instance == null )
147            {
148                _instance = GameObject.FindObjectOfType<Teleport>();
149            }
150        }
151    }

```

```

136         return _instance;
137     }
138 }
139
140
141
142 //-----
143 void Awake()
144 {
145     _instance = this;
146
147     chaperoneInfoInitializedAction = ChaperoneInfo.InitializedAction(
148         OnChaperoneInfoInitialized );
149
150     pointerLineRenderer = GetComponentInChildren<LineRenderer>();
151     teleportPointerObject = pointerLineRenderer.gameObject;
152
153     int tintColorID = Shader.PropertyToID( "_TintColor" );
154     fullTintAlpha = pointVisibleMaterial.GetColor( tintColorID ).a;
155
156     teleportArc = GetComponent<TeleportArc>();
157     teleportArc.traceLayerMask = traceLayerMask;
158
159     loopingAudioMaxVolume = loopingAudioSource.volume;
160
161     playAreaPreviewCorner.SetActive( false );
162     playAreaPreviewSide.SetActive( false );
163
164     float invalidReticleStartingScale = invalidReticleTransform.
165         localScale.x;
166     invalidReticleMinScale *= invalidReticleStartingScale;
167     invalidReticleMaxScale *= invalidReticleStartingScale;
168 }
169
170 //-----
171 void Start()
172 {
173     teleportMarkers = GameObject.FindObjectsOfType<
174         TeleportMarkerBase>();
175
176     HidePointer();
177
178     player = InteractionSystem.Player.instance;
179
180     if ( player == null )
181     {
182         Debug.LogError("<b>[SteamVR Interaction]</b> Teleport: No Player
183             instance found in map.");
184         Destroy( this.gameObject );
185         return;
186     }
187
188     CheckForSpawnPoint();

```



```
186
187     Invoke( "ShowTeleportHint", 5.0f );
188 }
189
190
191 //-----
192 void OnEnable()
193 {
194     chaperoneInfoInitializedAction.enabled = true;
195     OnChaperoneInfoInitialized(); // In case it's already initialized
196 }
197
198
199 //-----
200 void OnDisable()
201 {
202     chaperoneInfoInitializedAction.enabled = false;
203     HidePointer();
204 }
205
206
207 //-----
208 private void CheckForSpawnPoint()
209 {
210     foreach ( TeleportMarkerBase teleportMarker in teleportMarkers )
211     {
212         TeleportPoint teleportPoint = teleportMarker as TeleportPoint;
213         if ( teleportPoint && teleportPoint.playerSpawnPoint )
214         {
215             teleportingToMarker = teleportMarker;
216             TeleportPlayer();
217             break;
218         }
219     }
220 }
221
222
223 //-----
224 public void HideTeleportPointer()
225 {
226     if ( pointerHand != null )
227     {
228         HidePointer();
229     }
230 }
231
232
233 //-----
234 void Update()
235 {
236     Hand oldPointerHand = pointerHand;
237     Hand newPointerHand = null;
238
239     foreach ( Hand hand in player.hands )
```

```

240 {
241     if ( visible )
242     {
243         if ( WasTeleportButtonReleased( hand ) )
244         {
245             if ( pointerHand == hand ) //This is the pointer hand
246             {
247                 TryTeleportPlayer();
248             }
249         }
250     }
251
252     if ( WasTeleportButtonPressed( hand ) )
253     {
254         newPointerHand = hand;
255     }
256 }
257
258 //If something is attached to the hand that is preventing teleport
259 if ( allowTeleportWhileAttached && !allowTeleportWhileAttached.
    teleportAllowed )
260 {
261     HidePointer();
262 }
263 else
264 {
265     if ( !visible && newPointerHand != null )
266     {
267         //Begin showing the pointer
268         ShowPointer( newPointerHand, oldPointerHand );
269     }
270     else if ( visible )
271     {
272         if ( newPointerHand == null && !IsTeleportButtonDown(
            pointerHand ) )
273         {
274             //Hide the pointer
275             HidePointer();
276         }
277         else if ( newPointerHand != null )
278         {
279             //Move the pointer to a new hand
280             ShowPointer( newPointerHand, oldPointerHand );
281         }
282     }
283 }
284
285 if ( visible )
286 {
287     UpdatePointer();
288
289     if ( meshFading )
290     {
291         UpdateTeleportColors();

```

```

292     }
293
294     if ( onActivateObjectTransform.gameObject.activeSelf && Time.time
        - pointerShowStartTime > activateObjectTime )
295     {
296         onActivateObjectTransform.gameObject.SetActive( false );
297     }
298 }
299 else
300 {
301     if ( onDeactivateObjectTransform.gameObject.activeSelf && Time.
        time - pointerHideStartTime > deactivateObjectTime )
302     {
303         onDeactivateObjectTransform.gameObject.SetActive( false );
304     }
305 }
306 }
307
308
309 //-----
310 private void UpdatePointer()
311 {
312     Vector3 pointerStart = pointerStartTransform.position;
313     Vector3 pointerEnd;
314     Vector3 pointerDir = pointerStartTransform.forward;
315     bool hitSomething = false;
316     bool showPlayAreaPreview = false;
317     Vector3 playerFeetOffset = player.trackingOriginTransform.position
        - player.feetPositionGuess;
318
319     Vector3 arcVelocity = pointerDir * arcDistance;
320
321     TeleportMarkerBase hitTeleportMarker = null;
322
323     //Check pointer angle
324     float dotUp = Vector3.Dot( pointerDir, Vector3.up );
325     float dotForward = Vector3.Dot( pointerDir, player.hmdTransform.
        forward );
326     bool pointerAtBadAngle = false;
327     if ( ( dotForward > 0 && dotUp > 0.75f ) || ( dotForward < 0.0f &&
        dotUp > 0.5f ) )
328     {
329         pointerAtBadAngle = true;
330     }
331
332     //Trace to see if the pointer hit anything
333     RaycastHit hitInfo;
334     teleportArc.SetArcData( pointerStart, arcVelocity, true,
        pointerAtBadAngle );
335     if ( teleportArc.DrawArc( out hitInfo ) )
336     {
337         hitSomething = true;
338         hitTeleportMarker = hitInfo.collider.GetComponentInParent<
            TeleportMarkerBase>();

```

```

339     }
340
341     if ( pointerAtBadAngle )
342     {
343         hitTeleportMarker = null;
344     }
345
346     HighlightSelected( hitTeleportMarker );
347
348     if ( hitTeleportMarker != null ) //Hit a teleport marker
349     {
350         if ( hitTeleportMarker.locked )
351         {
352             teleportArc.SetColor( pointerLockedColor );
353 #if (UNITY_5_4)
354             pointerLineRenderer.SetColors( pointerLockedColor,
355                                             pointerLockedColor );
356 #else
357             pointerLineRenderer.startColor = pointerLockedColor;
358             pointerLineRenderer.endColor = pointerLockedColor;
359 #endif
360             destinationReticleTransform.gameObject.SetActive( false );
361         }
362         else
363         {
364             teleportArc.SetColor( pointerValidColor );
365 #if (UNITY_5_4)
366             pointerLineRenderer.SetColors( pointerValidColor,
367                                             pointerValidColor );
368 #else
369             pointerLineRenderer.startColor = pointerValidColor;
370             pointerLineRenderer.endColor = pointerValidColor;
371 #endif
372             destinationReticleTransform.gameObject.SetActive(
373                 hitTeleportMarker.showReticle );
374         }
375
376         offsetReticleTransform.gameObject.SetActive( true );
377
378         invalidReticleTransform.gameObject.SetActive( false );
379
380         pointedAtTeleportMarker = hitTeleportMarker;
381         pointedAtPosition = hitInfo.point;
382
383         if ( showPlayAreaMarker )
384         {
385             //Show the play area marker if this is a teleport area
386             TeleportArea teleportArea = pointedAtTeleportMarker as
387                 TeleportArea;
388             if ( teleportArea != null && !teleportArea.locked &&
389                 playAreaPreviewTransform != null )
390             {
391                 Vector3 offsetToUse = playerFeetOffset;

```

```

388         //Adjust the actual offset to prevent the play area marker
           from moving too much
389         if ( !movedFeetFarEnough )
390         {
391             float distanceFromStartingOffset = Vector3.Distance(
                   playerFeetOffset, startingFeetOffset );
392             if ( distanceFromStartingOffset < 0.1f )
393             {
394                 offsetToUse = startingFeetOffset;
395             }
396             else if ( distanceFromStartingOffset < 0.4f )
397             {
398                 offsetToUse = Vector3.Lerp( startingFeetOffset,
                   playerFeetOffset, ( distanceFromStartingOffset - 0.1f
                   ) / 0.3f );
399             }
400             else
401             {
402                 movedFeetFarEnough = true;
403             }
404         }
405
406         playAreaPreviewTransform.position = pointedAtPosition +
           offsetToUse;
407
408         showPlayAreaPreview = true;
409     }
410 }
411
412 pointerEnd = hitInfo.point;
413 }
414 else //Hit neither
415 {
416     destinationReticleTransform.gameObject.SetActive( false );
417     offsetReticleTransform.gameObject.SetActive( false );
418
419     teleportArc.SetColor( pointerInvalidColor );
420 #if (UNITY_5_4)
421     pointerLineRenderer.SetColors( pointerInvalidColor,
           pointerInvalidColor );
422 #else
423     pointerLineRenderer.startColor = pointerInvalidColor;
424     pointerLineRenderer.endColor = pointerInvalidColor;
425 #endif
426     invalidReticleTransform.gameObject.SetActive( !pointerAtBadAngle
           );
427
428     //Orient the invalid reticle to the normal of the trace hit point
429     Vector3 normalToUse = hitInfo.normal;
430     float angle = Vector3.Angle( hitInfo.normal, Vector3.up );
431     if ( angle < 15.0f )
432     {
433         normalToUse = Vector3.up;
434     }

```

```

435     invalidReticleTargetRotation = Quaternion.FromToRotation( Vector3
436         .up, normalToUse );
437
438     invalidReticleTransform.rotation = Quaternion.Slerp(
439         invalidReticleTransform.rotation,
440         invalidReticleTargetRotation, 0.1f );
441
442     //Scale the invalid reticle based on the distance from the player
443     float distanceFromPlayer = Vector3.Distance( hitInfo.point,
444         player.hmdTransform.position );
445     float invalidReticleCurrentScale = Util.RemapNumberClamped(
446         distanceFromPlayer, invalidReticleMinScaleDistance,
447         invalidReticleMaxScaleDistance, invalidReticleMinScale,
448         invalidReticleMaxScale );
449     invalidReticleScale.x = invalidReticleCurrentScale;
450     invalidReticleScale.y = invalidReticleCurrentScale;
451     invalidReticleScale.z = invalidReticleCurrentScale;
452     invalidReticleTransform.transform.localScale =
453         invalidReticleScale;
454
455     pointedAtTeleportMarker = null;
456
457     if ( hitSomething )
458     {
459         pointerEnd = hitInfo.point;
460     }
461     else
462     {
463         pointerEnd = teleportArc.GetArcPositionAtTime( teleportArc.
464             arcDuration );
465     }
466
467     //Debug floor
468     if ( debugFloor )
469     {
470         floorDebugSphere.gameObject.SetActive( false );
471         floorDebugLine.gameObject.SetActive( false );
472     }
473
474     if ( playAreaPreviewTransform != null )
475     {
476         playAreaPreviewTransform.gameObject.SetActive(
477             showPlayAreaPreview );
478     }
479
480     if ( !showOffsetReticle )
481     {
482         offsetReticleTransform.gameObject.SetActive( false );
483     }
484
485     destinationReticleTransform.position = pointedAtPosition;
486     invalidReticleTransform.position = pointerEnd;
487     onActivateObjectTransform.position = pointerEnd;
488     onDeactivateObjectTransform.position = pointerEnd;

```

```

479         offsetReticleTransform.position = pointerEnd - playerFeetOffset;
480
481         reticleAudioSource.transform.position = pointedAtPosition;
482
483         pointerLineRenderer.SetPosition( 0, pointerStart );
484         pointerLineRenderer.SetPosition( 1, pointerEnd );
485     }
486
487
488     //-----
489     void FixedUpdate()
490     {
491         if ( !visible )
492         {
493             return;
494         }
495
496         if ( debugFloor )
497         {
498             //Debug floor
499             TeleportArea teleportArea = pointedAtTeleportMarker as
                TeleportArea;
500             if ( teleportArea != null )
501             {
502                 if ( floorFixupMaximumTraceDistance > 0.0f )
503                 {
504                     floorDebugSphere.gameObject.SetActive( true );
505                     floorDebugLine.gameObject.SetActive( true );
506
507                     RaycastHit raycastHit;
508                     Vector3 traceDir = Vector3.down;
509                     traceDir.x = 0.01f;
510                     if ( Physics.Raycast( pointedAtPosition + 0.05f * traceDir,
                        traceDir, out raycastHit, floorFixupMaximumTraceDistance,
                        floorFixupTraceLayerMask ) )
511                     {
512                         floorDebugSphere.transform.position = raycastHit.point;
513                         floorDebugSphere.material.color = Color.green;
514                     #if (UNITY_5_4)
515                         floorDebugLine.SetColors( Color.green, Color.green );
516                     #else
517                         floorDebugLine.startColor = Color.green;
518                         floorDebugLine.endColor = Color.green;
519                     #endif
520
521                     floorDebugLine.SetPosition( 0, pointedAtPosition );
522                     floorDebugLine.SetPosition( 1, raycastHit.point );
523                 }
524                 else
525                 {
526                     Vector3 rayEnd = pointedAtPosition + ( traceDir *
                        floorFixupMaximumTraceDistance );
527                     floorDebugSphere.transform.position = rayEnd;
528                     floorDebugSphere.material.color = Color.red;
529                 }
530             }
531         }
532     }
533
534     #if (UNITY_5_4)
535     void LateUpdate()
536     {
537         if ( !visible )
538         {
539             return;
540         }
541
542         if ( debugFloor )
543         {
544             //Debug floor
545             TeleportArea teleportArea = pointedAtTeleportMarker as
                TeleportArea;
546             if ( teleportArea != null )
547             {
548                 if ( floorFixupMaximumTraceDistance > 0.0f )
549                 {
550                     floorDebugSphere.gameObject.SetActive( true );
551                     floorDebugLine.gameObject.SetActive( true );
552
553                     RaycastHit raycastHit;
554                     Vector3 traceDir = Vector3.down;
555                     traceDir.x = 0.01f;
556                     if ( Physics.Raycast( pointedAtPosition + 0.05f * traceDir,
                        traceDir, out raycastHit, floorFixupMaximumTraceDistance,
                        floorFixupTraceLayerMask ) )
557                     {
558                         floorDebugSphere.transform.position = raycastHit.point;
559                         floorDebugSphere.material.color = Color.green;
560                     }
561                     else
562                     {
563                         Vector3 rayEnd = pointedAtPosition + ( traceDir *
                            floorFixupMaximumTraceDistance );
564                         floorDebugSphere.transform.position = rayEnd;
565                         floorDebugSphere.material.color = Color.red;
566                     }
567                 }
568             }
569         }
570     }
571     #endif
572 }

```

```

529         floorDebugLine.SetColors( Color.red, Color.red );
530 #else
531         floorDebugLine.startColor = Color.red;
532         floorDebugLine.endColor = Color.red;
533 #endif
534         floorDebugLine.SetPosition( 0, pointedAtPosition );
535         floorDebugLine.SetPosition( 1, rayEnd );
536     }
537 }
538 }
539 }
540 }
541
542
543 //-----
544 private void OnChaperoneInfoInitialized()
545 {
546     ChaperoneInfo chaperone = ChaperoneInfo.instance;
547
548     if ( chaperone.initialized && chaperone.roomscale )
549     {
550         //Set up the render model for the play area bounds
551
552         if ( playAreaPreviewTransform == null )
553         {
554             playAreaPreviewTransform = new GameObject( "
                    PlayAreaPreviewTransform" ).transform;
555             playAreaPreviewTransform.parent = transform;
556             Util.ResetTransform( playAreaPreviewTransform );
557
558             playAreaPreviewCorner.SetActive( true );
559             playAreaPreviewCorners = new Transform[4];
560             playAreaPreviewCorners[0] = playAreaPreviewCorner.transform;
561             playAreaPreviewCorners[1] = Instantiate( playAreaPreviewCorners
                    [0] );
562             playAreaPreviewCorners[2] = Instantiate( playAreaPreviewCorners
                    [0] );
563             playAreaPreviewCorners[3] = Instantiate( playAreaPreviewCorners
                    [0] );
564
565             playAreaPreviewCorners[0].transform.parent =
                    playAreaPreviewTransform;
566             playAreaPreviewCorners[1].transform.parent =
                    playAreaPreviewTransform;
567             playAreaPreviewCorners[2].transform.parent =
                    playAreaPreviewTransform;
568             playAreaPreviewCorners[3].transform.parent =
                    playAreaPreviewTransform;
569
570             playAreaPreviewSide.SetActive( true );
571             playAreaPreviewSides = new Transform[4];
572             playAreaPreviewSides[0] = playAreaPreviewSide.transform;
573             playAreaPreviewSides[1] = Instantiate( playAreaPreviewSides[0]
                    );

```



```

574         playAreaPreviewSides[2] = Instantiate( playAreaPreviewSides[0]
575         );
576         playAreaPreviewSides[3] = Instantiate( playAreaPreviewSides[0]
577         );
578
579         playAreaPreviewSides[0].transform.parent =
580         playAreaPreviewTransform;
581         playAreaPreviewSides[1].transform.parent =
582         playAreaPreviewTransform;
583         playAreaPreviewSides[2].transform.parent =
584         playAreaPreviewTransform;
585         playAreaPreviewSides[3].transform.parent =
586         playAreaPreviewTransform;
587     }
588
589     float x = chaperone.playAreaSizeX;
590     float z = chaperone.playAreaSizeZ;
591
592     playAreaPreviewSides[0].localPosition = new Vector3( 0.0f, 0.0f,
593     0.5f * z - 0.25f );
594     playAreaPreviewSides[1].localPosition = new Vector3( 0.0f, 0.0f,
595     -0.5f * z + 0.25f );
596     playAreaPreviewSides[2].localPosition = new Vector3( 0.5f * x -
597     0.25f, 0.0f, 0.0f );
598     playAreaPreviewSides[3].localPosition = new Vector3( -0.5f * x +
599     0.25f, 0.0f, 0.0f );
600
601     playAreaPreviewSides[0].localScale = new Vector3( x - 0.5f, 1.0f,
602     1.0f );
603     playAreaPreviewSides[1].localScale = new Vector3( x - 0.5f, 1.0f,
604     1.0f );
605     playAreaPreviewSides[2].localScale = new Vector3( z - 0.5f, 1.0f,
606     1.0f );
607     playAreaPreviewSides[3].localScale = new Vector3( z - 0.5f, 1.0f,
608     1.0f );
609
610     playAreaPreviewSides[0].localRotation = Quaternion.Euler( 0.0f,
611     0.0f, 0.0f );
612     playAreaPreviewSides[1].localRotation = Quaternion.Euler( 0.0f,
613     180.0f, 0.0f );
614     playAreaPreviewSides[2].localRotation = Quaternion.Euler( 0.0f,
615     90.0f, 0.0f );
616     playAreaPreviewSides[3].localRotation = Quaternion.Euler( 0.0f,
617     270.0f, 0.0f );
618
619     playAreaPreviewCorners[0].localPosition = new Vector3( 0.5f * x -
620     0.25f, 0.0f, 0.5f * z - 0.25f );
621     playAreaPreviewCorners[1].localPosition = new Vector3( 0.5f * x -
622     0.25f, 0.0f, -0.5f * z + 0.25f );
623     playAreaPreviewCorners[2].localPosition = new Vector3( -0.5f * x
624     + 0.25f, 0.0f, -0.5f * z + 0.25f );
625     playAreaPreviewCorners[3].localPosition = new Vector3( -0.5f * x
626     + 0.25f, 0.0f, 0.5f * z - 0.25f );

```

```

606     playAreaPreviewCorners[0].localRotation = Quaternion.Euler( 0.0f,
607         0.0f, 0.0f );
608     playAreaPreviewCorners[1].localRotation = Quaternion.Euler( 0.0f,
609         90.0f, 0.0f );
610     playAreaPreviewCorners[2].localRotation = Quaternion.Euler( 0.0f,
611         180.0f, 0.0f );
612     playAreaPreviewCorners[3].localRotation = Quaternion.Euler( 0.0f,
613         270.0f, 0.0f );
614
615     playAreaPreviewTransform.gameObject.SetActive( false );
616 }
617
618 //-----
619 private void HidePointer()
620 {
621     if ( visible )
622     {
623         pointerHideStartTime = Time.time;
624     }
625
626     visible = false;
627     if ( pointerHand )
628     {
629         if ( ShouldOverrideHoverLock() )
630         {
631             //Restore the original hovering interactable on the hand
632             if ( originalHoverLockState == true )
633             {
634                 pointerHand.HoverLock( originalHoveringInteractable );
635             }
636             else
637             {
638                 pointerHand.HoverUnlock( null );
639             }
640         }
641     }
642
643     //Stop looping sound
644     loopingAudioSource.Stop();
645     PlayAudioClip( pointerAudioSource, pointerStopSound );
646 }
647
648 teleportPointerObject.SetActive( false );
649
650 teleportArc.Hide();
651
652 foreach ( TeleportMarkerBase teleportMarker in teleportMarkers )
653 {
654     if ( teleportMarker != null && teleportMarker.markerActive &&
655         teleportMarker.gameObject != null )
656     {
657         teleportMarker.gameObject.SetActive( false );
658     }
659 }

```

```

655         destinationReticleTransform.gameObject.SetActive( false );
656         invalidReticleTransform.gameObject.SetActive( false );
657         offsetReticleTransform.gameObject.SetActive( false );
658
659
660         if ( playAreaPreviewTransform != null )
661         {
662             playAreaPreviewTransform.gameObject.SetActive( false );
663         }
664
665         if ( onActivateObjectTransform.gameObject.activeSelf )
666         {
667             onActivateObjectTransform.gameObject.SetActive( false );
668         }
669         onDeactivateObjectTransform.gameObject.SetActive( true );
670
671         pointerHand = null;
672     }
673
674
675     //-----
676     private void ShowPointer( Hand newPointerHand, Hand oldPointerHand )
677     {
678         if ( !visible )
679         {
680             pointedAtTeleportMarker = null;
681             pointerShowStartTime = Time.time;
682             visible = true;
683             meshFading = true;
684
685             teleportPointerObject.SetActive( false );
686             teleportArc.Show();
687
688             foreach ( TeleportMarkerBase teleportMarker in teleportMarkers )
689             {
690                 if ( teleportMarker.markerActive && teleportMarker.
691                     ShouldActivate( player.feetPositionGuess ) )
692                 {
693                     teleportMarker.gameObject.SetActive( true );
694                     teleportMarker.Highlight( false );
695                 }
696             }
697
698             startingFeetOffset = player.trackingOriginTransform.position -
699                 player.feetPositionGuess;
700             movedFeetFarEnough = false;
701
702             if ( onDeactivateObjectTransform.gameObject.activeSelf )
703             {
704                 onDeactivateObjectTransform.gameObject.SetActive( false );
705             }
706             onActivateObjectTransform.gameObject.SetActive( true );
707
708             loopingAudioSource.clip = pointerLoopSound;

```

```

707     loopingAudioSource.loop = true;
708     loopingAudioSource.Play();
709     loopingAudioSource.volume = 0.0f;
710 }
711
712
713 if ( oldPointerHand )
714 {
715     if ( ShouldOverrideHoverLock() )
716     {
717         //Restore the original hovering interactable on the hand
718         if ( originalHoverLockState == true )
719         {
720             oldPointerHand.HoverLock( originalHoveringInteractable );
721         }
722         else
723         {
724             oldPointerHand.HoverUnlock( null );
725         }
726     }
727 }
728
729 pointerHand = newPointerHand;
730
731 if ( visible && oldPointerHand != pointerHand )
732 {
733     PlayAudioClip( pointerAudioSource, pointerStartSound );
734 }
735
736 if ( pointerHand )
737 {
738     pointerStartTransform = GetPointerStartTransform( pointerHand );
739
740     if ( pointerHand.currentAttachedObject != null )
741     {
742         allowTeleportWhileAttached = pointerHand.currentAttachedObject.
            GetComponent<AllowTeleportWhileAttachedToHand>();
743     }
744
745     //Keep track of any existing hovering interactable on the hand
746     originalHoverLockState = pointerHand.hoverLocked;
747     originalHoveringInteractable = pointerHand.hoveringInteractable;
748
749     if ( ShouldOverrideHoverLock() )
750     {
751         pointerHand.HoverLock( null );
752     }
753
754     pointerAudioSource.transform.SetParent( pointerStartTransform );
755     pointerAudioSource.transform.localPosition = Vector3.zero;
756
757     loopingAudioSource.transform.SetParent( pointerStartTransform );
758     loopingAudioSource.transform.localPosition = Vector3.zero;
759 }

```

```
760     }
761
762
763     //-----
764     private void UpdateTeleportColors()
765     {
766         float deltaTime = Time.time - pointerShowStartTime;
767         if ( deltaTime > meshFadeTime )
768         {
769             meshAlphaPercent = 1.0f;
770             meshFading = false;
771         }
772         else
773         {
774             meshAlphaPercent = Mathf.Lerp( 0.0f, 1.0f, deltaTime /
775                 meshFadeTime );
776
777             //Tint color for the teleport points
778             foreach ( TeleportMarkerBase teleportMarker in teleportMarkers )
779             {
780                 teleportMarker.SetAlpha( fullTintAlpha * meshAlphaPercent,
781                     meshAlphaPercent );
782             }
783         }
784
785     //-----
786     private void PlayAudioClip( AudioSource source, AudioClip clip )
787     {
788         source.clip = clip;
789         source.Play();
790     }
791
792
793     //-----
794     private void PlayPointerHaptic( bool validLocation )
795     {
796         if ( pointerHand != null )
797         {
798             if ( validLocation )
799             {
800                 pointerHand.TriggerHapticPulse( 800 );
801             }
802             else
803             {
804                 pointerHand.TriggerHapticPulse( 100 );
805             }
806         }
807     }
808
809
810     //-----
811     private void TryTeleportPlayer()
```

```

812 {
813     if ( visible && !teleporting )
814     {
815         if ( pointedAtTeleportMarker != null && pointedAtTeleportMarker.
            locked == false )
816         {
817             //Pointing at an unlocked teleport marker
818             teleportingToMarker = pointedAtTeleportMarker;
819             InitiateTeleportFade();
820
821             CancelTeleportHint();
822         }
823     }
824 }
825
826
827 //-----
828 private void InitiateTeleportFade()
829 {
830     teleporting = true;
831
832     currentFadeTime = teleportFadeTime;
833
834     TeleportPoint teleportPoint = teleportingToMarker as TeleportPoint;
835     if ( teleportPoint != null && teleportPoint.teleportType ==
        TeleportPoint.TeleportPointType.SwitchToNewScene )
836     {
837         currentFadeTime *= 3.0f;
838         Teleport.ChangeScene.Send( currentFadeTime );
839     }
840
841     SteamVR_Fade.Start( Color.clear, 0 );
842     SteamVR_Fade.Start( Color.black, currentFadeTime );
843
844     headAudioSource.transform.SetParent( player.hmdTransform );
845     headAudioSource.transform.localPosition = Vector3.zero;
846     PlayAudioClip( headAudioSource, teleportSound );
847
848     Invoke( "TeleportPlayer", currentFadeTime );
849 }
850
851 //-----
852 private void TeleportPlayer()
853 {
854     teleporting = false;
855
856     Teleport.PlayerPre.Send( pointedAtTeleportMarker );
857
858     SteamVR_Fade.Start( Color.clear, currentFadeTime );
859
860     TeleportPoint teleportPoint = teleportingToMarker as TeleportPoint;
861     Vector3 teleportPosition = pointedAtPosition;
862
863

```

```

864     if ( teleportPoint != null )
865     {
866         teleportPosition = teleportPoint.transform.position;
867
868         //Teleport to a new scene
869         if ( teleportPoint.teleportType == TeleportPoint.
            TeleportPointType.SwitchToNewScene )
870         {
871             teleportPoint.TeleportToScene();
872             return;
873         }
874     }
875
876     // Find the actual floor position below the navigation mesh
877     TeleportArea teleportArea = teleportingToMarker as TeleportArea;
878     if ( teleportArea != null )
879     {
880         if ( floorFixupMaximumTraceDistance > 0.0f )
881         {
882             RaycastHit raycastHit;
883             if ( Physics.Raycast( teleportPosition + 0.05f * Vector3.down,
                Vector3.down, out raycastHit,
                floorFixupMaximumTraceDistance, floorFixupTraceLayerMask )
884             )
885             {
886                 teleportPosition = raycastHit.point;
887             }
888         }
889
890         if ( teleportingToMarker.ShouldMovePlayer() )
891         {
892             Vector3 playerFeetOffset = player.trackingOriginTransform.
                position - player.feetPositionGuess;
893             player.trackingOriginTransform.position = teleportPosition +
                playerFeetOffset;
894         }
895         else
896         {
897             teleportingToMarker.TeleportPlayer( pointedAtPosition );
898         }
899
900         Teleport.Player.Send( pointedAtTeleportMarker );
901     }
902
903
904     //-----
905     private void HighlightSelected( TeleportMarkerBase hitTeleportMarker
906     )
907     {
908         if ( pointedAtTeleportMarker != hitTeleportMarker ) //Pointing at a
            new teleport marker
909         {
910             if ( pointedAtTeleportMarker != null )

```

```

910     {
911         pointedAtTeleportMarker.Highlight( false );
912     }
913
914     if ( hitTeleportMarker != null )
915     {
916         hitTeleportMarker.Highlight( true );
917
918         prevPointedAtPosition = pointedAtPosition;
919         PlayPointerHaptic( !hitTeleportMarker.locked );
920
921         PlayAudioClip( reticleAudioSource, goodHighlightSound );
922
923         loopingAudioSource.volume = loopingAudioMaxVolume;
924     }
925     else if ( pointedAtTeleportMarker != null )
926     {
927         PlayAudioClip( reticleAudioSource, badHighlightSound );
928
929         loopingAudioSource.volume = 0.0f;
930     }
931 }
932 else if ( hitTeleportMarker != null ) //Pointing at the same
    teleport marker
933 {
934     if ( Vector3.Distance( prevPointedAtPosition, pointedAtPosition )
935         > 1.0f )
936     {
937         prevPointedAtPosition = pointedAtPosition;
938         PlayPointerHaptic( !hitTeleportMarker.locked );
939     }
940 }
941
942
943 //-----
944 public void ShowTeleportHint()
945 {
946     CancelTeleportHint();
947
948     hintCoroutine = StartCoroutine( TeleportHintCoroutine() );
949 }
950
951
952 //-----
953 public void CancelTeleportHint()
954 {
955     if ( hintCoroutine != null )
956     {
957         ControllerButtonHints.HideTextHint(player.leftHand,
958             teleportAction);
959         ControllerButtonHints.HideTextHint(player.rightHand,
960             teleportAction);

```



```

960         StopCoroutine( hintCoroutine );
961         hintCoroutine = null;
962     }
963
964     CancelInvoke( "ShowTeleportHint" );
965 }
966
967
968 //-----
969 private IEnumerator TeleportHintCoroutine()
970 {
971     float prevBreakTime = Time.time;
972     float prevHapticPulseTime = Time.time;
973
974     while ( true )
975     {
976         bool pulsed = false;
977
978         //Show the hint on each eligible hand
979         foreach ( Hand hand in player.hands )
980         {
981             bool showHint = IsEligibleForTeleport( hand );
982             bool isShowingHint = !string.IsNullOrEmpty(
983                 ControllerButtonHints.GetActiveHintText( hand,
984                     teleportAction ) );
985             if ( showHint )
986             {
987                 if ( !isShowingHint )
988                 {
989                     ControllerButtonHints.ShowTextHint( hand, teleportAction, "
990                         Teleport" );
991                     prevBreakTime = Time.time;
992                     prevHapticPulseTime = Time.time;
993                 }
994
995                 if ( Time.time > prevHapticPulseTime + 0.05f )
996                 {
997                     //Haptic pulse for a few seconds
998                     pulsed = true;
999
1000                     hand.TriggerHapticPulse( 500 );
1001                 }
1002             }
1003             else if ( !showHint && isShowingHint )
1004             {
1005                 ControllerButtonHints.HideTextHint( hand, teleportAction);
1006             }
1007         }
1008
1009         if ( Time.time > prevBreakTime + 3.0f )
1010         {
1011             //Take a break for a few seconds
1012             yield return new WaitForSeconds( 3.0f );
1013         }
1014     }

```

```

1011         prevBreakTime = Time.time;
1012     }
1013
1014     if ( pulsed )
1015     {
1016         prevHapticPulseTime = Time.time;
1017     }
1018
1019     yield return null;
1020 }
1021 }
1022
1023
1024 //-----
1025 public bool IsEligibleForTeleport( Hand hand )
1026 {
1027     if ( hand == null )
1028     {
1029         return false;
1030     }
1031
1032     if ( !hand.gameObject.activeInHierarchy )
1033     {
1034         return false;
1035     }
1036
1037     if ( hand.hoveringInteractable != null )
1038     {
1039         return false;
1040     }
1041
1042     if ( hand.noSteamVRFallbackCamera == null )
1043     {
1044         if ( hand.isActive == false )
1045         {
1046             return false;
1047         }
1048
1049         //Something is attached to the hand
1050         if ( hand.currentAttachedObject != null )
1051         {
1052             AllowTeleportWhileAttachedToHand
1053                 allowTeleportWhileAttachedToHand = hand.
1054                 currentAttachedObject.GetComponent<
1055                 AllowTeleportWhileAttachedToHand>();
1056
1057             if ( allowTeleportWhileAttachedToHand != null &&
1058                 allowTeleportWhileAttachedToHand.teleportAllowed == true )
1059             {
1060                 return true;
1061             }
1062             else
1063             {
1064                 return false;
1065             }
1066         }
1067     }

```

```

1061         }
1062     }
1063 }
1064
1065     return true;
1066 }
1067
1068
1069 //-----
1070 private bool ShouldOverrideHoverLock()
1071 {
1072     if ( !allowTeleportWhileAttached || allowTeleportWhileAttached.
        overrideHoverLock )
1073     {
1074         return true;
1075     }
1076
1077     return false;
1078 }
1079
1080
1081 //-----
1082 private bool WasTeleportButtonReleased( Hand hand )
1083 {
1084     if ( IsEligibleForTeleport( hand ) )
1085     {
1086         if ( hand.noSteamVRFallbackCamera != null )
1087         {
1088             return Input.GetKeyUp( KeyCode.T );
1089         }
1090         else
1091         {
1092             return teleportAction.GetStateUp(hand.handType);
1093
1094             //return hand.controller.GetPressUp(
1095                 SteamVR_Controller.ButtonMask.Touchpad );
1096         }
1097     }
1098
1099     return false;
1100 }
1101
1102 //-----
1103 private bool IsTeleportButtonDown( Hand hand )
1104 {
1105     if ( IsEligibleForTeleport( hand ) )
1106     {
1107         if ( hand.noSteamVRFallbackCamera != null )
1108         {
1109             return Input.GetKey( KeyCode.T );
1110         }
1111         else
1112         {
1113             return teleportAction.GetState(hand.handType);

```

```

1113     }
1114 }
1115
1116     return false;
1117 }
1118
1119
1120 //-----
1121 private bool WasTeleportButtonPressed( Hand hand )
1122 {
1123     if ( IsEligibleForTeleport( hand ) )
1124     {
1125         if ( hand.noSteamVRFallbackCamera != null )
1126         {
1127             return Input.GetKeyDown( KeyCode.T );
1128         }
1129         else
1130         {
1131             return teleportAction.GetStateDown(hand.handType);
1132
1133             //return hand.controller.GetPressDown(
1134                 SteamVR_Controller.ButtonMask.Touchpad );
1135         }
1136     }
1137
1138     return false;
1139 }
1140
1141 //-----
1142 private Transform GetPointerStartTransform( Hand hand )
1143 {
1144     if ( hand.noSteamVRFallbackCamera != null )
1145     {
1146         return hand.noSteamVRFallbackCamera.transform;
1147     }
1148     else
1149     {
1150         return hand.transform;
1151     }
1152 }
1153 }
1154 }

```

9.5.16. TeleportArc.cs

```

1 //===== Copyright (c) Valve Corporation, All rights reserved.
2 //=====
3 // Purpose: Displays the arc lines for teleporting and does the traces
4 //

```

```

5  //
6  =====
7  using UnityEngine;
8
9  namespace Valve.VR.InteractionSystem
10 {
11     //
12     -----
13
14     public class TeleportArc : MonoBehaviour
15     {
16         public int segmentCount = 60;
17         public float thickness = 0.01f;
18
19         [Tooltip("The amount of time in seconds to predict the motion of
20             the projectile.")]
21         public float arcDuration = 3.0f;
22
23         [Tooltip("The amount of time in seconds between each segment of
24             the projectile.")]
25         public float segmentBreak = 0.025f;
26
27         [Tooltip("The speed at which the line segments of the arc move.")
28             ]
29         public float arcSpeed = 0.2f;
30
31         public Material material;
32
33         [HideInInspector]
34         public int traceLayerMask = 0;
35
36         //Private data
37         private LineRenderer[] lineRenderers;
38         private float arcTimeOffset = 0.0f;
39         private float prevThickness = 0.0f;
40         private int prevSegmentCount = 0;
41         private bool showArc = true;
42         private Vector3 startPos;
43         private Vector3 projectileVelocity;
44         private bool useGravity = true;
45         private Transform arcObjectsTransform;
46         private bool arcInvalid = false;
47         private float scale = 1;
48
49         //-----
50         void Start()
51         {
52             arcTimeOffset = Time.time;
53         }
54     }
55 }

```

```

52 //-----
53 void Update()
54 {
55     //scale arc to match player scale
56     scale = Player.instance.transform.lossyScale.x;
57     if (thickness != prevThickness || segmentCount !=
        prevSegmentCount)
58     {
59         CreateLineRendererObjects();
60         prevThickness = thickness;
61         prevSegmentCount = segmentCount;
62     }
63 }
64
65
66
67 //-----
68 private void CreateLineRendererObjects()
69 {
70     //Destroy any existing line renderer objects
71     if (arcObjectsTransfrom != null)
72     {
73         Destroy(arcObjectsTransfrom.gameObject);
74     }
75
76     GameObject arcObjectsParent = new GameObject("ArcObjects");
77     arcObjectsTransfrom = arcObjectsParent.transform;
78     arcObjectsTransfrom.SetParent(this.transform);
79
80     //Create new line renderer objects
81     lineRenderers = new LineRenderer[segmentCount];
82     for (int i = 0; i < segmentCount; ++i)
83     {
84         GameObject newObject = new GameObject("LineRenderer_" + i
            );
85         newObject.transform.SetParent(arcObjectsTransfrom);
86
87         lineRenderers[i] = newObject.AddComponent<LineRenderer>()
            ;
88
89         lineRenderers[i].receiveShadows = false;
90         lineRenderers[i].reflectionProbeUsage = UnityEngine.
            Rendering.ReflectionProbeUsage.Off;
91         lineRenderers[i].lightProbeUsage = UnityEngine.Rendering.
            LightProbeUsage.Off;
92         lineRenderers[i].shadowCastingMode = UnityEngine.
            Rendering.ShadowCastingMode.Off;
93         lineRenderers[i].material = material;
94 #if (UNITY_5_4)
95         lineRenderers[i].SetWidth(thickness, thickness);
96 #else
97         lineRenderers[i].startWidth = thickness * scale;
98         lineRenderers[i].endWidth = thickness * scale;
99 #endif

```

```

100         lineRenderers[i].enabled = false;
101     }
102 }
103
104
105 //-----
106 public void SetArcData(Vector3 position, Vector3 velocity, bool
107     gravity, bool pointerAtBadAngle)
108 {
109     startPos = position;
110     projectileVelocity = velocity;
111     useGravity = gravity;
112
113     if (arcInvalid && !pointerAtBadAngle)
114     {
115         arcTimeOffset = Time.time;
116     }
117     arcInvalid = pointerAtBadAngle;
118 }
119
120 //-----
121 public void Show()
122 {
123     showArc = true;
124     if (lineRenderers == null)
125     {
126         CreateLineRendererObjects();
127     }
128 }
129
130
131 //-----
132 public void Hide()
133 {
134     //Hide the line segments if they were previously being shown
135     if (showArc)
136     {
137         HideLineSegments(0, segmentCount);
138     }
139     showArc = false;
140 }
141
142
143 //-----
144 // Draws each segment of the arc individually
145 //-----
146 public bool DrawArc(out RaycastHit hitInfo)
147 {
148     float timeStep = arcDuration / segmentCount;
149
150     float currentTimeOffset = (Time.time - arcTimeOffset) *
151         arcSpeed;

```

```

152 //Reset the arc time offset when it has gone beyond a segment
    length
153 if (currentTimeOffset > (timeStep + segmentBreak))
154 {
155     arcTimeOffset = Time.time;
156     currentTimeOffset = 0.0f;
157 }
158
159 float segmentStartTime = currentTimeOffset;
160
161 float arcHitTime = FindProjectileCollision(out hitInfo);
162
163 if (arcInvalid)
164 {
165     //Only draw first segment
166     lineRenderers[0].enabled = true;
167     lineRenderers[0].SetPosition(0, GetArcPositionAtTime(0.0f
        ));
168     lineRenderers[0].SetPosition(1, GetArcPositionAtTime(
        arcHitTime < timeStep ? arcHitTime : timeStep));
169
170     HideLineSegments(1, segmentCount);
171 }
172 else
173 {
174     //Draw the first segment outside the loop if needed
175     int loopStartSegment = 0;
176     if (segmentStartTime > segmentBreak)
177     {
178         float firstSegmentEndTime = currentTimeOffset -
            segmentBreak;
179         if (arcHitTime < firstSegmentEndTime)
180         {
181             firstSegmentEndTime = arcHitTime;
182         }
183         DrawArcSegment(0, 0.0f, firstSegmentEndTime);
184
185         loopStartSegment = 1;
186     }
187
188     bool stopArc = false;
189     int currentSegment = 0;
190     if (segmentStartTime < arcHitTime)
191     {
192         for (currentSegment = loopStartSegment;
            currentSegment < segmentCount; ++currentSegment)
193         {
194             //Clamp the segment end time to the arc duration
195             float segmentEndTime = segmentStartTime +
                timeStep;
196             if (segmentEndTime >= arcDuration)
197             {
198                 segmentEndTime = arcDuration;
199                 stopArc = true;

```



```

200         }
201
202         if (segmentEndTime >= arcHitTime)
203         {
204             segmentEndTime = arcHitTime;
205             stopArc = true;
206         }
207
208         DrawArcSegment(currentSegment, segmentStartTime,
209             segmentEndTime);
210
211         segmentStartTime += timeStep + segmentBreak;
212
213         //If the previous end time or the next start time
214         //is beyond the duration then stop the arc
215         if (stopArc || segmentStartTime >= arcDuration ||
216             segmentStartTime >= arcHitTime)
217         {
218             break;
219         }
220     }
221     else
222     {
223         currentSegment--;
224     }
225
226     //Hide the rest of the line segments
227     HideLineSegments(currentSegment + 1, segmentCount);
228 }
229
230 return arcHitTime != float.MaxValue;
231 }
232
233 //-----
234 private void DrawArcSegment(int index, float startTime, float
235     endTime)
236 {
237     lineRenderers[index].enabled = true;
238     lineRenderers[index].SetPosition(0, GetArcPositionAtTime(
239         startTime));
240     lineRenderers[index].SetPosition(1, GetArcPositionAtTime(
241         endTime));
242 }
243
244 //-----
245 public void SetColor(Color color)
246 {
247     for (int i = 0; i < segmentCount; ++i)
248     {
249         #if (UNITY_5_4)
250             lineRenderers[i].SetColors(color, color);
251         }
252     }

```

```

248 #else
249     lineRenderers[i].startColor = color;
250     lineRenderers[i].endColor = color;
251 #endif
252     }
253 }
254
255
256 //-----
257 private float FindProjectileCollision(out RaycastHit hitInfo)
258 {
259     float timeStep = arcDuration / segmentCount;
260     float segmentStartTime = 0.0f;
261
262     hitInfo = new RaycastHit();
263
264     Vector3 segmentStartPos = GetArcPositionAtTime(
265         segmentStartTime);
266     for (int i = 0; i < segmentCount; ++i)
267     {
268         float segmentEndTime = segmentStartTime + timeStep;
269         Vector3 segmentEndPos = GetArcPositionAtTime(
270             segmentEndTime);
271
272         if (Physics.Linecast(segmentStartPos, segmentEndPos, out
273             hitInfo, traceLayerMask))
274         {
275             if (hitInfo.collider.GetComponent<IgnoreTeleportTrace
276                 >() == null)
277             {
278                 Util.DrawCross(hitInfo.point, Color.red, 0.5f);
279                 float segmentDistance = Vector3.Distance(
280                     segmentStartPos, segmentEndPos);
281                 float hitTime = segmentStartTime + (timeStep * (
282                     hitInfo.distance / segmentDistance));
283                 return hitTime;
284             }
285         }
286
287         segmentStartTime = segmentEndTime;
288         segmentStartPos = segmentEndPos;
289     }
290
291     return float.MaxValue;
292 }
293
294 //-----
295 public Vector3 GetArcPositionAtTime(float time)
296 {
297     Vector3 gravity = useGravity ? Physics.gravity : Vector3.zero
298     ;

```

```

294         Vector3 arcPos = startPos + ((projectileVelocity * time) +
295             (0.5f * time * time) * gravity) * scale;
296         return arcPos;
297     }
298
299     //-----
300     private void HideLineSegments(int startSegment, int endSegment)
301     {
302         if (lineRenderers != null)
303         {
304             for (int i = startSegment; i < endSegment; ++i)
305             {
306                 lineRenderers[i].enabled = false;
307             }
308         }
309     }
310 }
311 }

```

9.5.17. TeleportArea

```

1  //===== Copyright (c) Valve Corporation, All rights reserved.
2  //=====
3  // Purpose: An area that the player can teleport to
4  //
5  //
6
7  using UnityEngine;
8  #if UNITY_EDITOR
9  using UnityEditor;
10 #endif
11
12 namespace Valve.VR.InteractionSystem
13 {
14     //
15     -----
16
17     public class TeleportArea : TeleportMarkerBase
18     {
19         //Public properties
20         public Bounds meshBounds { get; private set; }
21
22         //Private data
23         private MeshRenderer areaMesh;
24         private int tintColorId = 0;
25         private Color visibleTintColor = Color.clear;
26         private Color highlightedTintColor = Color.clear;
27         private Color lockedTintColor = Color.clear;

```

```

26     private bool highlighted = false;
27
28     //-----
29     public void Awake()
30     {
31         areaMesh = GetComponent<MeshRenderer>();
32
33         tintColorId = Shader.PropertyToID( "_TintColor" );
34
35         CalculateBounds();
36     }
37
38
39     //-----
40     public void Start()
41     {
42         visibleTintColor = Teleport.instance.areaVisibleMaterial.GetColor(
43             tintColorId );
44         highlightedTintColor = Teleport.instance.areaHighlightedMaterial.
45             GetColor( tintColorId );
46         lockedTintColor = Teleport.instance.areaLockedMaterial.GetColor(
47             tintColorId );
48     }
49
50
51     //-----
52     public override bool ShouldActivate( Vector3 playerPosition )
53     {
54         return true;
55     }
56
57
58     //-----
59     public override bool ShouldMovePlayer()
60     {
61         return true;
62     }
63
64
65     //-----
66     public override void Highlight( bool highlight )
67     {
68         if ( !locked )
69         {
70             highlighted = highlight;
71
72             if ( highlight )
73             {
74                 areaMesh.material = Teleport.instance.areaHighlightedMaterial;
75             }
76             else
77             {
78                 areaMesh.material = Teleport.instance.areaVisibleMaterial;
79             }
80         }
81     }

```

```
77     }
78 }
79
80
81 //-----
82 public override void SetAlpha( float tintAlpha, float alphaPercent )
83 {
84     Color tintedColor = GetTintColor();
85     tintedColor.a *= alphaPercent;
86     areaMesh.material.SetColor( tintColorId, tintedColor );
87 }
88
89
90 //-----
91 public override void UpdateVisuals()
92 {
93     if ( locked )
94     {
95         areaMesh.material = Teleport.instance.areaLockedMaterial;
96     }
97     else
98     {
99         areaMesh.material = Teleport.instance.areaVisibleMaterial;
100     }
101 }
102
103
104 //-----
105 public void UpdateVisualsInEditor()
106 {
107     if (Teleport.instance == null)
108         return;
109
110     areaMesh = GetComponent<MeshRenderer>();
111
112     if ( locked )
113     {
114         areaMesh.sharedMaterial = Teleport.instance.areaLockedMaterial;
115     }
116     else
117     {
118         areaMesh.sharedMaterial = Teleport.instance.areaVisibleMaterial;
119     }
120 }
121
122
123 //-----
124 private bool CalculateBounds()
125 {
126     MeshFilter meshFilter = GetComponent<MeshFilter>();
127     if ( meshFilter == null )
128     {
129         return false;
130     }
131 }
```

```

131
132     Mesh mesh = meshFilter.sharedMesh;
133     if ( mesh == null )
134     {
135         return false;
136     }
137
138     meshBounds = mesh.bounds;
139     return true;
140 }
141
142
143 //-----
144 private Color GetTintColor()
145 {
146     if ( locked )
147     {
148         return lockedTintColor;
149     }
150     else
151     {
152         if ( highlighted )
153         {
154             return highlightedTintColor;
155         }
156         else
157         {
158             return visibleTintColor;
159         }
160     }
161 }
162 }
163
164
165 #if UNITY_EDITOR
166 //
167
168 [CustomEditor( typeof( TeleportArea ) )]
169 public class TeleportAreaEditor : Editor
170 {
171     //-----
172     void OnEnable()
173     {
174         if ( Selection.activeTransform != null )
175         {
176             TeleportArea teleportArea = Selection.activeTransform.
177                 GetComponent<TeleportArea>();
178             if ( teleportArea != null )
179             {
180                 teleportArea.UpdateVisualsInEditor();
181             }
182         }
183     }
184 }

```

```

182
183
184 //-----
185 public override void OnInspectorGUI()
186 {
187     DrawDefaultInspector();
188
189     if ( Selection.activeTransform != null )
190     {
191         TeleportArea teleportArea = Selection.activeTransform.
            GetComponent<TeleportArea>();
192         if ( GUI.changed && teleportArea != null )
193         {
194             teleportArea.UpdateVisualsInEditor();
195         }
196     }
197 }
198 }
199 #endif
200 }

```

9.5.18. TeleportMarkerBase.cs

```

1 //===== Copyright (c) Valve Corporation, All rights reserved.
2 //=====
3 //
4 // Purpose: Base class for all the objects that the player can teleport
5 // to
6 //
7 //
8 //=====
9
10 using UnityEngine;
11
12 namespace Valve.VR.InteractionSystem
13 {
14     //
15     -----
16
17     public abstract class TeleportMarkerBase : MonoBehaviour
18     {
19         public bool locked = false;
20         public bool markerActive = true;
21
22         //-----
23         public virtual bool showReticle
24         {
25             get
26             {
27                 return true;
28             }
29         }
30     }
31 }

```

```

24     }
25
26
27     //-----
28     public void SetLocked( bool locked )
29     {
30         this.locked = locked;
31
32         UpdateVisuals();
33     }
34
35
36     //-----
37     public virtual void TeleportPlayer( Vector3 pointedAtPosition )
38     {
39     }
40
41
42     //-----
43     public abstract void UpdateVisuals();
44
45     //-----
46     public abstract void Highlight( bool highlight );
47
48     //-----
49     public abstract void SetAlpha( float tintAlpha, float alphaPercent );
50
51     //-----
52     public abstract bool ShouldActivate( Vector3 playerPosition );
53
54     //-----
55     public abstract bool ShouldMovePlayer();
56 }
57 }

```

9.5.19. TeleportPoint.cs

```

1  //===== Copyright (c) Valve Corporation, All rights reserved.
   //=====
2  //
3  // Purpose: Single location that the player can teleport to
4  //
5  //
   //=====
6
7  using UnityEngine;
8  using UnityEngine.UI;
9  #if UNITY_EDITOR
10 using UnityEditor;
11 #endif
12

```



```

13 namespace Valve.VR.InteractionSystem
14 {
15     //
16     -----
17
18     public class TeleportPoint : TeleportMarkerBase
19     {
20         public enum TeleportPointType
21         {
22             MoveToLocation,
23             SwitchToNewScene
24         };
25
26         //Public variables
27         public TeleportPointType teleportType = TeleportPointType.
28             MoveToLocation;
29         public string title;
30         public string switchToScene;
31         public Color titleVisibleColor;
32         public Color titleHighlightedColor;
33         public Color titleLockedColor;
34         public bool playerSpawnPoint = false;
35
36         //Private data
37         private bool gotRelevantComponents = false;
38         private MeshRenderer markerMesh;
39         private MeshRenderer switchSceneIcon;
40         private MeshRenderer moveLocationIcon;
41         private MeshRenderer lockedIcon;
42         private MeshRenderer pointIcon;
43         private Transform lookAtJointTransform;
44         private new Animation animation;
45         private Text titleText;
46         private Player player;
47         private Vector3 lookAtPosition = Vector3.zero;
48         private int tintColorID = 0;
49         private Color tintColor = Color.clear;
50         private Color titleColor = Color.clear;
51         private float fullTitleAlpha = 0.0f;
52
53         //Constants
54         private const string switchSceneAnimation = "switch_scenes_idle";
55         private const string moveLocationAnimation = "move_location_idle";
56         private const string lockedAnimation = "locked_idle";
57
58         //-----
59         public override bool showReticle
60         {
61             get
62             {
63                 return false;
64             }
65         }
66     }
67 }

```

```

64
65
66 //-----
67 void Awake()
68 {
69     GetRelevantComponents();
70
71     animation = GetComponent<Animation>();
72
73     tintColorID = Shader.PropertyToID( "_TintColor" );
74
75     moveLocationIcon.gameObject.SetActive( false );
76     switchSceneIcon.gameObject.SetActive( false );
77     lockedIcon.gameObject.SetActive( false );
78
79     UpdateVisuals();
80 }
81
82
83 //-----
84 void Start()
85 {
86     player = Player.instance;
87 }
88
89
90 //-----
91 void Update()
92 {
93     if ( Application.isPlaying )
94     {
95         lookAtPosition.x = player.hmdTransform.position.x;
96         lookAtPosition.y = lookAtJointTransform.position.y;
97         lookAtPosition.z = player.hmdTransform.position.z;
98
99         lookAtJointTransform.LookAt( lookAtPosition );
100     }
101 }
102
103
104 //-----
105 public override bool ShouldActivate( Vector3 playerPosition )
106 {
107     return ( Vector3.Distance( transform.position, playerPosition ) >
108             1.0f );
109 }
110
111 //-----
112 public override bool ShouldMovePlayer()
113 {
114     return true;
115 }
116

```

```
117
118 //-----
119 public override void Highlight( bool highlight )
120 {
121     if ( !locked )
122     {
123         if ( highlight )
124         {
125             SetMeshMaterials( Teleport.instance.pointHighlightedMaterial ,
126                             titleHighlightedColor );
127         }
128         else
129         {
130             SetMeshMaterials( Teleport.instance.pointVisibleMaterial ,
131                             titleVisibleColor );
132         }
133     }
134
135     if ( highlight )
136     {
137         pointIcon.gameObject.SetActive( true );
138         animation.Play();
139     }
140     else
141     {
142         pointIcon.gameObject.SetActive( false );
143         animation.Stop();
144     }
145 }
146
147 //-----
148 public override void UpdateVisuals()
149 {
150     if ( !gotRelevantComponents )
151     {
152         return;
153     }
154
155     if ( locked )
156     {
157         SetMeshMaterials( Teleport.instance.pointLockedMaterial ,
158                         titleLockedColor );
159
160         pointIcon = lockedIcon;
161
162         animation.clip = animation.GetClip( lockedAnimation );
163     }
164     else
165     {
166         SetMeshMaterials( Teleport.instance.pointVisibleMaterial ,
167                         titleVisibleColor );
168
169         switch ( teleportType )
```

```

167     {
168         case TeleportPointType.MoveToLocation:
169             {
170                 pointIcon = moveLocationIcon;
171
172                 animation.clip = animation.GetClip( moveLocationAnimation );
173                 ;
174             }
175             break;
176         case TeleportPointType.SwitchToNewScene:
177             {
178                 pointIcon = switchSceneIcon;
179
180                 animation.clip = animation.GetClip( switchSceneAnimation );
181             }
182             break;
183     }
184
185     titleText.text = title;
186 }
187
188
189 //-----
190 public override void SetAlpha( float tintAlpha, float alphaPercent )
191 {
192     tintColor = markerMesh.material.GetColor( tintColorID );
193     tintColor.a = tintAlpha;
194
195     markerMesh.material.SetColor( tintColorID, tintColor );
196     switchSceneIcon.material.SetColor( tintColorID, tintColor );
197     moveLocationIcon.material.SetColor( tintColorID, tintColor );
198     lockedIcon.material.SetColor( tintColorID, tintColor );
199
200     titleColor.a = fullTitleAlpha * alphaPercent;
201     titleText.color = titleColor;
202 }
203
204
205 //-----
206 public void SetMeshMaterials( Material material, Color textColor )
207 {
208     markerMesh.material = material;
209     switchSceneIcon.material = material;
210     moveLocationIcon.material = material;
211     lockedIcon.material = material;
212
213     titleColor = textColor;
214     fullTitleAlpha = textColor.a;
215     titleText.color = titleColor;
216 }
217
218
219 //-----

```

```

220 public void TeleportToScene()
221 {
222     if ( !string.IsNullOrEmpty( switchToScene ) )
223     {
224         Debug.Log("<b>[SteamVR Interaction]</b> TeleportPoint: Hook up
                your level loading logic to switch to new scene: " +
                switchToScene );
225     }
226     else
227     {
228         Debug.LogError("<b>[SteamVR Interaction]</b> TeleportPoint:
                Invalid scene name to switch to: " + switchToScene );
229     }
230 }
231
232
233 //-----
234 public void GetRelevantComponents()
235 {
236     markerMesh = transform.Find( "teleport_marker_mesh" ).GetComponent<
        MeshRenderer>();
237     switchSceneIcon = transform.Find( "teleport_marker_lookat_joint/
        teleport_marker_icons/switch_scenes_icon" ).GetComponent<
        MeshRenderer>();
238     moveLocationIcon = transform.Find( "teleport_marker_lookat_joint/
        teleport_marker_icons/move_location_icon" ).GetComponent<
        MeshRenderer>();
239     lockedIcon = transform.Find( "teleport_marker_lookat_joint/
        teleport_marker_icons/locked_icon" ).GetComponent<MeshRenderer
        >();
240     lookAtJointTransform = transform.Find( "
        teleport_marker_lookat_joint" );
241
242     titleText = transform.Find( "teleport_marker_lookat_joint/
        teleport_marker_canvas/teleport_marker_canvas_text" ).
        GetComponent<Text>();
243
244     gotRelevantComponents = true;
245 }
246
247
248 //-----
249 public void ReleaseRelevantComponents()
250 {
251     markerMesh = null;
252     switchSceneIcon = null;
253     moveLocationIcon = null;
254     lockedIcon = null;
255     lookAtJointTransform = null;
256     titleText = null;
257 }
258
259
260 //-----

```

```

261 public void UpdateVisualsInEditor()
262 {
263     if ( Application.isPlaying )
264     {
265         return;
266     }
267
268     GetRelevantComponents();
269
270     if ( locked )
271     {
272         lockedIcon.gameObject.SetActive( true );
273         moveLocationIcon.gameObject.SetActive( false );
274         switchSceneIcon.gameObject.SetActive( false );
275
276         markerMesh.sharedMaterial = Teleport.instance.pointLockedMaterial
277         ;
278         lockedIcon.sharedMaterial = Teleport.instance.pointLockedMaterial
279         ;
280
281         titleText.color = titleLockedColor;
282     }
283     else
284     {
285         lockedIcon.gameObject.SetActive( false );
286
287         markerMesh.sharedMaterial = Teleport.instance.
288             pointVisibleMaterial;
289         switchSceneIcon.sharedMaterial = Teleport.instance.
290             pointVisibleMaterial;
291         moveLocationIcon.sharedMaterial = Teleport.instance.
292             pointVisibleMaterial;
293
294         titleText.color = titleVisibleColor;
295
296         switch ( teleportType )
297         {
298             case TeleportPointType.MoveToLocation:
299             {
300                 moveLocationIcon.gameObject.SetActive( true );
301                 switchSceneIcon.gameObject.SetActive( false );
302             }
303             break;
304             case TeleportPointType.SwitchToNewScene:
305             {
306                 moveLocationIcon.gameObject.SetActive( false );
307                 switchSceneIcon.gameObject.SetActive( true );
308             }
309             break;
310         }
311     }
312
313     titleText.text = title;

```

```

310         ReleaseRelevantComponents();
311     }
312 }
313
314
315 #if UNITY_EDITOR
316     //
317     -----
318     [CustomEditor( typeof( TeleportPoint ) )]
319     public class TeleportPointEditor : Editor
320     {
321         //-----
322         void OnEnable()
323         {
324             if ( Selection.activeTransform )
325             {
326                 TeleportPoint teleportPoint = Selection.activeTransform.
327                     GetComponent<TeleportPoint>();
328                 if ( teleportPoint != null )
329                     teleportPoint.UpdateVisualsInEditor();
330             }
331         }
332         //-----
333         public override void OnInspectorGUI()
334         {
335             DrawDefaultInspector();
336
337             if ( Selection.activeTransform )
338             {
339                 TeleportPoint teleportPoint = Selection.activeTransform.
340                     GetComponent<TeleportPoint>();
341                 if ( GUI.changed )
342                 {
343                     teleportPoint.UpdateVisualsInEditor();
344                 }
345             }
346         }
347     }
348 #endif

```

9.5.20. UIElement.cs

```

1  //===== Copyright (c) Valve Corporation, All rights reserved.
2  //=====
3  // Purpose: UIElement that responds to VR hands and generates UnityEvents
4  //

```

```

5  //
   =====
6
7  using UnityEngine;
8  using UnityEngine.Events;
9  using UnityEngine.UI;
10 using System;
11
12 namespace Valve.VR.InteractionSystem
13 {
14     //
   -----
15
16     [RequireComponent( typeof( Interactable ) )]
17     public class UIElement : MonoBehaviour
18     {
19         public CustomEvents.UnityEventHand onHandClick;
20
21         protected Hand currentHand;
22
23         //-----
24         protected virtual void Awake()
25         {
26             Button button = GetComponent<Button>();
27             if ( button )
28             {
29                 button.onClick.AddListener( OnButtonClick );
30             }
31
32
33         //-----
34         protected virtual void OnHandHoverBegin( Hand hand )
35         {
36             currentHand = hand;
37             InputModule.instance.HoverBegin( gameObject );
38             ControllerButtonHints.ShowButtonHint( hand, hand.uiInteractAction);
39         }
40
41
42         //-----
43         protected virtual void OnHandHoverEnd( Hand hand )
44         {
45             InputModule.instance.HoverEnd( gameObject );
46             ControllerButtonHints.HideButtonHint( hand, hand.uiInteractAction);
47             currentHand = null;
48         }
49
50
51         //-----
52         protected virtual void HandHoverUpdate( Hand hand )
53         {

```



```

54         if ( hand.uiInteractAction != null && hand.uiInteractAction.
55             GetStateDown(hand.handType) )
56         {
57             InputModule.instance.Submit( gameObject );
58             ControllerButtonHints.HideButtonHint( hand, hand.uiInteractAction
59             );
60         }
61     }
62     //-----
63     protected virtual void OnButtonClick()
64     {
65         onHandClick.Invoke( currentHand );
66     }
67 }
68
69 #if UNITY_EDITOR
70 //
71 //-----
72 [UnityEditor.CustomEditor( typeof( UIElement ) )]
73 public class UIElementEditor : UnityEditor.Editor
74 {
75     //-----
76     // Custom Inspector GUI allows us to click from within the UI
77     //-----
78     public override void OnInspectorGUI()
79     {
80         DrawDefaultInspector();
81
82         UIElement uiElement = (UIElement)target;
83         if ( GUILayout.Button( "Click" ) )
84         {
85             InputModule.instance.Submit( uiElement.gameObject );
86         }
87     }
88 }
89 #endif

```


Apéndice A

Assets importados

Nombre	Tipo	Descripción	Autor	Fuente
Electric+lab	Model	Mesa de laboratorio	'Men in Black'	3D Warehouse
metal+fence+audrius	Model	Jaula de seguridad	'Storboda1980'	3D Warehouse
door+classroom	Model	Puerta	'Wong Yee Woon G.'	3D Warehouse
AS-120mm	Model	Ventana	'pizzi_lace'	3D Warehouse
Siemens-Sirius-ACT	Model	Seta emergencia	'Chris W.'	3D Warehouse
computer+desk	Model	Mesa con ordenador	'Paztecarms'	3D Warehouse
sikringskap	Model	Cuadro eléctrico	'Kai'	3D Warehouse
B0201	Model	Modelo de pruebas	'Press Start'	Unity Asset Store
Starfield Skybox	Skybox	Cielo	'Nightsoundgames'	Unity Asset Store
PBR Materials light	Material	Texturas	'Pixel Processor'	Unity Asset Store
Free architectural textures	Material	Texturas	'Crazytextures'	Unity Asset Store
IRB120	Model	Robot	'ABB'	abb.com

Cuadro A.1: Assets importados de fuentes externas.

Apéndice B

Imágenes adicionales de la aplicación

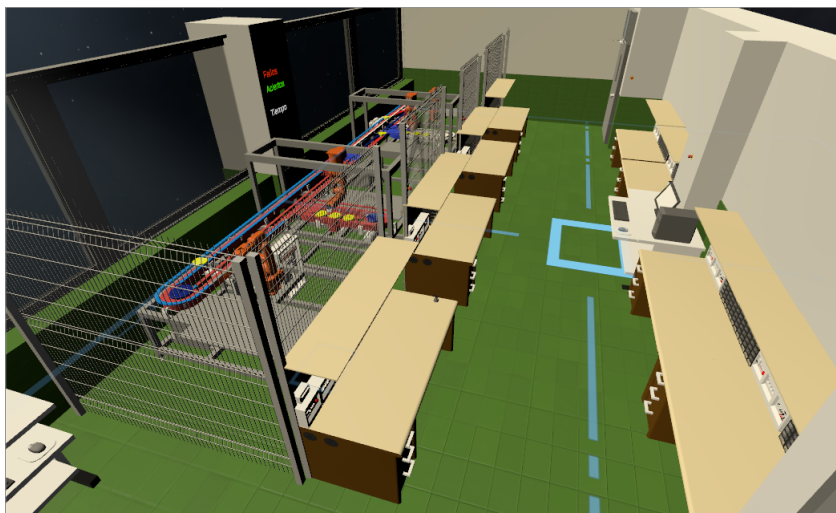


Figura B.1: Vista del laboratorio general.

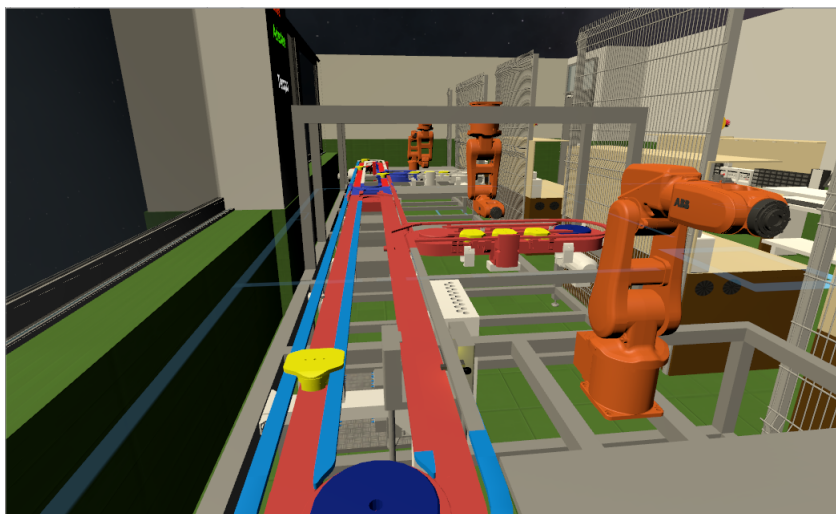


Figura B.2: Vista de la planta.

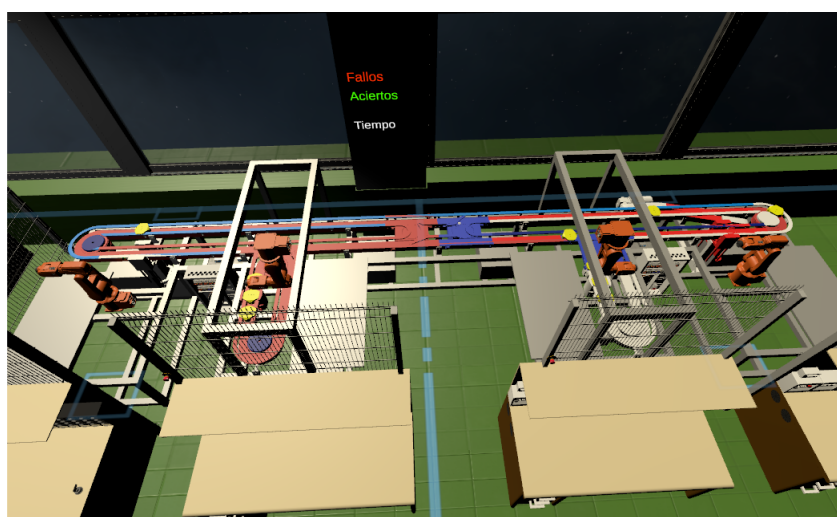


Figura B.3: Vista de la planta desde arriba.



Figura B.4: Vista de la zona de trabajo.

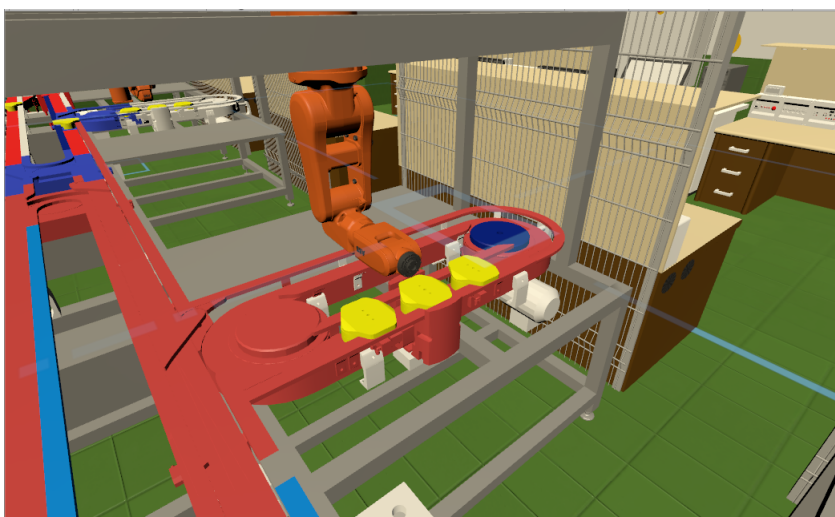


Figura B.5: Vista del robot ABB en la planta.

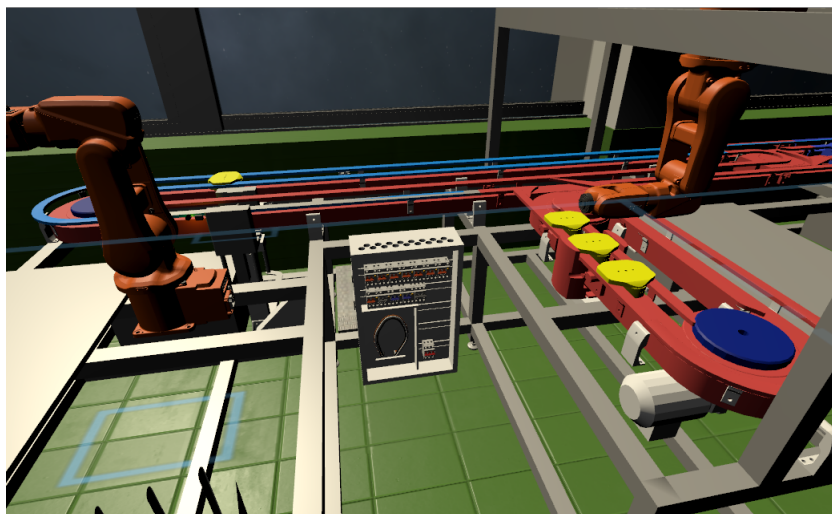


Figura B.6: Detalle de la zona izquierda de la planta.

Bibliografía

- [1] *3D Warehouse*. 2019. Último acceso 01/07/19. URL: <https://3dwarehouse.sketchup.com/>.
- [2] *ABB*. 2019. Último acceso 01/07/19. URL: <https://new.abb.com/es>.
- [3] Amparo Caballero. *Industria 4.0 a través de Realidad Virtual y Realidad Aumentada*. 2017. Último acceso 01/07/19. URL: <http://www.innoarea.com/industria-4-0-a-traves-de-realidad-virtual-y-realidad-aumentada/>.
- [4] *Historia de la realidad virtual*. Último acceso 30/06/19. URL: https://es.wikipedia.org/wiki/Historia_de_la_realidad_virtual.
- [5] Joseph Hocking. *Unity in Action*. 20 Baldwin Road, Shelter Island, NY 11964: Manning, 2015. ISBN: 9781617292323.
- [6] Iraltacamino. *Historia de la realidad virtual*. 2016. Último acceso 01/07/19. URL: <http://caminosantiago360.com/historia-la-realidad-virtual/>.
- [7] Carlos Álvarez Vereterra. Trabajo Fin de Máster. *Uso de la realidad virtual para el entrenamiento del personal de operación y mantenimiento: aplicación a la minifábrica ICAI*. Madrid, 2018.
- [8] Carlos Álvarez Vereterra. Trabajo Fin de Máster. “Uso de la realidad virtual para el entrenamiento del personal de operación y mantenimiento: aplicación a la minifábrica ICAI”. En: Madrid: ICAI, 2018. Cap. 3, págs. 18-25.
- [9] Carlos Álvarez Vereterra. Trabajo Fin de Máster. “Uso de la realidad virtual para el entrenamiento del personal de operación y mantenimiento: aplicación a la minifábrica ICAI”. En: Madrid: ICAI, 2018. Cap. 3, pág. 18.
- [10] Jack Nove. *ZeroLight, BMW, HTC VIVE and NVIDIA Team Up for Virtual Reality Showcase at CES 2019*. 2019. Último acceso 01/07/19. URL: <https://zerolight.com/news/press-releases/zerolight-bmw-htc-vive-and-nvidia-team-up-for-virtual-reality-showcase-at-ces-2019>.
- [11] *Oculus*. 2019. Último acceso 01/07/19. URL: https://www.oculus.com/rift/?locale=es_ES#oui-csl-rift-games=games-tale.
- [12] *Oculus Rift*. Último acceso 02/07/19. URL: https://es.wikipedia.org/wiki/Oculus_Rift.

- [13] José L. Ortega. *Un repaso a la historia de la realidad virtual*. 2016. Último acceso 01/07/19. URL: <https://es.ign.com/realidad-virtual/109691/feature/un-repaso-a-la-historia-de-la-realidad-virtual>.
- [14] *Pixo Group*. 2019. Último acceso 08/07/19. URL: <https://pixogroup.com/>.
- [15] Julián Pérez Porto y María Merino. *Definición de realidad virtual*. 2013. Último acceso 01/07/19. URL: definicion.de/realidad-virtual.
- [16] *Steam VR*. 2019. Último acceso 01/07/19. URL: <https://www.steamvr.com/en/>.
- [17] *Unity*. 2019. Último acceso 01/07/19. URL: <https://unity3d.com/es/unity>.
- [18] *Unity Documentation*. 2019. Último acceso 01/07/19. URL: <https://docs.unity3d.com/ScriptReference/>.
- [19] Cámara Valencia. *Caminar con éxito hacia la Industria 4.0: Capítulo 18 - Realidad Aumentada y Virtual*. 2018. Último acceso 01/07/19. URL: <https://ticnegocios.camaravalencia.com>.
- [20] Mundo Virtual. *¿Qué es la realidad virtual?* Último acceso 01/07/19. URL: mundo-virtual.com/que-es-la-realidad-virtual.
- [21] *Welcome to the Evolution of Safety Training: Virtual Reality*. 2019. Último acceso 01/07/19. URL: https://www.3m.com/3M/en_US/worker-health-safety-us/3m-ppe-training/virtual-reality/.