UNIVERSIDAD DE CASTILLA-LA MANCHA

ESCUELA SUPERIOR DE INFORMÁTICA

GRADO EN INGENIERÍA INFORMÁTICA

Tecnología Específica de Computación

TRABAJO FIN DE GRADO

RoboTIC: un juego serio basado en realidad aumentada para el aprendizaje de la programación

Francisco Manuel García Sánchez-Belmonte

julio, 2020

UNIVERSIDAD DE CASTILLA-LA MANCHA

ESCUELA SUPERIOR DE INFORMÁTICA

DEPARTAMENTO DE TECNOLOGÍAS Y SISTEMAS DE INFORMACIÓN

Tecnología Específica de Computación

TRABAJO FIN DE GRADO

RoboTIC: un juego serio basado en realidad aumentada para el aprendizaje de la programación

Autor: Francisco Manuel García Sánchez-Belmonte

Tutor: Dr. Carlos González Morcillo

Co-Tutor: Santiago Sánchez Sobrino

julio, 2020

RoboTIC

[1] https://github.com/JesusSalido/TFG_ESI_UCLM
[2] https://www.overleaf.com/latex/templates/plantilla-de-tfg-escuela-superior-de-informatica-uclm/phjgscmfqtsw

Tribunal:

Presidente: _____

Vocal: _____

Secretario: _____

Fecha de defensa: _____

Calificación: _____

Presidente	Vocal	Secretario
Fdo.:	Fdo.:	Fdo.:

"Por fin."

Resumen

Nos encontramos al comienzo de la Cuarta Revolución Industrial por lo que cada día es más importante el conocimiento de la programación y en un futuro será requisito imprescindible para la gran mayoría de ámbitos de la actividad humana.

Aprender a programar no es una tarea sencilla, y cuenta con diversos factores que dificultan su enseñanza en edades tempranas. La necesidad de entender conceptos abstractos, la escasa motivación que puede tener el estudiante a la hora de resolver ejercicios de programación y el enfoque tradicional de ciertos métodos pueden tener un impacto negativo en los procesos de enseñanza-aprendizaje.

Para paliar esta situación y apoyado en el inmenso potencial que ofrece la realidad aumentada surge RoboTIC: un juego serio en el que se ha prestado muy especial atención a atraer y motivar a los estudiantes mediante gráficos en 3D, un paradigma de interacción natural persona-computador y el desarrollo de todo un sistema de metáforas visuales para guiar a los usuarios a través de una serie de niveles con el fin de enseñar de forma gradual las habilidades necesarias para llevar a cabo con éxito la programación de ordenadores.

Abstract

We are at the beginning of the Fourth Industrial Revolution so knowledge of programming is increasingly important and in the future will be a prerequisite for the vast majority of areas of human activity.

Learning to code is not an easy task and has several factors that make it difficult to teach at an early age. The need to understand abstract concepts, the low motivation that the student may have when solving programming exercises and the traditional approach of certain methods may have a negative impact on the teaching-learning processes.

To alleviate this situation and supported by the immense potential offered by augmented reality arises RoboTIC: a serious game in which special attention has been paid to attract and motivate students with the use of 3D graphics, a paradigm of natural human-computer interaction and the development of a whole system of visual metaphors to guide users through a series of levels in order to gradually teach the skills needed to successfully carry out computer programming.

AGRADECIMIENTOS

A mis padres por su apoyo constante, a Ainhoa por aguantarme durante toda la carrera y a Ana por evitar que este ultimo año me tire por la ventana.

Por supuesto no puedo dejar fuera a todos los amigos que he hecho durante el grado ya que sin sus apuntes y las noches de biblioteca hasta arriba de cafeína seguramente seguiría en primero peleándome con física... y desde luego igual que me acuerdo de lo académico no me voy a olvidar de comentar por aquí lo bien que nos lo hemos pasado (y nos lo vamos a seguir pasando) cada jueves en el *Traste*. No voy a poner vuestros nombres porque sois un montón y si se me olvida alguien seguro que me lo está recordando hasta el día en que me muera.

Por ultimo pero no menos importante, a Carlos y a Santi por dirigir este proyecto y al resto de profesores de la ESI porque sin las *cosas de ordenadores* que me han enseñado durante estos años hacer un TFG de Ingeniería Informática habría sido mucho más complicado.

Francisco Manuel García Sánchez-Belmonte
Ciudad Real, 2020

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

ÍNDICE DE LISTADOS

CAPÍTULO 1

INTRODUCCIÓN

A lo largo de las últimas décadas la programación se ha vuelto una rama del conocimiento omnipresente en todos los niveles de la sociedad. Si bien el desarrollo y la implantación de los computadores en la vida diaria ha supuesto un gran impulso en el progreso de la actividad humana también trae consigo nuevos retos que superar: ¿cómo afrontar la problemática de inculcar el aprendizaje de algo tan complejo (y necesario) como es el conocimiento de la programación de ordenadores desde una temprana edad con el fin de formar individuos que puedan desenvolverse correctamente en los entornos profesionales y sociales del futuro?

Durante toda la historia una serie de descubrimientos tecnológicos han cambiado radicalmente la forma de entender el mundo y de vivir de la población del momento: el control del fuego revolucionó desde la dieta a la manera de defenderse, introdujo nuevas posibilidades y técnicas para crear herramientas más avanzadas, forjar metales, poder continuar la actividad tras caer la noche y en condiciones de frío; la máquina de vapor impulsó la industria a niveles insospechados, trajo el ferrocarril y enormes barcos; el telégrafo, la radio y el teléfono permitieron que el ser humano se comunicara con una inmediatez que jamás se había visto hasta la fecha.

Durante el siglo XX se produjeron una serie de avances tecnológicos cada vez más rápidos en el campo de la computación con el desarrollo de múltiples máquinas y sistemas de todo tipo. Al principio maquinas enormes, caras y solitarias; después potentes, accessibles, conectadas y de múltiples tamaños y presentaciones para cualquier fin imaginable. Otra Revolución Industrial había comenzado.

En el siglo XXI los computadores han cambiado la forma de ver y entender el mundo a niveles nunca antes vistos. Vivimos en el contexto de la Cuarta Revolución Industrial[15] y nos encontramos en una sociedad hiperconectada en que la información viaja en centésimas de segundo de un punto a otro del planeta y ya no se puede entender la industria sin todo un conjunto de sistemas informáticos alrededor de ella.

Tal revolución no viene sin problemas asociados como por ejemplo la brecha digital (la desigualdad en el acceso a las nuevas tecnologías), la necesidad de ser flexible y abierto a los cambios que se suceden rápidamente en el mundo de las TIC y la casi imposición de tener conocimientos del uso y la programación de los dispositivos que nos rodean.

La programación de ordenadores no es solo una herramienta de trabajo más pues entre otros muchos beneficios de esta podemos citar que es un buen método para enseñar pensamiento crítico y matemático, aumenta la creatividad mediante la búsqueda e implementación de soluciones a problemas y mejora la confianza en uno mismo al resolverlos y por supuesto propicia la adquisición de habilidades de trabajo en equipo tan importantes en el mundo actual.[9]

Con tal cantidad de ventajas sobre la mesa es difícil encontrar las razones por las cuales la programación no se incluye de forma generalizada en los currículos de todas las etapas de la educación y se establece como una competencia básica que los estudiantes deben adquirir y dominar como las matemáticas o el aprendizaje de la lengua.

Algunas de estas razones podrían ser la anteriormente mencionada brecha digital, las condiciones socioeconómicas y nivel de desarrollo diferente de cada ciudad y país y por supuesto que la programación se considera una disciplina compleja de enseñar y complicada de aprender.

El aprendizaje de la programación requiere mucha práctica la cual se consigue mediante repetición de ejercicios y práctica deliberada por lo que se puede convertir en algo muy difícil para estudiantes principiantes. Para aportar una solución a este problema surge RoboTIC.

RoboTIC es una evolución del proyecto *"RoboTIC: a serious game based on augmented reality for learning programming"*[14] financiado por Telefónica y desarrollado por Furious Koalas S.L. en el año 2018 siendo el actual trabajo una refactorización completa en la que no se ha reutilizado código y se ha intentado proporcionar funcionalidad nueva como un editor de niveles y diversas mejoras en la usabilidad. RoboTIC combina la ludificación y la realidad aumentada para crear un entorno ameno y divertido en el que enseñar las bases de la programación a escolares de forma simple y efectiva.

¿Por qué dirigido especialmente a personas de corta edad? Uno de los grandes problemas de la informática es la cantidad de mitos que pesan sobre ella (como que es para gente rara o solo para hombres) por lo que es interesante acercar la programación a las personas antes de que sean desalentados del aprendizaje de la misma por estos prejuicios. También los estudiantes jóvenes tienen mayor habilidad para aprender lenguajes naturales y quizá este proceso se pueda extrapolar a los lenguajes de programación, además de darles la posibilidad desde temprana edad de dejar de ser meros consumidores pasivos de productos digitales para pasar a ser creadores activos.[6]

La ludificación o gamificación es un proceso por el cual se aplican elementos que tradicionalmente pertenecen a juegos en actividades que no lo son. Algunos de estos elementos pueden ser sistemas que otorguen puntos al usuario por realizar ciertas tareas, crear una estructura de niveles con progresión entre ellos, rankings para explotar la competitividad entre usuarios, recompensas de diversos tipos por tener éxito, etc.

Algunos ejemplos de ludificación puede ser un videojuego que recompense al jugador con puntos cada vez que hace una tarea beneficiosa para él como dormir una cantidad de horas apropiada, una aplicación con un ranking de qué usuario corre más tiempo, aumentar el número de participantes de unas elecciones mediante una lotería entre todos los votantes, otorgar puntos de experiencia a los clientes de una tienda para cambiarlos por niveles que se traducen en descuentos o servicios especiales o gratuitos…

Octalysis[1] es un framework de gamificación surgido de la necesidad de analizar qué hace a un juego divertido y cómo construir estrategias para mantener al usuario motivado. Se basa en las ocho características que hacen un juego divertido y motivador para el jugador colocadas en los lados de un octágono.

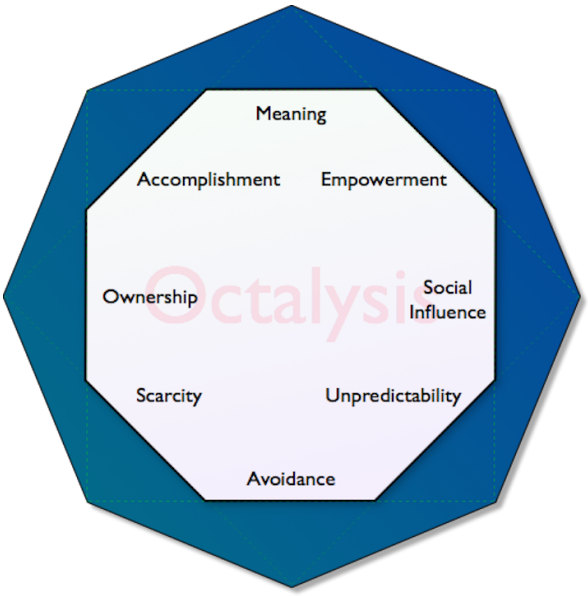


Figura 1.1: Framework Octalysis (por Yu-kai Chou, CC BY-SA 4.0)

Estas características principales son:[4]

- *Meaning* (significado): se produce cuando un jugador cree que está haciendo algo grande o que fue elegido en algún sentido.
- *Accomplishment* (realización): la sensación que se genera al recibir una recompensa o desarrollar alguna habilidad o progreso tras superar algún tipo de reto.
- *Empowerment* (empoderamiento): esta característica se refiere concretamente a dar al usuario una forma de expresar su creatividad.
- *Ownership* (pertenencia): los usuarios se sienten motivados cuando acumulan algún tipo de bien, por ejemplo monedas de juego u objetos y mejor se sentirán cuanto más exclusivos sean dichos objetos.
- *Social influence* (influencia social): cuando una persona ve que otra tiene una habilidad o en general cualquier cosa que no tiene se sentirá influenciado a intentar conseguir los mismo.
- *Scarcity* (escasez): querer algo por el mero hecho de no tenerlo. Por ejemplo lanzar una beta cerrada de un producto software hará que los potenciales compradores acudan cuando el servicio se abra simplemente porque antes no podían ser parte de ello.
- *Unpredictability* (imprevisibilidad): el muy humano comportamiento de querer saber qué pasará a continuación.
- *Avoidance* (evasión): se basa en evitar que algo malo pase, por ejemplo perder puntos o trabajo ya hecho.

[1] https://yukaichou.com/gamification-examples/octalysis-complete-gamification-framework/

Muchos son los beneficios de la ludificación pero de especial relevancia para este trabajo es que mezclada con la programación puede crear una experiencia de aprendizaje amena y divertida que mejore la absorción de conocimientos por parte de los estudiantes y los predisponga a desarrollar un interés por la informática y la tecnología. A lo largo del proyecto se explotarán los conceptos de accomplishment y de empowerment para motivar al jugador a completar los niveles de RoboTIC y desarrollar los suyos propios a la vez que aprende programación.

A día de hoy ya existen varias herramientas de gamificación para la enseñanza de la programación pero tienen una carencia muy importante, sobretodo en las que están enfocadas a usuarios de corta edad: no permiten una interacción natural del jugador con el sistema pues lo más natural es interactuar directamente con la imagen que recibimos del mundo real, eliminando la percepción que tiene el usuario de estar utilizando un ordenador.

Las interfaces naturales de usuario (NUI, *Natural User Interfaces*) son un tipo de interacción persona-computador en la que la interfaz no puede ser percibida, por tanto eliminando dificultades a la hora de introducir a los usuarios al sistema y se contrapone a lo común de usar dispositivos artificiales como ratones o teclados para realizar la entrada de comandos.

La realidad aumentada abre un mundo nuevo de posibilidades de cara a la educación al integrar fluidamente el mundo real con todas las posibilidades del contenido multimedia digital pudiendo conseguir una experiencia más enriquecedora y entretenida para el estudiante logrando aumentar el volumen de conocimientos adquiridos sin afectar a la delicada capacidad de atención de las personas en edad escolar.

Entre algunos ejemplos de aplicación se encontraría el hacer que la historia cobre vida y trascienda el papel mediante simulaciones, la observación de la geometría de los diferentes cuerpos y de múltiples procesos físicos, geológicos y químicos sin siquiera salir del aula y de muy especial interés para este trabajo es la visualización de algoritmos básicos de forma amena y divertida para el usuario.

La realidad aumentada combinada con un sistema de entrada usando gestos permite a RoboTIC romper la barrera entre el usuario y el sistema adoptando efectivamente una interfaz de usuario natural. Existen múltiples formas y dispositivos para lograr una experiencia de realidad aumentada cada uno con sus ventajas y desventajas las cuales se tratarán en profundidad en sucesivos capítulos pero para este proyecto se ha elegido desarrollarlo para el headset Hololens de Microsoft.

CAPÍTULO 2

ESTADO DEL ARTE

2.1. ANTECEDENTES

Es cierto que a lo largo de los años han surgido multitud de proyectos con la finalidad de enseñar programación de diferentes maneras, algunas más tradicionales que otras y se ha considerado constructivo llevar a cabo una pequeña investigación sobre algunos de los proyectos más relevantes en este campo para poder apreciar por qué RoboTIC se ha diseñado con ciertas características en mente.

2.1.1. AgentCubes

AgentCubes[1] es un entorno de programación online basado en bloques que hace énfasis en la creación de entornos en 3D con herramientas especializadas para ello. Una característica reseñable de este proyecto es su enfoque *cliffhanger* en el que los estudiantes construyen los modelos de su juego y empiezan a programar lo más pronto posible para tener una versión funcional en apenas una hora. Esta inmediatez de resultados consigue motivar a un 65 % de los usuarios a continuar el desarrollo de su videojuego. Fuera de la educación ha llegado incluso a ser usado por la NASA para simular experimentos a bordo del transbordador espacial Discovery.[12]

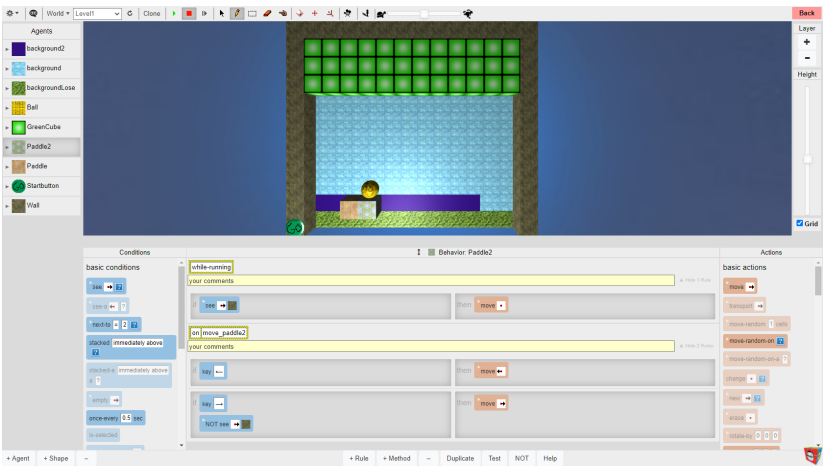


Figura 2.1: Captura de pantalla de AgentCubes

[1] https://agentsheets.com/about

2.1.2. Alice

Desarrollado por la universidad Carnegie Mellon, hace hincapié en una interfaz arrastrar y soltar para limitar dentro de lo posible los problemas técnicos que suelen encontrar los estudiantes que están empezando en el mundo de la programación. El objetivo de Alice[2] es facilitar la creación de entornos 3D en los que el usuario podrá modelar el comportamiento y la apariencia de los objetos que habitan en ellos.[5]

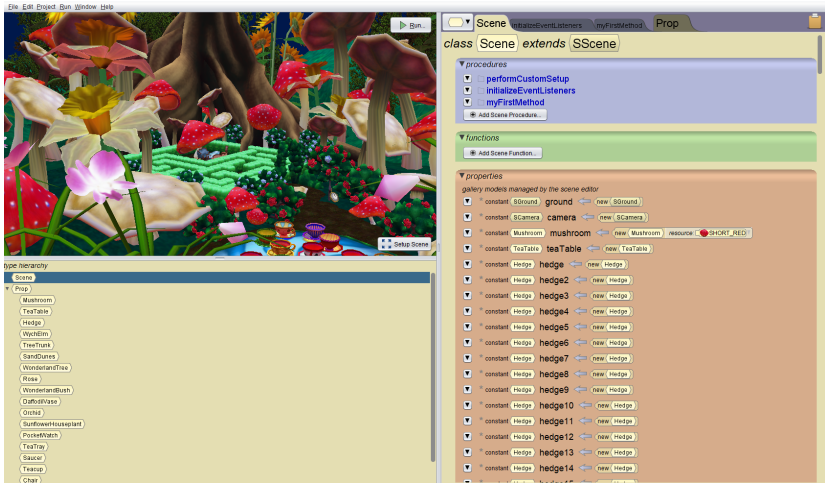


Figura 2.2: Captura de pantalla de Alice

2.1.3. Blockly

A diferencia de la mayoría de los proyectos citados en este capítulo, Blockly[3] es una librería JavaScript enfocada al desarrollo de aplicaciones de programación por bloques en vez de un entorno de desarrollo integrado. Algunos desarrollos que usan Blockly son: App Inventor, CODE, Microsoft MakeCode, etc. Entre sus características más reseñables se encuentran que es extensible, compatible con la mayoría de navegadores y completamente de lado del cliente sin dependencias con el servidor.[3] También muy interesante de cara a la educación es su funcionalidad de traducir el lenguaje de bloques a código sintácticamente correcto en varios lenguajes de programación convencionales (JavaScript, Python, PHP, Lua y Dart entre otros).

[2] https://www.alice.org/
[3] https://developers.google.com/blockly

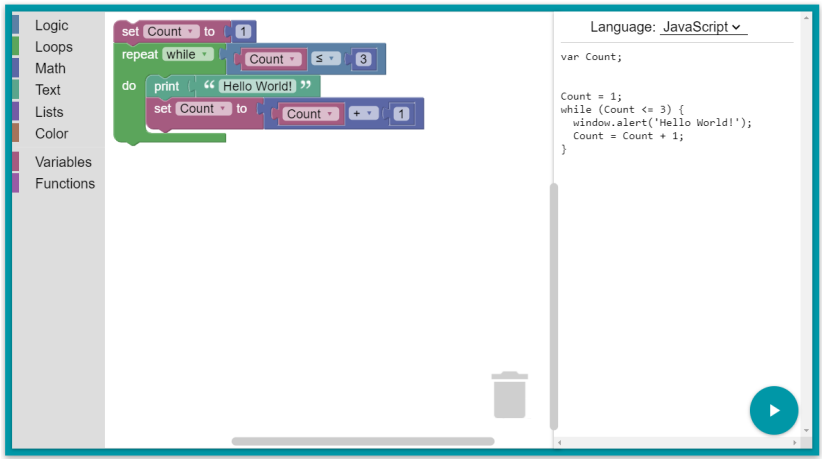


Figura 2.3: Captura de pantalla de Blockly

2.1.4. Etoys

Squeak Etoys[4] es un entorno de programación libre basado en bloques e inspirado por lenguajes como LOGO o Hypercard, Etoys tiene un enfoque en la multimedia con soporte para sonido, vídeos, imágenes y por supuesto gráficos en dos y tres dimensiones. Algo que lo diferencia de la mayoría de programas de este tipo es que incluye la característica de poder compartir el entorno en tiempo real a través de internet lo que abre muchas posibilidades de cara a la educación, tanto en refuerzo de lo explicado en el aula como a permitir una educación completamente a distancia que es algo de especial relevancia en estos tiempos.[10]

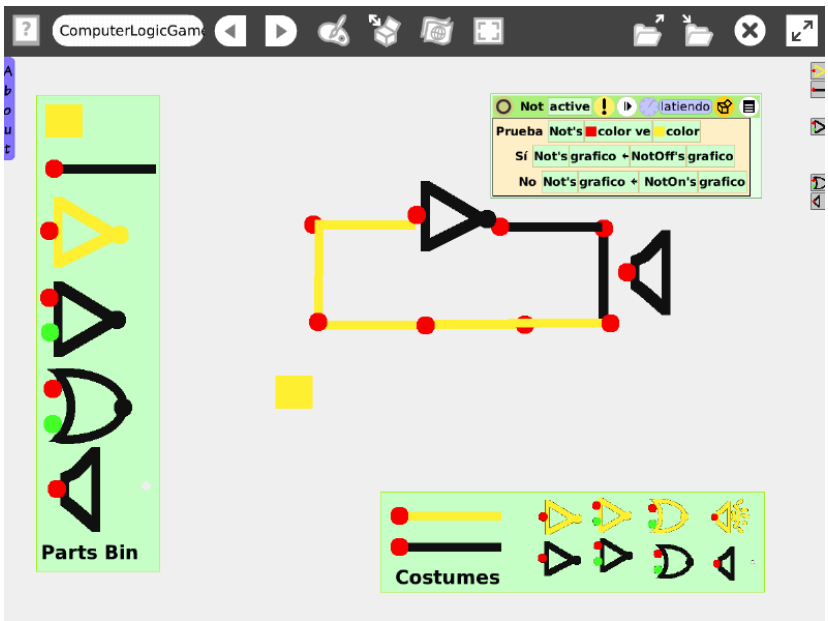


Figura 2.4: Captura de pantalla de Etoys

[4] http://www.squeakland.org/

2.1.5. Hackety Hack

Al igual que Etoys, Hackety Hack[5] es una aplicación de código abierto pero se aleja del paradigma de bloques tan común en estos entornos para optar por enseñar los fundamentos de la sintaxis de Ruby lo que puede verse de cierta forma como una barrera de entrada para los estudiantes más jóvenes o inexpertos sin embargo eso trae consigo la ventaja de enseñar algo más próximo a lo que se usa en el mundo real.[2]

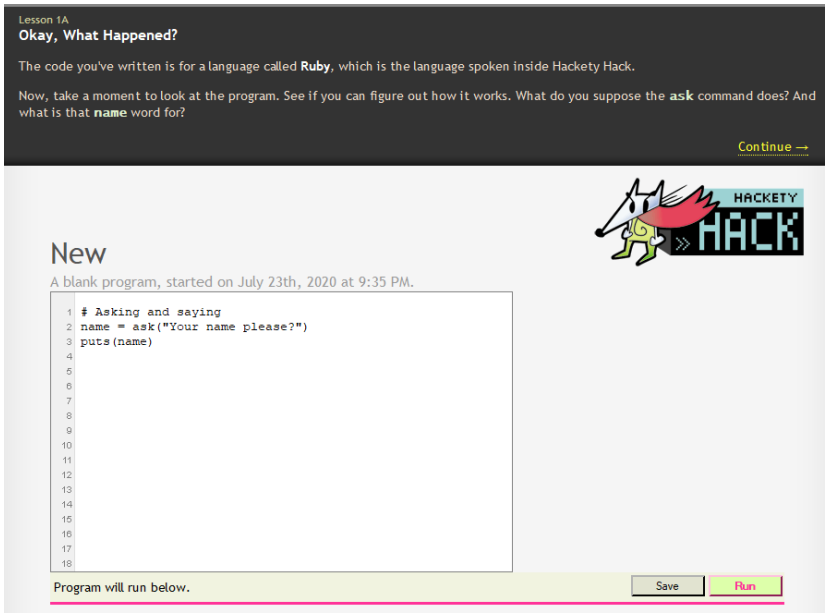


Figura 2.5: Captura de pantalla de Hackety Hack

2.1.6. Karel

Propuesto por Richard E. Pattis en el libro Karel[6] the Robot su metodología consiste en limitar el lenguaje a una serie de comandos imperativos sencillos cuyas acciones están dirigidas a un robot que las ejecuta gráficamente. Algunas de estas instrucciones hacen que el robot avance, se mueva a la izquierda, tome un *beeper*… e incluye conceptos como procedimientos y diversas estructuras de control.[13]

[5] http://www.hacketyhack.net/
[6] https://www.cs.mtsu.edu/ untch/karel/

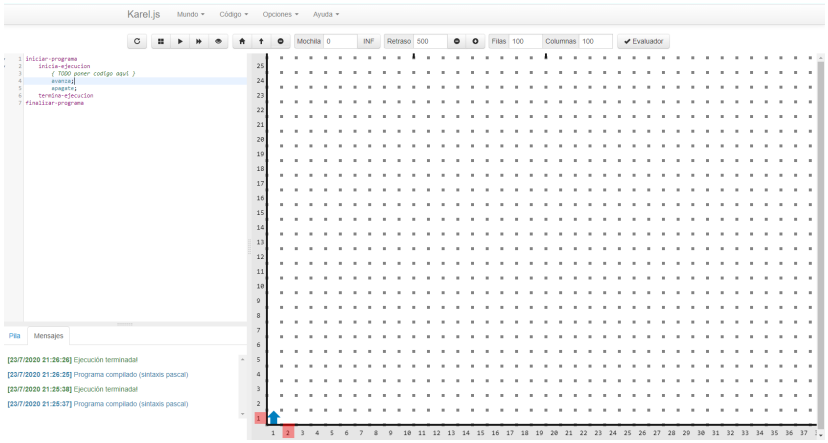


Figura 2.6: Captura de pantalla de Karel.js[7]

2.1.7. Kodu

Enfocado en la creación de historias animadas y videojuegos, Kodu[8] incluye un lenguaje completamente dirigido por eventos en el que los usuarios deben colocar de forma secuencial las fichas que forman las instrucciones del lenguaje.[8] Es gratis, está desarrollado por Microsoft y se puede controlar completamente con un mando de Xbox 360 lo que es una ventaja a la hora de introducir en la programación a personas de corta edad que posiblemente tengan más familiaridad con estos dispositivos que con el teclado y ratón.

Figura 2.7: Captura de pantalla de Kodu

2.1.8. Logo

Logo[9] es un dialecto de Lisp diseñado para la enseñanza de la programación y tiene una filosofía basada en el constructivismo que en palabras llanas ve el aprendizaje como un proceso en el que se genera conocimiento a la vez que la persona interactúa con su mundo y con otras personas.[7] La parte más famosa de Logo es su tortuga que se mueve usando comandos sencillos (forward, right, etc) y deja un rastro de por donde ha pasado permitiendo realizar toda clase de dibujos.

[8] https://www.kodugamelab.com/
[9] https://el.media.mit.edu/logo-foundation/

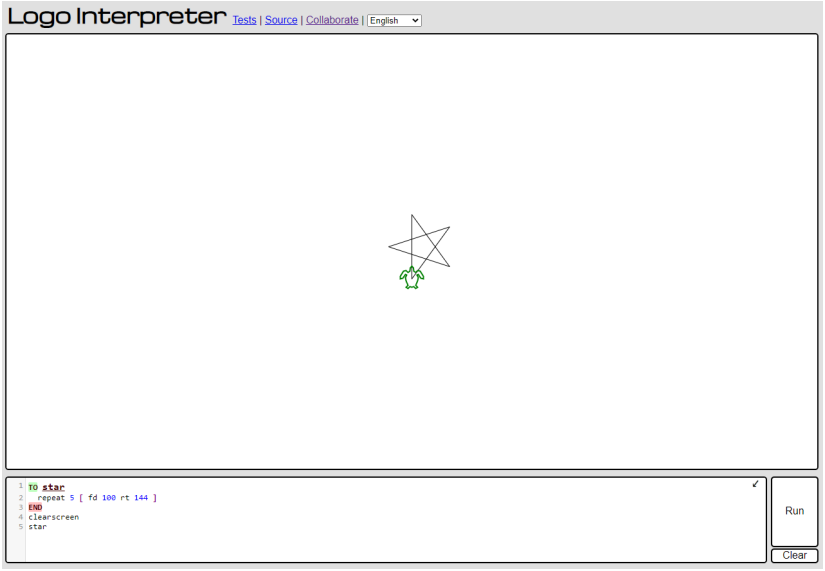


Figura 2.8: Captura de pantalla de Logo Interpreter[10]

2.1.9. Scratch

Scratch[11] es un entorno de desarrollo que permite a los estudiantes programar de forma gráfica usando bloques que encajan entre ellos a modo de piezas de puzzle solo si son compatibles entre sí y pueden representar desde variables de todo tipo y bucles hasta eventos y *multithreading*. Permite crear historias interactivas, animaciones y juegos y compartirlos con el resto de usuarios.[11]

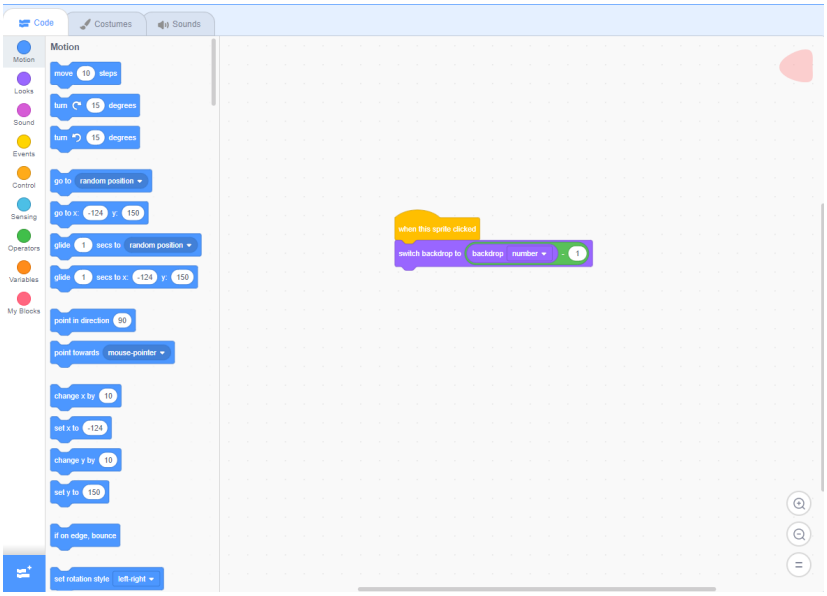


Figura 2.9: Captura de pantalla de Scratch

[11] https://scratch.mit.edu/

2.1.10. Snap!

Snap![12] es una reimplementación de Scratch extendida para soportar que el usuario pueda crear sus propios bloque de construcción a modo de procedimientos y soporta listas entre otras cosas lo que lo propone como una alternativa seria para enseñar programación incluso a estudiantes universitarios.[1]

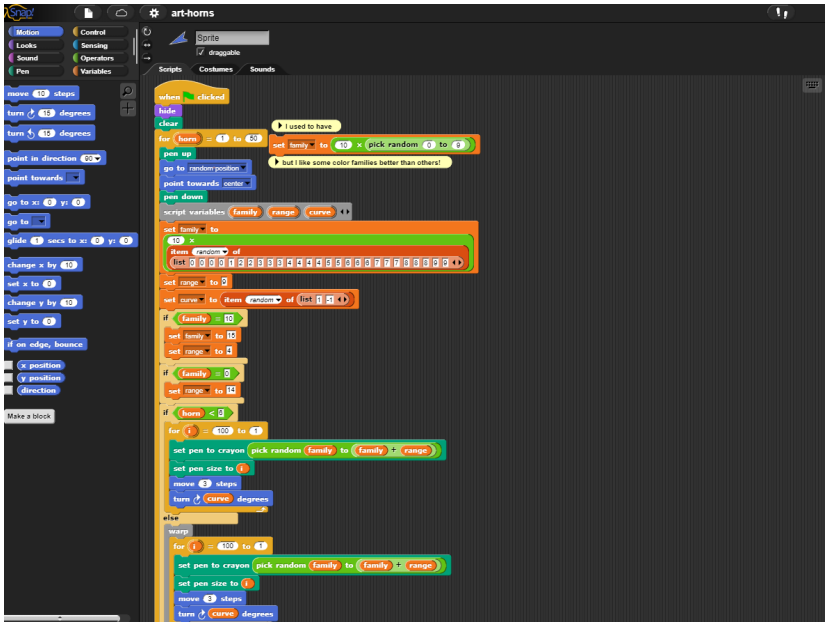


Figura 2.10: Captura de pantalla de Snap!

2.2. ROBOTIC DENTRO DE LA PROGRAMACIÓN EDUCATIVA

Como se ha podido apreciar en las descripciones de proyectos anteriores surgen ciertos patrones en este tipo de lenguajes por ejemplo el uso de multimedia, representación gráfica del significado del programa, programación por bloques, simplificación de las instrucciones lo máximo posible, etc, son algunas de las estrategias que usan para resultar más accesibles a las personas jóvenes sin conocimientos previos.

RoboTIC posee un enfoque mixto entre todas ellas sobre todo tomando inspiración de Karel y Logo en cuanto al uso de ciertas estructuras de control e instrucciones imperativas simples pero con doble componente gráfico: se representa de forma visual el flujo del programa y las instrucciones que lo forman (tomando la forma de diferentes tramos de carretera) y además los resultados que genera su ejecución.

El uso completo de metáforas gráficas para representar instrucciones (un tramo carretera con un icono en vez de un comando textual *move 10*) no es casual y se ha elegido para rebajar todo lo posible la barrera de entrada a la aplicación minimizando la entrada de comandos a una serie de botones que generan las piezas de carretera adecuadas además de utilizar la realidad aumentada para favorecer una experiencia fluida entre el usuario y el lenguaje.

[12] https://snap.berkeley.edu/

2.3. REALIDAD AUMENTADA

Existen múltiples formas y dispositivos para lograr una experiencia de realidad aumentada cada uno con sus ventajas y desventajas. Podemos dividir los dispositivos en dos categorías: no especializados y especializados, y a su vez en los especializados podemos distinguir entre aquellos que están destinados a usarse como *heads-up displays* de los que buscan una inmersión profunda del usuario en la experiencia de AR. A continuación se muestra una lista con un ejemplo de cada una de estas calificaciones:

- *No especializado.* Computador más cámara: usando alguna de las múltiples bibliotecas existentes en el mercado (ARKit, ARCore, Vuforia, EasyAR y ARToolKit entre muchas otras), una cámara y un computador (un ordenador de sobremesa, un tablet, un smartphone, etc) se pueden diseñar sistemas de realidad aumentada cuya ventaja principal es la poca inversión monetaria que deberán llevar a cabo los usuarios finales en cuanto a hardware en la mayoría de los casos debido a la masiva adopción de dispositivos como los smartphones (aproximadamente tres billones de personas en el mundo tienen a día de hoy un teléfono inteligente). La principal desventaja de esta solución es la menor inmersión respecto a opciones de hardware especializado.
- *Especializado, heads-up display.* Google Glass Enterprise Edition 2: desarrollada por Google con el objetivo de producir una computadora ubicua destaca por ser un aparato ligero (46g), potente y con un aspecto que recuerda bastante a unas gafas normales pero con la desventaja relativa de ser más un *heads-up display* que un dispositivo de realidad aumentada con finalidad de inmersión lo cual las convierte en inadecuadas para este proyecto.
- *Especializado, centrado en inmersión.* Microsoft Hololens: proporcionan una gran sensación de inmersión con una comunidad grande detrás y unas especificaciones técnicas más que correctas. Podemos citar el peso y el precio como desventajas.

Para este proyecto se ha llegado a la conclusión de que la opción más interesante es realizar la implementación para dispositivos de la categoría especializada y centrada en inmersión y se ha elegido finalmente Microsoft Hololens como hardware especifico al ser apropiado para la cuestión y además el único accesible al alumno actualmente lo cual no quita que sería interesante que en un futuro funcionara en más aparatos del mismo tipo o incluso en computadores de escritorio o smartphones.

CAPÍTULO 3

OBJETIVO

3.1. OBJETIVO GENERAL

El objetivo general de este proyecto es diseñar y desarrollar el prototipo de un juego serio que contribuya al aprendizaje de la programación de personas en edad escolar mediante metáforas visuales y realidad aumentada.

El sistema final comprenderá el software desarrollado así como una unidad del *headset* Microsoft HoloLens. Además del juego en sí, también se programará un editor de niveles para que tanto jugadores como educadores puedan crear y compartir sus propios niveles.

3.2. OBJETIVOS ESPECÍFICOS

Estos objetivos generales pueden ser desglosados en diferentes objetivos más concretos que se expondrán a continuación.

Facilidad de uso: al estar orientado principalmente a personas de corta edad sin experiencia previa en entornos de programación el sistema final deberá ser lo más simple y directo de usar.

Interacción natural: las entradas al software serán gestuales y en consonancia con el requisito anterior deberán contar con la menor cantidad de gestos posible. Además la salida del software al usuario se realizará a través de un sistema de realidad aumentada.

Mantenibilidad y escalabilidad: el proyecto tiene que estar organizado de forma que contribuya a la escalabilidad y documentado de forma extensiva para facilitar la mantenibilidad del mismo y la adición de nuevas características.

Edición de niveles: la creatividad es una parte importante de este proyecto y por tanto los usuarios como los educadores deberán poder editar y crear sus propios niveles, los cuales se guardarán en formato JSON para fomentar el intercambio de los mismos.

Posibilidad de ser portado a otras plataformas: es importante que las herramientas utilizadas en el desarrollo del proyecto abran la puerta a que funcione en diferentes sistemas de realidad aumentada (o incluso en versión de escritorio) si así se considera oportuno en un futuro.

CAPÍTULO 4

METODOLOGÍA

De entre todas las alternativas posibles la metodología empleada para la construcción de este videojuego ha sido basada en el Desarrollo Rápido de Aplicaciones (*RAD, Rapid Application Development*), debido a que se ha considerado interesante el tener disponibles prototipos funcionales del programa lo antes posible en el ciclo de desarrollo.

El proceso ha consistido primero en la elicitación de los requisitos que debe cumplir el sistema y que se ven expuestos en la sección de objetivos de este documento. Aparte de los objetivos hay requisitos más concretos que debe satisfacer el programa entre los que podemos citar qué tramos de carreteras habrá, qué acciones podrá llevar a cabo el robot, de qué forma interactuará el usuario con el sistema, etc, que han sido estudiados en base al proyecto previo en el que RoboTIC está basado mediante la visualización de diversos vídeos principalmente. Tal como se menciona en la introducción RoboTIC parte de un proyecto previo del que no se ha tomado absolutamente nada salvo ideas sobre la jugabilidad y apariencia que habrá de tener el sistema final.

Una vez con los requisitos claros se ha comenzado a prototipar. RoboTIC consta de cuatro prototipos en total pero en esta parte se encajan del uno al tres. A continuación se expone de manera ordenada qué se ha realizado en cada uno de ellos:

- Prototipo 1. En esta primera etapa del desarrollo se ha procedido a poner en pie los sistemas esenciales: carga de los niveles, escenario básico, botones, colocación de las carreteras y una forma muy poco óptima de navegación sobre las mismas usando *pathfinding*. Es considerado una etapa de adecuación a las herramientas pues no se contaba con ningún conocimiento previo de C#, Unity o desarrollo de videojuegos. Al momento de completar este prototipo el proyecto ya contaba con una jugabilidad definitiva pero con bastantes fallos, pobre optimización y sin ningún tipo de soporte para realidad aumentada.
- Prototipo 2. Una vez con las bases completadas se comenzó la construcción del segundo prototipo cuyos cambios principales son un sistema de paso de mensajes para evitar en la medida de lo posible el acoplamiento entre objetos, un sistema de movimiento en las carreteras estable y optimizado y soporte para la realidad aumentada.
- Prototipo 3. Si bien el anterior prototipo tenia soporte para realidad aumentada realmente no estaba funcionando nativamente en las *HoloLens* por lo que fueron necesarios varios ajustes para poder generar un ejecutable. Muy importante de esta versión es por fin la aparición del editor de niveles el cual es un objetivo requerido en este TFG.

La fase final del proyecto consta de un prototipo, el prototipo 4. Durante este periodo del desarrollo del sistema se han hecho las pruebas necesarias y la corrección de bugs de última hora que hayan podido surgir. Otra característica muy importante de esta fase es que se ha generado la mayoría de documentación relativa al proyecto usando la herramienta Doxygen y también se ha escrito esta memoria.

4.1. MEDIOS UTILIZADOS DURANTE EL DESARROLLO

En esta sección se presentará el equipamiento y los medios usados durante el desarrollo del proyecto.

4.1.1. Medios hardware

Los medios hardware que han sido necesarios para el desarrollo de este proyecto han sido los siguientes:

- Asus K551LN. Ordenador portátil propiedad del alumno donde se desarrollará todo el proyecto.
- Microsoft HoloLens.[1] Proporcionado por Furious Koalas Interactive será el entorno hardware para el que se desarrolle toda la parte de realidad aumentada del proyecto.

4.1.2. Medios software

Los diferentes programas que han sido requeridos para el desarrollo del trabajo han sido:

- Windows 10 Home. Sistema operativo en el que este proyecto será desarrollado. Imprescindible usar este y no otro por motivos de compatibilidad con la plataforma de realidad aumentada elegida.
- GitHub Desktop.[2] Herramienta de control de versiones usada para gestionar el repositorio de código fuente. Se ha elegido este programa ya que el repositorio estará alojado en GitHub[3] y se considera una opción muy apropiada para trabajar sobre esa plataforma.
- Blender.[4] Programa de código abierto para el modelado en 3D.
- Unity.[5] Plataforma de desarrollo en tiempo real y motor de videojuegos seleccionado para este proyecto.
- Microsoft Visual Studio Community 2019.[6] IDE utilizado para escribir las partes de código C# de la aplicación.
- Doxygen [7]: Herramienta de generación automática de documentación a partir de ficheros de código fuente con comentarios.

[1] https://docs.microsoft.com/es-es/hololens/hololens1-hardware
[2] https://desktop.github.com/
[3] https://github.com/
[4] https://www.blender.org/
[5] https://unity.com/es
[6] https://visualstudio.microsoft.com/es/vs/community/
[7] https://www.doxygen.nl/index.html

4.1.3. Lenguajes de programación y bibliotecas

- C#: Lenguaje de programación orientado a objetos desarrollado y estandarizado por Microsoft con sintaxis derivada de C/C++ y un modelo de objetos basado en el de Java. Se usará este lenguaje puesto que es el que mejor soporte ofrece por parte de Unity.
- Microsoft Mixed Reality Toolkit[8]: Proyecto de Microsoft para ofrecer una serie de componentes y características usadas para acelerar el desarrollo de aplicaciones de realidad mixta.
- LeanTween[9]: Biblioteca de *tweening* usada para el movimiento del personaje sobre las carreteras.

[8] https://github.com/microsoft/MixedRealityToolkit-Unity
[9] https://github.com/dentedpixel/LeanTween

CAPÍTULO 5

RESULTADOS

En este capítulo se trata el sistema finalizado y la arquitectura que se ha desarrollado a lo largo de estos meses. Para mantener el documento breve se ha decidido hablar solamente de las partes más importantes, aún así si el lector desea ahondar en mayor profundidad en el funcionamiento del programa se ofrece un anexo con la documentación generada por Doxygen.

5.1. CONSIDERACIONES PREVIAS

El objetivo de RoboTIC es dirigir al robot a la meta de cada nivel sorteando una serie de obstáculos (lava, agua, desniveles, etc) escribiendo para ello algoritmos que toman la forma de carreteras. Esta premisa trata de enseñar a los jugadores las bases de la programación de ordenadores sin ser de ninguna forma abrumadora para principiantes. El robot entiende un conjunto de instrucciones a las que se han añadido estructuras de control para formar un sencillo lenguaje imperativo de fácil aprendizaje.

El repositorio donde se encuentra alojado el código fuente de RoboTIC se puede visitar a través del siguiente enlace:

Listado 5.1: Enlace al repositorio

```
1 https://github.com/OhEsPaco/RoboTIC
```

En Unity los proyectos están compuestos por escenas las cuales son estructuras de datos que contienen los objetos (gráficos, sonidos, lógica, etc) que forman el juego y puede haber un número indefinido de ellas. En este caso se ha decidido que sea solo una escena la que englobe la totalidad del juego (Main.unity) y para facilitar la organización se han ordenado por temáticas los objetos que la forman. A continuación se comentarán los aspectos más importantes de la arquitectura, prestando especial atención a aquellos componentes del sistema que cuentan con mayor complejidad.

Figura 5.1: Main.unity usando GameObjects vacíos como separadores

Se puede apreciar la organización de RoboTIC de una forma mejor con el siguiente diagrama general del sistema:

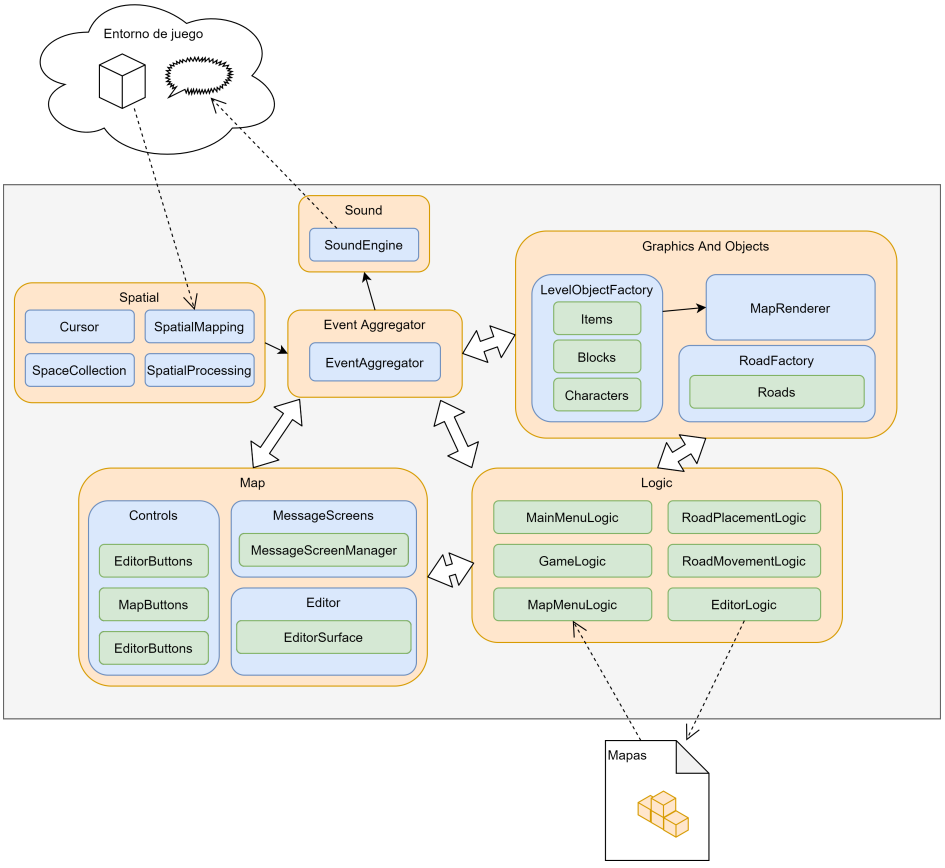


Figura 5.2: Diagrama general del sistema

5.2. SUBSISTEMAS LÓGICOS

En RoboTIC las clases que representan la lógica de las distintas partes del programa son GameLogic, EditorLogic, MapMenuLogic, RoadPlacementLogic, MainMenuLogic y RoadMovementLogic y se encuentran dentro de la carpeta Assets/Scripts/GameLogic. Todas ellas, a excepción de RoadMovementLogic, implementan un patrón singleton pues al contar con que habrá una sola instancia de cada clase en la escena y poder acceder a ella mediante una referencia estática facilita mucho la comunicación entre estos objetos cuyo funcionamiento debe estar bien sincronizado.

- GameLogic (ver Anexo C 5.30). Entre sus responsabilidades se encuentran almacenar el estado de la partida, determinar si se pierde o se gana y enviar al personaje principal las acciones que debe llevar a cabo en cada momento. Se trata en la sección sobre el movimiento entre bloques.
- EditorLogic (ver Anexo C 5.18). Dirige el funcionamiento del editor de niveles y además se encarga de guardar los mapas generados para que puedan ser jugados más tarde.
- MapMenuLogic (ver Anexo C 5.47). Su función dentro de RoboTIC es cargar los niveles desde los archivos JSON correspondientes y mostrarlos al usuario para que pueda elegir el que quiera.
- RoadPlacementLogic (ver Anexo C 5.109). Contiene la lógica para formar la carretera de forma correcta. Se habla en profundidad de este proceso en la sección sobre la formación de las carreteras.
- RoadMovementLogic (ver Anexo C 5.107). Se encarga del movimiento sobre las carreteras del robot pequeño que simboliza el flujo del programa. Es la única que no implementa un patrón Singleton y siempre se interactúa con ella a través del agregador de mensajes. Se pueden consultar más detalles en la sección sobre el movimiento en las carreteras.
- MainMenuLogic (ver Anexo C 4.44). Es la lógica del menú que se le presenta al jugador nada más ejecutar el juego. Permite elegir entre jugar, abrir el editor o salir del programa.

5.3. EL NIVEL

Una parte muy importante de RoboTIC son los niveles pues contienen los retos que tendrá que superar el jugador. A lo largo del desarrollo del proyecto se ha intentado que estos niveles no sean simplemente algo que superar sin más pues afectaría negativamente a la cantidad de tiempo que los usuarios pasan con el juego disminuyendo con ello la posibilidad de que aprendan los conceptos relacionados con la programación de ordenadores que intenta transmitir RoboTIC pues es algo que requiere gran cantidad de práctica y dedicación.

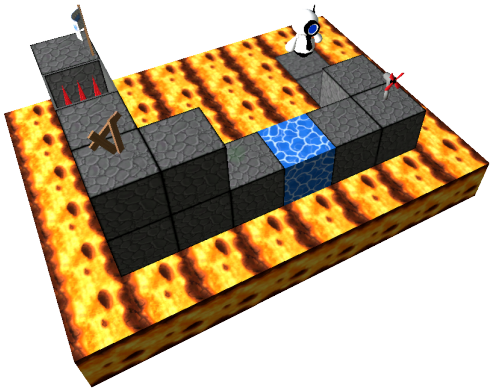


Figura 5.3: Ejemplo de nivel de RoboTIC

Los niveles por tanto se deben poder compartir y crear aparte de jugar por lo que se ha creído conveniente usar JSON para guardarlos pues es un sistema de serialización basado en texto sencillo así que se podrán estudiar y editar con un simple editor de textos (o con el editor de niveles que incorpora el juego y que será tratado en detalle más adelante) favoreciendo la creatividad y las posibilidades jugables, así como un sentimiento de comunidad entre los usuarios.

Los datos que definen un nivel de juego son los siguientes:

- *Nombre*. Aunque no es imprescindible se ha decidido incluir el nombre de cada nivel por motivos de identificación de los mismos. A los niveles construidos con el editor se les asigna automáticamente un UUID.
- *Posición del jugador*. Posición donde se encontrará el robot al inicio del nivel.
- *Posición de la meta*. Punto al que debe llegar el robot para que la partida se considere victoriosa.
- *Orientación del jugador*. Orientación inicial del jugador. Es un entero entre 0 y 3 para dar un total de cuatro orientaciones posibles. Cada orientación se calcula como un giro de $(90^o \cdot$ *Orientación del jugador)* grados en el eje *y*.
- *Número de instrucciones disponibles*. Cantidad de cada una de las diferentes instrucciones posibles, por ejemplo las veces que el jugador puede avanzar, girar, etc.
- *Tamaño del nivel*. Como veremos a continuación los items y los bloques del nivel se almacenan en un vector unidimensional de enteros y es necesario conocer el tamaño de cada dimensión para poder calcular qué índice de ese vector corresponde a una coordenada *(x, y, z)*.
- *Bloques e items*. Vector unidimensional de enteros en el que cada uno de estos representa

un bloque o item diferente. Dado un punto *(x, y, z)* se determina a qué elemento del vector unidimensional corresponde usando la formula *(x + z · tamaño del nivel en x + y · (tamaño del nivel en x · tamaño del nivel en z))*

Listado 5.2: Estructura de los niveles

```
[System.Serializable]
public class LevelData
{
    //Nombre del nivel
    public string levelName;
    //Tamaño del nivel en 'x', en 'y' y en 'z'
    public List<int> levelSize;
    //Posición inicial del jugador
    public List<int> playerPos;
    //Orientación inicial del jugador
    public int playerOrientation;
    //Posición de la meta
    public List<int> goal;
    //Instrucciones disponibles
    public AvailableInstructions availableInstructions;
    //Bloques e items
    public List<int> mapAndItems;

}
[System.Serializable]
public class AvailableInstructions
{
    //Condición
    public int condition;
    //Bucle
    public int loop;
    //Girar a la derecha
    public int turnRight;
    //Girar a la izquierda
    public int turnLeft;
    //Saltar
    public int jump;
    //Avanzar
    public int move;
    //Usar item
    public int action;
}
```

A continuación se muestra un pequeño nivel de juego en formato JSON para aumentar la claridad de la explicación y que se espera exactamente que contenga cada variable:

Listado 5.3: Nivel en formato JSON

```
{
  "levelName": "Test␣Level␣Spikes",
  "levelSize": [ 3, 2, 4 ],
  "playerPos": [ 1, 1, 0 ],
  "playerOrientation": 0,
  "goal": [ 1, 1, 3 ],
  "availableInstructions": {
    "condition": 0,
    "loop": 0,
    "turnRight": 0,
    "turnLeft": 0,
    "jump": 1,
    "move": 1,
    "action": 0
  },
  "mapAndItems": [
    1,3,1,
    1,3,1,
    1,1,1,
    1,3,1,

    0,0,0,
    0,26,0,
    0,0,0,
    0,0,0
  ]
}
```

La carga de los niveles en formato JSON es responsabilidad de la clase MapMenuLogic que los transforma en objetos del tipo LevelData (ver Anexo C 5.40) y el guardado de los mismos si el usuario decide crearlos dentro del editor de niveles incluido en el juego se lleva a cabo en el método SaveMap de la clase EditorLogic. Una vez se encuentran los niveles dentro del sistema es hora de generar los bloques e items que representa esta estructura de datos.

5.4. BLOQUES E ITEMS

En esta sección se hablará sobre qué bloques e items va a encontrar el jugador en RoboTIC desde un punto de vista técnico aunque se ha considerado constructivo mostrar primero la instanciación de estos objetos y cómo se colocan para formar un nivel.

5.4.1. Instanciado de bloques e items

En RoboTIC se ha centralizado la instanciación de objetos de juego en la clase LevelObjectFactory (ver Anexo C 5.42). Cada uno de estos objetos es un *prefab* (GameObject completo a modo de plantilla) y si bien se podrían instanciar desde cualquier parte del programa sería problemático de cara a la mantenibilidad del código. Esta clase contiene el método GetGameObjectInstance que dado un identificador numérico (los que se almacenan en mapAndItems de la clase LevelData vista anteriormente) retorna un LevelObject (ver Anexo C 5.41) listo para usar.

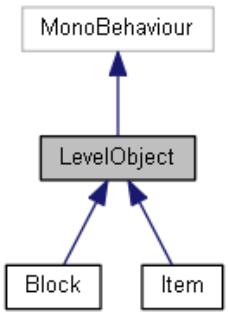


Figura 5.4: Diagrama de herencias de LevelObject

LevelObject es una clase abstracta que está extendida por Block (ver Anexo C 5.5) e Item (ver Anexo C 5.38) y contiene declaraciones de métodos que son comunes a ambos tipos de objetos además de permitir almacenar sus referencias conjuntamente dentro de la misma estructura.

5.4.2. Construcción de los mapas

Hasta ahora se ha tratado cómo representar los niveles del juego y cómo instanciar los bloques e items que los componen pero todo esto sería una tarea inútil si no existiera forma de poner a cada uno de estos objetos en su posición definitiva.

La clase MapRenderer (ver Anexo C 5.48) lleva a cabo esta tarea de instanciar y colocar los componentes de un mapa cuando recibe un mensaje del tipo MsgRenderMapAndItems el cual incluye principalmente el vector unidimensional conteniendo los identificadores de los bloques e items apropiados, el número de bloques en *x*, *y*, *z* y el *padre* (GameObject que contendrá los bloques instanciados). A continuación la fórmula para calcular la coordenada global definitiva de cada bloque:

$$padre.xyz + longitud \times bloque.xyz$$

El parámetro *padre.xyz* es un vector con las coordenadas globales del objeto padre y *bloque.xyz* la posición del bloque o item dentro del vector de identificadores. *longitud* es lo que mide cada uno de los lados de un bloque, está definido en MapRenderer y es muy importante pues las animaciones del robot principal se calculan en tiempo real para adaptarse a la extensión del escenario además de permitir cambiar el tamaño de bloques e items y que estos se sigan ubicando correctamente.

Instanciar GameObjects es un proceso costoso y para evitar bajones en la tasa de cuadros por segundo se ha recurrido al uso de corrutinas[1] con el fin de parar la generación de bloques cada vez que se completa una fila del mapa y retomarla durante el *frame* siguiente. Las corrutinas son una característica extremadamente útil de Unity que permite suspender la ejecución de una función y restablecerla más tarde (se puede ver como una manera de imitar el paralelismo).

[1] https://docs.unity3d.com/ScriptReference/Coroutine.html

5.4.3. Movimiento entre bloques

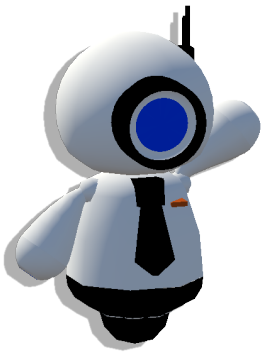


Figura 5.5: Este es RoboTIC

RoboTIC es el personaje principal del juego en sus dos versiones: la grande que se mueve por los bloques del escenario y la pequeña que hace lo mismo pero por las carreteras que simbolizan algoritmos y que se tratará en proximas secciones.

El movimiento del robot grande se calcula en la clase GameLogic que además de eso contiene el estado actual de la partida y determina si se gana o se pierde. El proceso general para que el robot ejecute una acción es comprobar si es posible llevarla a cabo y enviar un mensaje MsgBigRobotAction (ver Anexo C 5.58) que será recibido por la clase BigCharacter (ver Anexo C 5.4). Las acciones que entiende el robot son las siguientes:

Listado 5.4: Acciones del robot principal

```
public enum BigRobotActions
{
    //Saltar
    Jump,
    //Avanzar
    Move,
    //Girar 90º a la izquierda
    TurnLeft,
    //Girar 90º a la derecha
    TurnRight,
    //Animación de ganar
    Win,
    //Animación de perder
    Lose
}
```

Un problema que surgió pronto en el desarrollo del juego es que GameLogic determina las acciones que debe llevar a cabo el robot de una manera extremadamente rápida pero las animaciones que produce BigCharacter deben durar un tiempo concreto y que no empiecen nuevas mientras haya alguna en proceso.

La manera de solucionar esto ha sido implementar una cola de acciones. Cada vez que BigCharacter recibe un mensaje de ejecutar una acción, se calculan los parámetros necesarios y acto seguido se introduce en una lista de corrutinas. Esta aproximación tiene la ventaja de que asegura que cada acción se va a ejecutar en orden y va a durar el tiempo adecuado.

Listado 5.5: Método DoMove de BigCharacter

```
private void DoMove(in Vector3 target)
{
    AddAction(MoveCoroutine(actionSpeed, target));
}
```

Avanzar y el resto de animaciones concernientes al movimiento del robot (saltar y girar) se calculan en el momento pues como se mencionó anteriormente el tamaño de los bloques puede ser modificado.

Listado 5.6: Cálculo de la animación de avanzar

```
private IEnumerator MoveCoroutine(float speed, Vector3 target)
{
    NotifyStartOfAction();
    Vector3 startPos = transform.position;
    float distance = Vector3.Distance(startPos, target);

    for (float i = 0; i <= 1;)
    {
        i += ((speed * Time.deltaTime) / distance);
        transform.position = Vector3.Lerp(startPos, target, i);
        yield return null;
    }

    transform.position = target;
    NotifyEndOfAction();
}
```

Los métodos NotifyStartOfAction y NotifyEndOfAction son los que informan sobre cuando empieza y acaba una acción para que no se pisen entre si. En el siguiente fragmento se ve como cuando se detecta que no quedan más acciones por realizar se elimina la primera de la lista y se ejecuta.

Listado 5.7: Ejecución de la próxima acción

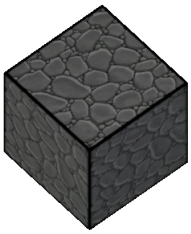
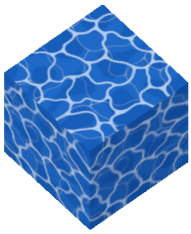
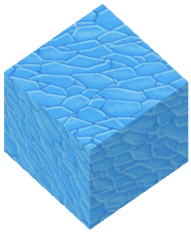
```
private void StartNextAction()
{
    if (actionList.Count > 0 && lastActionFinished)
    {
        IEnumerator coroutine = actionList[0];
        actionList.RemoveAt(0);
        if (coroutine != null)
        {
            StartCoroutine(coroutine);
        }
    }
}
```

5.4.4. Bloques del juego

Los niveles están compuestos principalmente por bloques que pueden tener diferentes propiedades, reacciones a items, etc. Una de sus funciones es la de dar al robot una superficie sobre la que moverse la cual por supuesto no está exenta de gran variedad de riesgos que el jugador debe entender y superar de forma satisfactoria desarrollando algoritmos con las instrucciones pertinentes.

Como se ha visto anteriormente los niveles del juego se guardan dentro de una lista de números enteros y cada uno de esos números hace referencia a un bloque o item en concreto. A continuación se expone una lista con cada uno de los bloques del juego y su función dentro de él:

Tabla 5.1: Breve análisis de los bloques

Bloque	Descripción
	El bloque Solid es la base de los niveles y como tal el jugador puede andar y saltar sobre él sin ningún tipo de penalización. Se puede destruir usando items con el efecto apropiado y se puede obtener al congelar Lava.
	El bloque Water representa agua y es peligroso para el jugador pues si intenta caminar sobre un bloque de este tipo se acabará la partida. Se puede congelar para formar un bloque Ice.
	El bloque Ice resulta de congelar el bloque Water y en esencia funciona como el bloque Solid y al igual que este puede ser tocado sin peligro y destruido con el item adecuado.

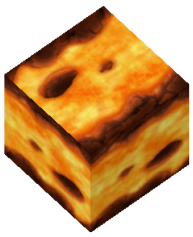
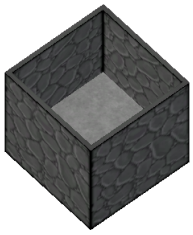
Bloque	Descripción
	El bloque Lava es peligroso y no se puede andar sobre él. Generalmente funciona igual que un bloque Water pero al ser congelado genera el bloque Solid.
	El bloque Spikes representa un agujero con pinchos al fondo por lo que provoca la muerte del robot si se pisa. Puede ser neutralizado o destruido con los items correspondientes.
	El bloque Lift es una plataforma que debe ser activada mediante el uso de items para que el robot pueda caminar por encima. Para aumentar las posibilidades en cuanto a formas de enfrentarse a un nivel también se permite que sea destruida.
	Aunque invisible, el bloque NoBlock es imprescindible para el juego pues permite que el jugador camine a través de él y en general se puede encontrar en cualquier espacio en el que no haya otro bloque.

Tabla 5.1: Breve análisis de los bloques

5.4.5. Funcionamiento de los bloques

Uno de los objetivos al desarrollar el sistema de bloques es que se pudieran añadir nuevos de forma sencilla. En general las propiedades que definen a un bloque son su tipo, sus propiedades y como reacciona a los diferentes items que hay en el juego.

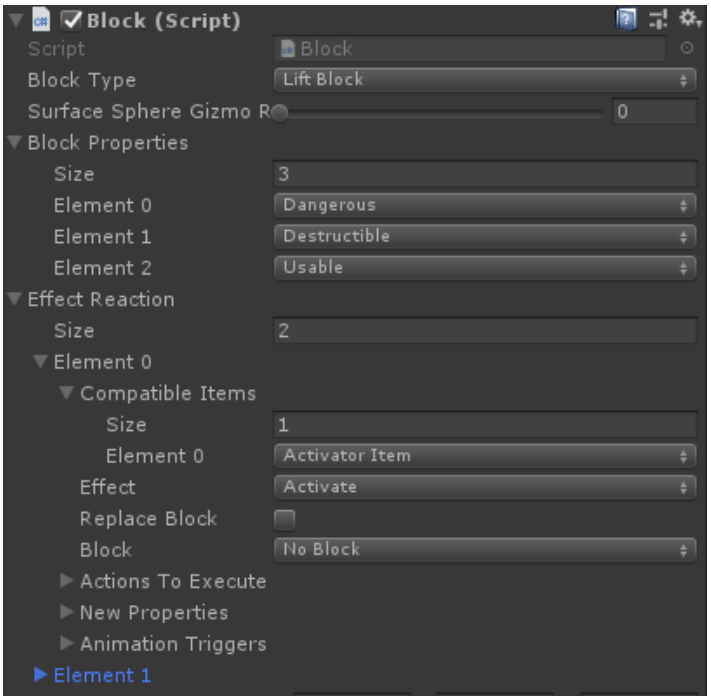


Figura 5.6: Bloque Lift

Se ha aprovechado la serialización de objetos de Unity para modelar los bloques mediante la clase Block como se puede apreciar en la imagen superior. Esta solución tiene la ventaja de que se pueden añadir bloques nuevos muy rápidamente además de modelar como reacciona a cada item y al jugador sin tener que añadir código nuevo. En los siguientes párrafos se detalla para qué sirven las variables más importantes de entre los que forman un bloque.

Block Type es un identificador para saber a qué objeto nos referimos a la hora de cargar el nivel.

Listado 5.8: Bloques posibles

```
public enum Blocks
{
    NoBlock = 0,
    WaterBlock = 1,
    LavaBlock = 2,
    SolidBlock = 3,
    LiftBlock = 4,
    SpikesBlock = 5,
    IceBlock = 6
};
```

Block Properties incluye las propiedades de este bloque respecto al jugador, por ejemplo un bloque con la propiedad *Inmaterial* permitirá que el robot lo atraviese pero uno marcado como *Dangerous* hará que pierda la partida al entrar en contacto con él. Estas propiedades son importantes para el desarrollo del juego pues las estructuras condicionales del lenguaje se basan en que el bloque al que va a avanzar el jugador cumpla o no la propiedad para saber por qué rama continuar la ejecución del algoritmo.

Listado 5.9: Propiedades de los bloques

```
public enum BlockProperties
{
    //Se puede atravesar
    Immaterial,

    //Se puede andar por encima sin peligro
    Walkable,

    //Se pierde la partida al entrar en contacto con él
    Dangerous,

    //Bloque congelado
    Icy,

    //Se puede destruir
    Destructible,

    //Se puede usar
    Usable,

    //Se puede congelar
    Freezable
}
```

Effect Reaction es interesante pues sirve para modelar la respuesta del bloque a los diferentes items del juego. Las reacciones que se pueden formar son infinitas pues para cada una se puede elegir qué item la causa, qué tipo de efecto produce en el bloque *(None, Freeze, Destroy, Activate)*, si hay que reemplazar el bloque antiguo por uno nuevo, acciones a ejecutar *(Use, Destroy, Place, Activate, Rebind)*, se pueden definir las nuevas propiedades que tendrá el bloque tras el efecto y si habrá que ejecutar *animation triggers* para reproducir animaciones concretas sin necesidad de modificar el código.

5.4.6. Items

Aparte de bloques los niveles pueden contener *items* que son una serie de objetos que puede usar el jugador para ayudarse en su tarea de llevar al robot a la meta del nivel. Estos objetos aportan variedad a las partidas y diversas formas de enfrentarse a los obstáculos, por ejemplo un bloque *Water* se podría congelar y destruir para superarlo usando los objetos apropiados en vez de limitarnos a rodearlo.

A continuación se exponen todos los objetos del sistema final:

Tabla 5.2: Breve análisis de los items

Item	Descripción
	El item Flag se usa para marcar la meta de un nivel. Todos los niveles deben tener una bandera y no puede ser tomada por el jugador.

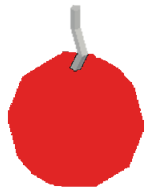
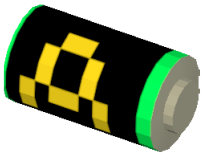
Item	Descripción
	El item Bomb destruye ciertos bloques al ser usado en ellos, principalmente bloque sólidos como Solid o Ice.
	El item Activator activa los bloques con los que se usa (y son compatibles con él). Usando este item en el bloque Lift conseguiremos que suba la plataforma y el robot pueda pasar al otro lado.
	El item Fan congela los bloques líquidos, en este caso Water y Lava transformándolos en sólidos y permitiendo ser pisados por el jugador o destruidos con el objeto Bomb.
	El item Planks representa unas planchas de madera cuyo uso es ser colocadas en el bloque Spikes para convertirlo en seguro para el jugador.

Tabla 5.2: Breve análisis de los items

Los items que se toman durante el recorrido de un nivel se guardan en el inventario. El inventario es una estructura tipo pila por lo que siempre se usará el último objeto recogido. A priori limitante para el jugador, esta característica simplifica en gran medida la planificación de los algoritmos necesarios para superar un nivel al reducir las instrucciones para el manejo de objetos a la instrucción de usar.

5.4.7. Funcionamiento de los items

El comportamiento de los items se ve representado mediante la clase Item que contiene una serie de atributos serializables como por ejemplo el tipo de item que es, si lo puede coger el jugador, etc. Al igual que con los bloques, es muy fácil añadir nuevos items al juego o modificar los existentes simplemente cambiando algunos parámetros. A continuación se explica detalladamente el significado de cada uno de estos atributos:

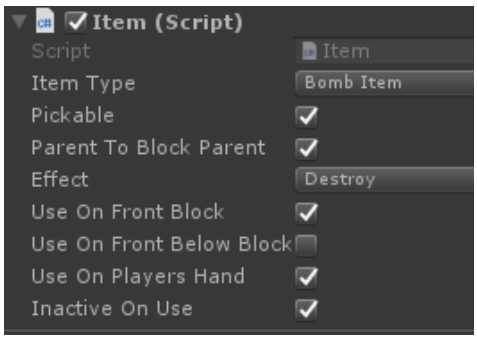


Figura 5.7: Atributos serializables de Item.cs

- *Item Type*. Identifica el tipo de item que representa este objeto. En el sistema actual se puede elegir entre *PlankItem, FanItem, FlagItem, ActivatorItem y BombItem* y se podrían añadir más de ser necesario.
- *Pickable*. Indica si el objeto se puede tomar o no en el sentido de hacer que forme parte del inventario del jugador. Esto es muy útil para ciertos objetos como la bandera u otras decoraciones que no deben moverse de su sitio.
- *Parent To Block Parent*. Hace que el objeto tome como padre el bloque sobre el que se utiliza. Su utilidad principal es hacer que el item una vez usado se torne parte del bloque, imprescindible por ejemplo a la hora de usar las planchas con el bloque *spikes*.
- *Effect*. Es el efecto que tendrá el item al usarse con un bloque. Los efectos que podemos encontrar actualmente en el sistema son *None, Freeze, Destroy y Activate*.
- *Use On Front Block*. Indica si el item podrá usarse en el bloque que se encuentre justo enfrente del robot.
- *Use On Front Below Block*. Indica si el item podrá usarse en el bloque que se encuentre justo debajo del que está enfrente del robot.
- *Use On Players Hand*. Significa que el item se usará en la mano del robot en vez de en la superficie del bloque al que está destinado este objeto.
- *Inactive On Use*. Tras ser usado el objeto será marcado como inactivo si esta propiedad es verdadera.

5.5. SISTEMA DE CARRETERAS

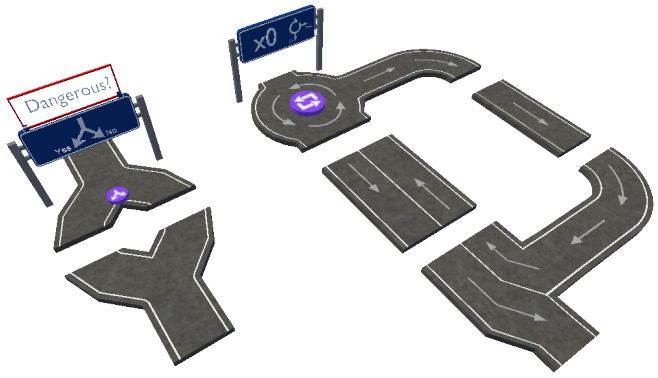


Figura 5.8: Imagen de todas las carreteras

El sistema de carreteras es una parte de vital importancia para el proyecto, al fin y al cabo representa la totalidad de las metáforas sobre el que se basa la experiencia de cara al usuario y con diferencia ha sido la parte que mayor tiempo de trabajo ha requerido.

El funcionamiento básico es que el jugador debe hacer click sobre una serie de botones (tratados en detalle en el manual del juego) y eso genera una carretera que realmente es la representación visual de un algoritmo que el robot puede interpretar.

Este sistema está bien diferenciado en dos subsistemas, siendo el primero el que se encarga de colocar las carreteras en su sitio correspondiente en base a las entradas del usuario y el segundo el que permite que un robot pequeño que representa el flujo de ejecución del programa se mueva por estas carreteras.

5.5.1. Tramos y metáforas

La representación de los algoritmos en RoboTIC se apoya en una metáfora que toma diferentes tramos de carretera como bloque de construcción. Esta forma de mostrar algoritmos aporta la claridad y la inmediatez necesaria a la hora de que los estudiantes de programación principiantes aprendan con el menor número de complicaciones posibles. El lenguaje de RoboTIC puede calificarse de tipo imperativo y tiene soporte para estructuras condicionales y bucles que se tratarán en profundidad en los párrafos siguientes.

El bucle representa una estructura de bucle *for* de la que el usuario puede elegir el número de iteraciones, entre cero y nueve, y el robot iterará en ella hasta que el número de repeticiones restantes sea igual a cero. Un ejemplo en lenguaje java de lo que intenta hacer esta estructura es el siguiente:

Listado 5.10: Bucle

```
for(; numeroDeRepeticiones>0; numeroDeRepeticiones--){
    //Hacer algo
}
```

La representación dentro del juego del bucle es la siguiente:

Figura 5.9: Bucle en RoboTIC

La otra estructura que soporta el lenguaje es el condicional (*if*) en la cual se permite seleccionar una carta de condición y dependiendo de si dicha condición se cumple o no en el bloque al que vaya a avanzar el robot el flujo de ejecución tomará un camino u otro.

Figura 5.10: Condicional con la carta *walkable* seleccionada

Las cartas de condición que incluye el juego son las siguientes:

- *Dangerous*. En este caso la condición será verdadera si el bloque de enfrente del robot es peligroso de alguna manera, por ejemplo si es lava o agua y por tanto no debería pisarlo.
- *Destructible*. Si el bloque frente al jugador es destruible la condición será satisfecha.
- *Freezable*. Congelable, por ejemplo en el caso del agua que se puede congelar para formar hielo o de la lava para conseguir un bloque sólido.
- *Usable*. Esta condición será verdad si el jugador tiene en la posición más alta de su inventario un item que pueda ser usado con el bloque que se encuentra enfrente de él.
- *Walkable*. Verdad si el bloque de enfrente al robot se puede pisar de manera segura (hielo y bloque sólidos).

Por supuesto aparte estructuras de control este lenguaje de programación necesita una forma de representar las acciones que puede llevar a cabo el robot. El robot puede llevar a cabo acciones como avanzar un bloque, usar un item, saltar, girar a la izquierda y girar a la derecha. La representación gráfica de esto es un tramo de carretera recto con un botón (o hasta tres) el cual tiene un icono indicando la acción que se pretende que lleve a cabo el robot.

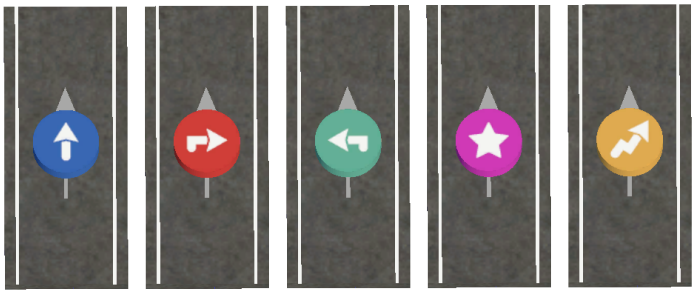


Figura 5.11: Avanzar, girar a la derecha, a la izquierda, usar item y saltar

5.5.2. Posicionamiento de las carreteras

Antes de hablar del cómo se establece cada carretera en su posición adecuada es necesario conocer de qué está compuesta pues esto facilitará enormemente la comprensión de este subsistema.

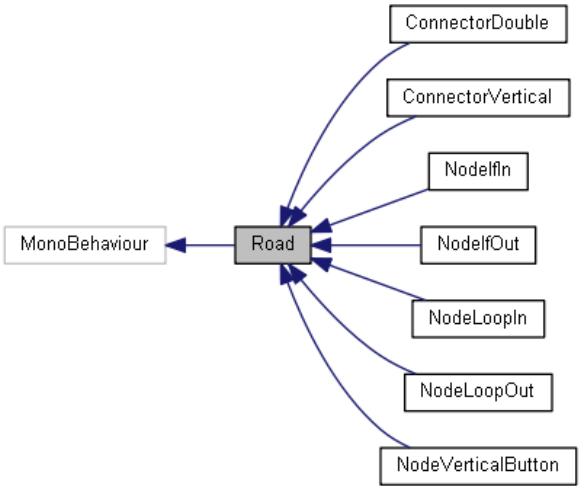


Figura 5.12: Diagrama de herencias de Road

Todos los tramos descienden de la clase Road (ver Anexo C 5.102) que contiene los métodos comunes a estos. De especial interés es el procedimiento ExecuteAction que toma como entrada una lista de cadenas de texto y en el que se puede implementar funcionalidad concreta de cada tramo a ejecutarse sin necesidad de métodos específicos. Cada tramo de carretera está compuesto de al menos tres cosas: inputs, outputs y caminos.

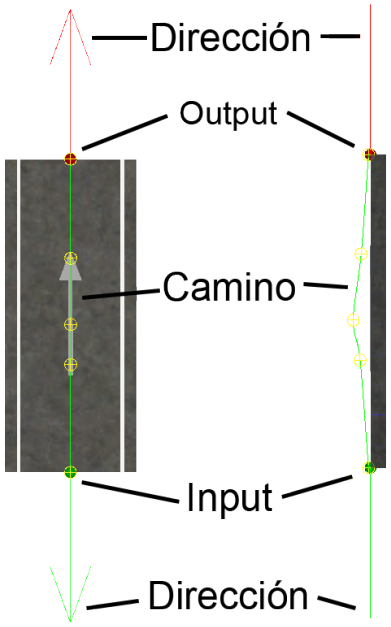


Figura 5.13: Componentes de las carreteras

Inputs y outputs

A la hora de colocar un tramo de carretera es importante saber en qué punto (o puntos) debe acoplarse con el resto de tramos. Para modelar esto se han añadido a los tramos de carretera una serie de inputs y outputs (descendientes de la clase RoadIO (ver Anexo C 5.106)) colocados de forma precisa que representan las entradas y las salidas a la carretera y además se pueden conectar entre opuestos (inputs con outputs y outputs con inputs).

Los IO no solo tienen información sobre posición sino también sobre dirección (la cual puede ser definida de forma informal como la normal al plano imaginario del borde de la carretera en la que se encuentra esa IO) pues se podrían dar casos en los que si bien es posible acoplar un input con un output esto no sea correcto y gracias a tener en cuenta esto es posible evitar esos errores. En el sistema se han definido cuatro direcciones *(forward, backward, left y right)* aunque se podrían añadir más si fuera necesario.

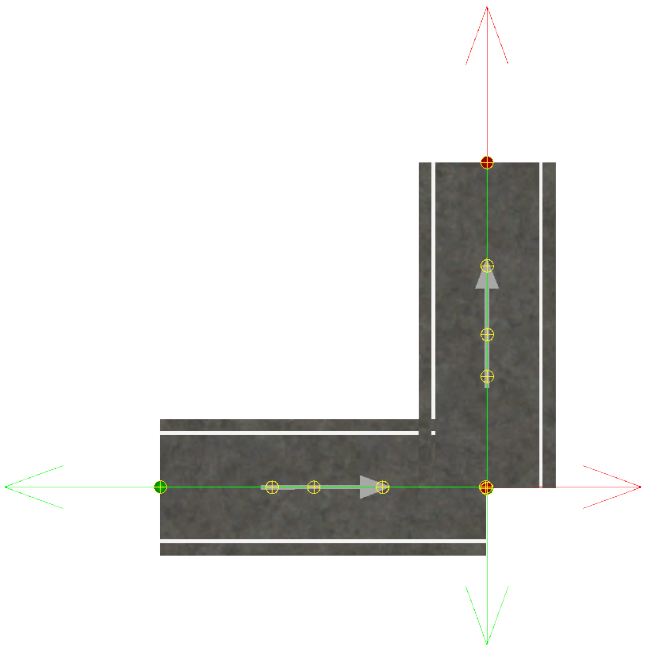


Figura 5.14: Problema por no tener en cuenta la dirección

Caminos

Un tramo de carretera puede contener uno o varios caminos (implementados en la clase PathContainer (ver Anexo C 5.95)) por los que el robot podrá moverse. Cada camino está compuesto por un nombre para poder hacer referencia a él, un input (punto inicial del camino), un output (punto final) y uno o más puntos intermedios entre estos.

Funciones, conectores y huecos

En el sistema se pueden encontrar dos tipos de tramos de carretera: de función y conectores. Las carreteras de función representan algo concreto (una estructura de bucle o *if*, una instrucción (o hasta tres) a ejecutar por el robot, etc) mientras que los conectores se usan a la hora de rellenar los posibles huecos que surjan.

Para que dos tramos de carretera sean compatibles entre sí sólo hay una condición: que cada input seleccionado de un tramo de carretera se pueda conectar con un output del otro tramo (y viceversa).

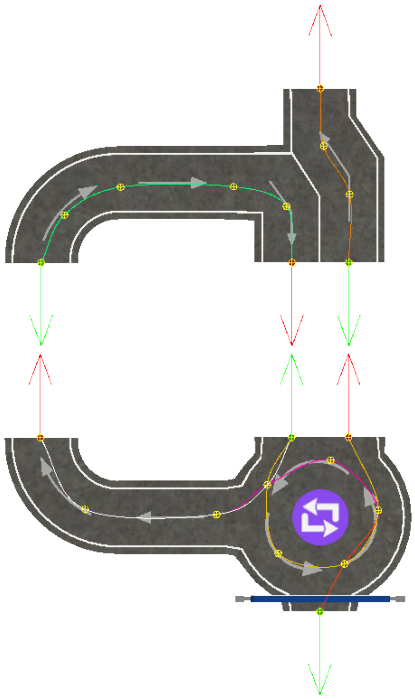


Figura 5.15: Hueco entre dos tramos de carretera

A continuación se muestra una versión simplificada del algoritmo para comprobar si dos carreteras se pueden conectar entre sí. La ventaja de esta solución es que elimina la necesidad de indicar manualmente qué tramos se deben conectar con qué tramos lo que es un trabajo tedioso y propenso a errores. Este algoritmo se encuentra implementado a través del método FillGapWithConnector de la clase RoadFactory (ver Anexo C 5.104).

1. El algoritmo tiene como entrada la lista de IO seleccionada de la carretera A y la carretera B.
2. Se toma como "pivote" la primera IO que se encuentre en la lista de la carretera A (aunque realmente podría usarse cualquiera).
3. Para cada IO seleccionada de la carretera B:
   - 3.1 Se "transporta" la carretera B a la posición del pivote usando como centro la IO de B seleccionada en esta iteración del bucle.
   - 3.2 Se comprueba si todas las IO seleccionadas de A y de B están satisfechas (cada input tiene un output lo suficientemente cerca).
   - 3.3 Si las condiciones se cumplen se sale del bucle y se determina que las carreteras son compatibles, si no se cumplen se pasa a la siguiente iteración.

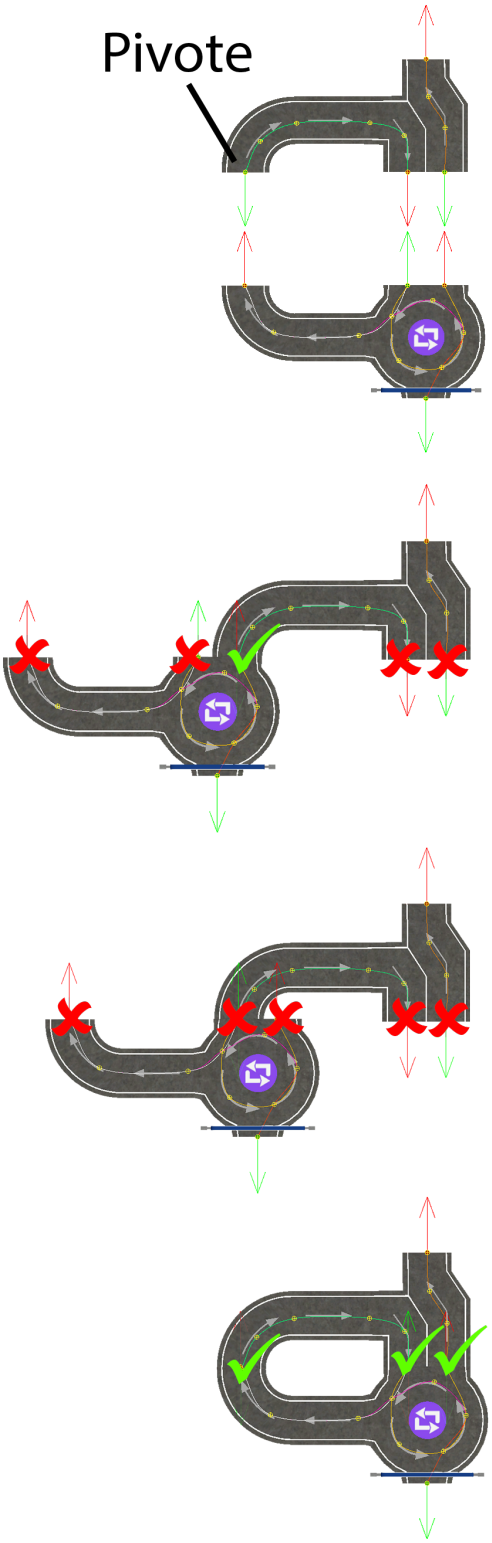


Figura 5.16: Proceso para saber si dos tramos son compatibles

Al introducir tramos de carretera entre dos existentes se pueden generar huecos para lo cual se ha creado la figura del conector, que es un tramo de carretera cuya única función es rellenar estos espacios.

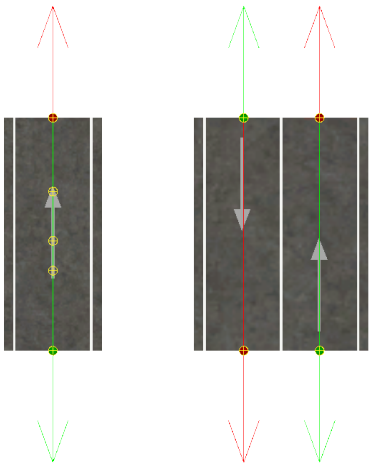


Figura 5.17: Algunos de los conectores usados en RoboTIC

Podemos aprovechar el algoritmo anterior para elegir automáticamente un conector que encaje en el hueco surgido entre dos tramos de carreteras, por ejemplo al introducir una instrucción dentro de un bucle:

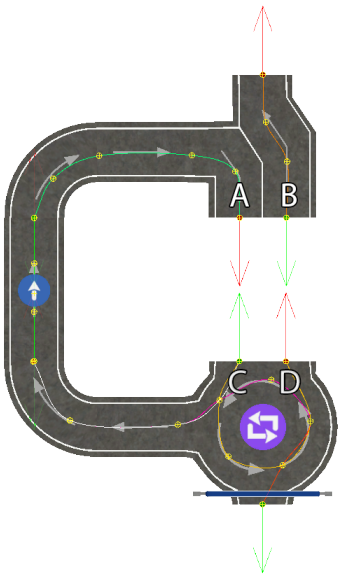


Figura 5.18: Hueco al introducir una instrucción en un bucle

Las IO *A, B, C y D* se han quedado libres pero valdría con buscar entre todos los conectores los que puedan satisfacer *(A, B)* y *(C, D)* y elegir cualquiera que se encuentre en ambos conjuntos.

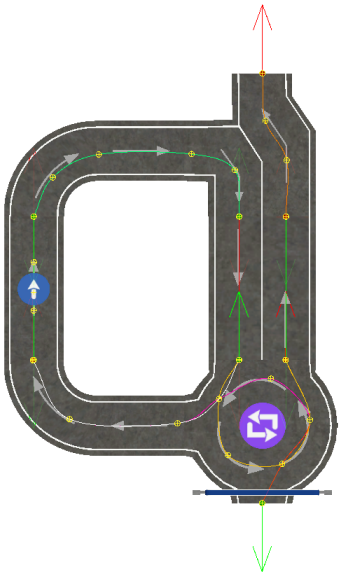


Figura 5.19: Hueco cerrado usando un conector

5.5.3. Formación de las carreteras

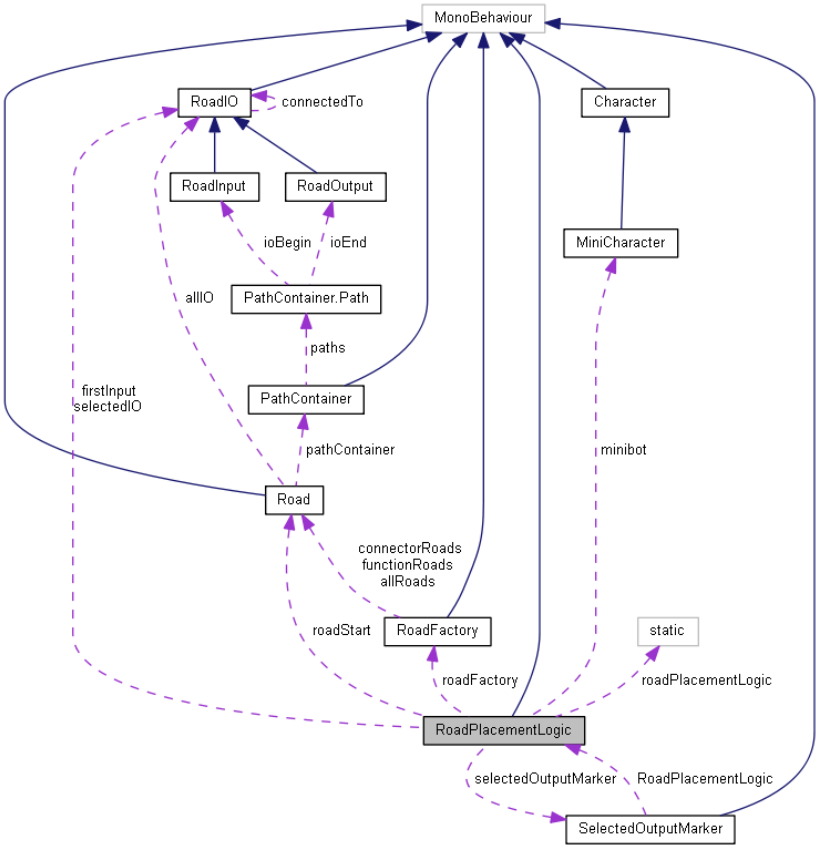


Figura 5.20: Diagrama del sistema de formación de carreteras

La colocación de los tramos de carretera ha sido un tema problemático a lo largo del proyecto y se ha rehecho varias veces hasta obtener una solución lo más óptima y escalable posible. Al principio se optó por un enfoque más estático en el que cada carretera tenía información sobre con qué carreteras podía colocarse y como hacerlo lo que acabó resultando algo extremadamente tedioso, poco elegante y con escalabilidad nula mientras que el sistema finalizado incorpora una serie de algoritmos que hacen fácil añadir nuevas carreteras.

Formar una carretera no es un proceso trivial pues sobretodo dentro de estructuras de control como los bucles se pueden generar una serie de huecos que tendrán que ser rellenados con un número indefinido de conectores.

En el diagrama anterior se muestra el sistema de formación de carreteras en su totalidad. Ya se ha hablado de la clase RoadFactory que es la responsable de decidir la mejor forma de cerrar un hueco entre tramos pero es la clase RoadPlacementLogic la que recibe los comandos del usuario y los convierte en los tramos de carretera apropiados a través del procedimiento SpawnRoads. El algoritmo usado es el siguiente y además de colocar las carreteras a generar en su sitio solventa todos los huecos que hayan podido surgir en el proceso:

1. El algoritmo tiene como entradas los identificadores de las carreteras a generar y la dirección en la que apunta el nuevo tramo de carretera (la dirección del output donde se conecta, generalmente).
2. Se generan los tramos de carretera en el orden en el que aparecen en la lista, se conectan entre sí y se colocan en su lugar correspondiente.
3. El número de tramos que habrá que poner en un hueco será el número de piezas que se han generado en el paso anterior puesto que por simplicidad se ha decidido que en el sistema todas las piezas tengan la misma longitud.
4. Se calcula la dirección contraria a la del nuevo tramo de carretera.
5. Se añaden los IO del último tramo de carretera generado que estén conectados a otras carreteras a una pila.
6. Se le asigna un 0 a este último tramo de carretera.
7. Mientras que el tamaño de la pila sea mayor que 0:
   - 7.1 Se toma la cabeza de la pila y se añade a una lista de IO procesadas. La llamaremos currentIO.
   - 7.2 Si currentIO está conectada a algo:
     - 7.2.1 Se toma el número que tiene asignado la carretera padre de esta IO. Para mayor claridad llamaremos a esta variable nextRoadLevel.
     - 7.2.2 Si la dirección de esta IO es la misma que la del tramo de carretera generado, nextRoadLevel++.
     - 7.2.3 Si la dirección de esta IO es la contraria que la del tramo de carretera generado, nextRoadLevel−.
     - 7.2.4 Si es cualquier otra dirección nextRoadLevel no se altera.
     - 7.2.5 Se toma la IO a la que está conectada, la llamaremos nextIO.
     - 7.2.6 Si nextRoadLevel<=0 estamos ante un posible hueco:
       - 7.2.6.1 Si la distancia del hueco es mayor que cierto margen de error se llena el hueco con los tramos de carretera necesarios teniendo en cuenta que cumplan las condiciones para encajar.
     - 7.2.7 Si nextRoadLevel>0:
       - 7.2.7.1 Si la carretera padre de nextIO no tiene número asignado se le asigna nextRoadLevel.
       - 7.2.7.2 Si la carretera padre de nextIO no está en la lista de carreteras procesadas se mueve a la posición de currentIO tomando como centro nextIO.
       - 7.2.7.3 Se añade toda la IO de la carretera padre de nextIO que esté conectada a otra carretera a la pila.

A continuación se presenta una versión simplificada del algoritmo con imágenes para mayor claridad:

1- Se selecciona el IO donde se desea poner el tramo de carretera.

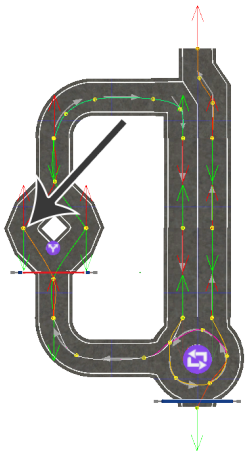


Figura 5.21: Selección de una IO

2- Se genera la nueva pieza si es posible y se conecta a los IO de las carreteras que ha separado.

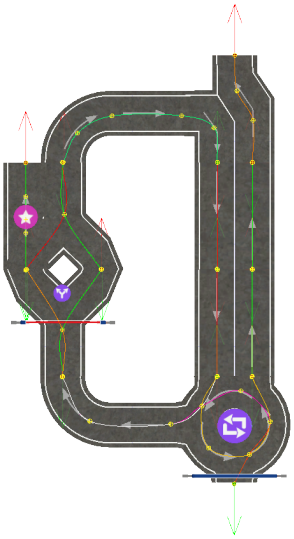


Figura 5.22: Colocación de la nueva pieza

3- Se le asigna al nuevo tramo un número, concretamente un 0 y a las carreteras conectadas a ésta un número que se ve decrementado si la dirección es contraria a la de la nueva pieza y aumentado si es la misma dirección. Además tenemos en cuenta mover cada tramo conectado a su posición final.

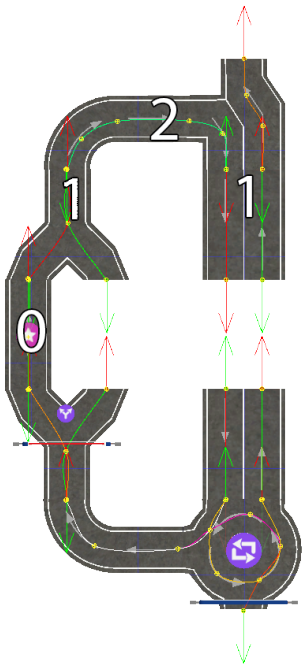


Figura 5.23: Proceso de búsqueda de huecos

4- A las carreteras cuyo número vaya a ser menor o igual que cero no se les llega a asignar el número ni se añade su IO a la pila pero tomamos este dato para saber que hemos encontrado un hueco y debe ser rellenado con conectores que encajen.

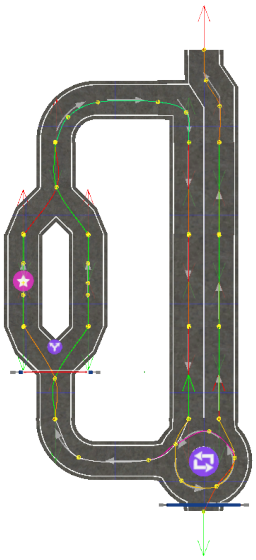


Figura 5.24: Carretera completada

Poner carreteras no lo es todo: también hay que poder quitarlas. Una vez acabado el sistema de colocar carreteras se comprobó que era extremadamente incomodo formar un algoritmo más o menos complejo y tener que reiniciar la partida entera tras cometer un fallo. RoadPlacementLogic contiene una pila que almacena qué se ha modificado en el sistema de carreteras tras cada acción del jugador en forma de instancias de la clase RoadChanges.

Listado 5.11: Clase RoadChanges

```
private class RoadChanges
{
    /// Carreteras añadidas en esta acción.
    public List<Road> addedRoads = new List<Road>();

    /// Conexiones cambiadas en esta acción.
    public Dictionary<RoadIO, RoadIO> connectionsChanged = new ↩
        ↪ Dictionary<RoadIO, RoadIO>();

    /// Botones añadidos en esta acción.
    public Tuple<NodeVerticalButton, VerticalButton> ↩
        ↪ addedButton = null;

    /// IO seleccionada antes de esta acción.
    public RoadIO selectedIOBack = null;
}
```

Si el jugador decide deshacer una acción el proceso que se llevará a cabo es eliminar las carreteras añadidas, revertir las conexiones al estado anterior, borrar los botones que se hayan añadido y finalmente mover la flecha de selección de IO al input u output en el que se encontraba para finalmente corregir las posiciones de los tramos de carretera.

5.5.4. Movimiento en las carreteras

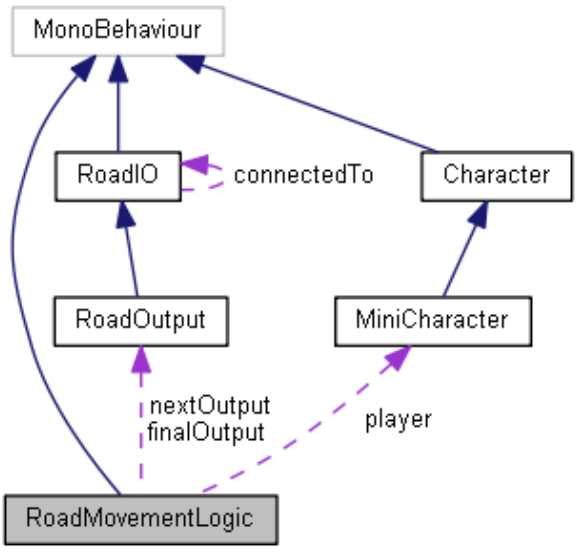


Figura 5.25: Sistema de movimiento en las carreteras

El flujo de ejecución de los algoritmos construidos con los tramos de carretera se representa mediante la posición de un personaje que se mueve sobre estos. Anteriormente se ha explicado que cada tramo contiene caminos que no son más que conjuntos de puntos que empiezan en la entrada (input) de una pieza de carretera y acaban en una salida (output) de la misma.

Para determinar qué camino de entre todos los posibles debe tomar el robot se necesitará conocer el input por el que el mismo quiere acceder al tramo actual de carretera y, en algunos casos, el estado interno del tramo.

La mayoría de las piezas de carretera de RoboTIC no necesitan estado interno y con el input es suficiente para determinar el camino que tomará el robot, sin embargo los bucles y los condicionales necesitan realizar algunos cálculos más, por ejemplo si el número de iteraciones es mayor que cero o si la carta de condición seleccionada está satisfecha. En este ejemplo tomado de la entrada a una estructura condicional se puede ver el procedimiento que se lleva a cabo para elegir por qué rama avanzará el robot:

Listado 5.12: Decisión de camino a la entrada de un condicional

```
public override bool GetPathAndOutput(in RoadInput input,
    out Path path, out RoadOutput output)
{
    if (input == inputIf)
    {
        if (IsConditionMet())
        {
            GetPathByName("Yes", out path);
        }
        else
        {
            GetPathByName("No", out path);
        }
        output = path.ioEnd;
        return true;
    }

    path = new Path();
    output = null;
    return false;
}
```

Una vez solucionado el problema de elegir el camino por el qué se moverá el personaje hay que determinar cómo se moverá por el mismo. Los conjuntos de puntos con los que cuentan los tramos de carreteras son limitados (uno a diez puntos por camino aproximadamente) e intentar que el robot se moviera directamente de un punto a otro generaría una animación muy deficiente por lo que se recurre a la interpolación (o *tweening*) para subsanar esto. Lo bueno de esta forma de guardar los caminos con unos pocos puntos e interpolando entre ellos es que se pueden añadir nuevos o modificar los existentes de una manera muy sencilla.

Se ha decidido utilizar la biblioteca LeanTween para realizar este proceso por su facilidad de uso y sus buenos resultados. En un primer momento se intentó usar iTween pero daba problemas al acelerar y frenar el movimiento del personaje dependiendo de la distancia a la que se encontraran los puntos de entrada de la interpolación (aún así iTween[2] continúa en uso en el sistema final para dibujar las vistas preliminares de los caminos en el editor de niveles).

La forma de recorrer una carretera es responsabilidad de la clase RoadMovementLogic (ver Anexo C 5.107) la cual se encarga de seleccionar qué camino deberá seguir a cabo el robot y poner en funcionamiento el proceso de interpolado que finalmente causará el movimiento del mismo.

[2] http://www.pixelplacement.com/itween/index.php

Listado 5.13: Método Update() de la clase RoadMovementLogic.cs

```
private void Update()
{
    //Primero se comprueba si el movimiento ha empezado para
    //no realizar cálculos cuando no es necesario.
    if (movementStarted)
    {
        //Si se ha acabado el proceso de interpolado en el camino anterior
        //hay que decidir si se ha acabado la carretera o no
        if (!LeanTween.isTweening(tweenDescr.id))
        {
            //Se acaba la carretera cuando el próximo output sea el output
            //final de la misma
            if (nextOutput == finalOutput)
            {
                Debug.Log("End of the road!!");
                movementStarted = false;
                GameLogic.Instance.FinishedMinibotMovement = true;
            }
            else
            {
                //Si no se ha acabado la carretera se debe pedir
                //más camino tomando el input al que está conectado
                //el ultimo output
                if (!StartNewPath((RoadInput) ↩
                    ↪ nextOutput.ConnectedTo, out tweenDescr))
                {
                    movementStarted = false;
                    Debug.LogError("No path available");
                    GameLogic.Instance.FinishedMinibotMovement ↩
                        ↪ = true;
                }
            }
        }
    }
}
```

Si bien se ha conseguido construir un sistema para el movimiento en las carreteras estable no siempre fue así. En el proceso del desarrollo de este Trabajo de Fin de Grado primero se experimentó con *pathfinding* para permitir al robot moverse dentro de la carretera pero esto resultó extremadamente complicado y propenso a fallos pues el personaje exhibía un movimiento impredecible que debía corregirse con el uso artificial de obstáculos invisibles para evitar vueltas atrás y cruces entre carriles de los tramos de carretera.

5.6. COMUNICACIÓN ENTRE OBJETOS

Uno de los muchos problemas que han surgido a lo largo de estos meses es el cómo comunicar unas clases con otras. Al principio se optó por usar siempre referencias pero esto demostró ser difícil de mantener según se iba añadiendo nueva funcionalidad.

Después se intentó el enfoque completamente opuesto de llevar a cabo un sistema de paso de mensajes y que toda la comunicación fuera a través de él pero se convirtió en algo tedioso de hacer (pues ya había partes del proyecto funcionando) y complicaba la sincronización de algunas partes

de la aplicación que es crítico que interactúen de forma síncrona (por ejemplo, gran parte de los subsistemas lógicos).

Al final lo más efectivo ha sido usar un enfoque mixto entre paso por referencia y mensajes según se determine más apropiado para la parte de la aplicación en cuestión.

5.6.1. Event Aggregator

El agregador de eventos (o *event aggregator* en inglés) es la parte que centraliza el envió de mensajes dentro de RoboTIC. Las clases que conforman este módulo pueden encontrarse en la carpeta Assets/Scripts/MessageHub y las más importantes son EventAggregator (ver Anexo C 5.27) y MessageWarehouse (ver Anexo C 5.54).

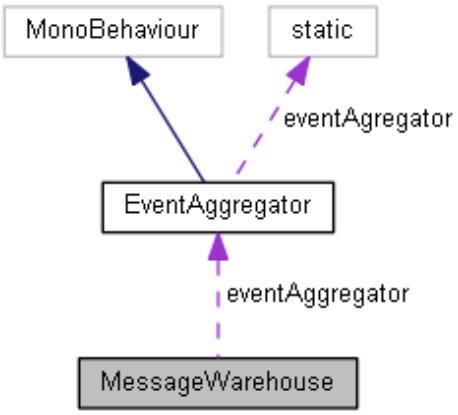


Figura 5.26: Diagrama del agregador de eventos

EventAggregator

Los objetos de tipo EventAggregator actúan como un punto central en el que los demás objetos del juego pueden suscribirse y publicar mensajes del tipo que deseen. Actualmente el sistema cuenta con 37 mensajes diferentes para múltiples propósitos (cargar el mapa, pedir ciertos datos, mandar acciones al robot protagonista, etc).

MessageWarehouse

En ocasiones es importante garantizar cierta sincronía en el envío y la recepción de los mensajes para evitar que ciertas acciones se ejecuten antes de tiempo. Para solucionar este problema se ha desarrollado MessageWarehouse que sirve como un almacén de mensajes guardando los que se requieran hasta que se necesite la respuesta lo que ha demostrado ser de gran utilidad a la hora de comunicar la lógica con los gráficos. A continuación se muestra un pequeño ejemplo de uso tomado de la clase GameLogic:

Listado 5.14: Ejemplo de uso de MessageWarehouse

```
// Con el objeto msgWar instancia de la clase
// MessageWarehouse.cs podemos publicar un mensaje
// y esperar hasta que la respuesta esté lista.
MsgBigCharacterAllActionsFinished msg = new ↩
    ↪ MsgBigCharacterAllActionsFinished();
msgWar.PublishMsgAndWaitForResponse ↩
    ↪ <MsgBigCharacterAllActionsFinished, bool>(msg);
bool outTmp;
yield return new WaitUntil(() => msgWar.IsResponseReceived ↩
    ↪ <MsgBigCharacterAllActionsFinished, bool>(msg, out outTmp));
// Evitando así que la pantalla que pregunta
// al jugador si quiere reintentar el nivel
// se muestre antes de tiempo
```

5.7. INTERACCIÓN CON EL ENTORNO

Un requisito imprescindible y que ha condicionado el desarrollo del proyecto es que se debe implementar una interfaz natural de usuario que use las *HoloLens* para cualquier interacción con el jugador. El primer problema a solucionar es cómo procesar el espacio que rodea al usuario pues a diferencia de en un videojuego convencional en RoboTIC el escenario no está predefinido al tomar como base el mundo real, seguido de cómo se interactuará con los objetos que componen el juego. En esta sección además de eso se tratará también el sistema de sonido implementado.

5.7.1. Sistema de sonido

El sistema de sonido de RoboTIC que se puede ver en la clase SoundEngine (ver Anexo C 5.113) se ha implementado con vistas a su ampliación en un futuro. Dicha clase recibe mensajes a través de EventAggregator con los *clips* a reproducir. En la actualidad soporta la reproducción de sonidos con y sin origen concreto pero se ha preferido centralizar el sonido en un objeto en específico pues de ser necesario se podrían encolar los *clips* y efectuar procesamientos como por ejemplo evitar que se reproduzcan demasiadas instancias del mismo sonido a la vez o evitar algunos concretos sin tener que modificar el código del resto del sistema.

Listado 5.15: Clase SoundEngine.cs

```
using UnityEngine;

[RequireComponent(typeof(AudioSource))]
public class SoundEngine : MonoBehaviour
{
    private AudioSource audioSource;

    private void Awake()
    {
        audioSource = GetComponent<AudioSource>();
        EventAggregator.Instance.Subscribe<MsgPlaySfx>(PlaySfx);
        EventAggregator.Instance.Subscribe<MsgPlaySfxAtPoint> ↩
            ↪ (PlaySfxAtPoint);
    }

    private void PlaySfx(MsgPlaySfx msg)
    {
        audioSource.PlayOneShot(msg.clip, msg.volume);
    }

    private void PlaySfxAtPoint(MsgPlaySfxAtPoint msg)
    {
        AudioSource.PlayClipAtPoint(msg.clip, msg.point, ↩
            ↪ msg.volume);
    }
}
```

5.7.2. Colocación de los objetos en el espacio

El proceso para capturar el espacio es ciertamente complejo y por ello se ha integrado en la solución código perteneciente al tutorial MR Spatial 230: Spatial mapping[3] para mantener el tiempo de desarrollo dentro de unos margenes aceptables. La forma en la que se lleva a cabo este proceso a grandes rasgos consiste en escanear el espacio cercano al usuario usando los sensores que se encuentran en las HoloLens generando una malla de triángulos de la habitación donde se encuentra el jugador. Una vez que está disponible en el sistema este *mesh* en bruto hay que identificar planos para finalmente poder colocar los elementos del juego en el plano más cercano al jugador donde haya espacio suficiente. En los párrafos siguientes se explicará el proceso de forma más detallada.

La generación del *mesh* es asíncrona y empieza con la llamada correspondiente a la clase SurfaceObserver proporcionada por Unity que se encuentra en el ensamblado *UnityEngine.VRModule* desde SpatialMappingObserver (ver Anexo C 5.116). Este procedimiento es costoso de llevar a cabo dependiendo de factores como los triángulos por metro cúbico y puede durar un número indeterminado de fotogramas pero una vez sea completado habrá disponible una representación del espacio en el que actualmente se encuentra el jugador. En la siguiente imagen se puede ver el modelo obtenido tras observar la habitación del autor de este documento. A modo de curiosidad se puede apreciar que las *HoloLens* son capaces de observar tras puertas de cristal.

[3] https://docs.microsoft.com/en-us/windows/mixed-reality/holograms-230

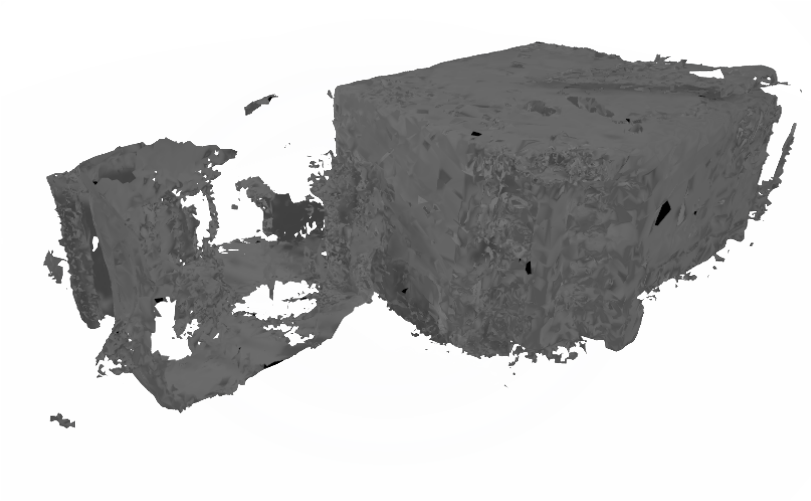


Figura 5.27: Modelo en bruto de una habitación

El resultado del paso anterior no es útil en el estado en el que se encuentra pues es necesario determinar qué partes del modelo son planos para poder ubicar en ellos los objetos del juego. La creación de planos a partir de los datos obtenidos previamente se realiza en una instancia de la clase PlaneFinding (ver Anexo C 5.98) a través de llamadas a la biblioteca de enlace dinámico PlaneFinding.dll. No todos los planos son iguales y por ello a cada uno se le asigna un tipo, por ejemplo pared si es vertical, suelo si es horizontal, techo si es horizontal y su normal apunta hacia abajo, etc, lo cual es de gran utilidad a la hora de colocar los objetos de juego. En la imagen de debajo se muestran los planos que se han calculado tomando como entrada la malla obtenida anteriormente.

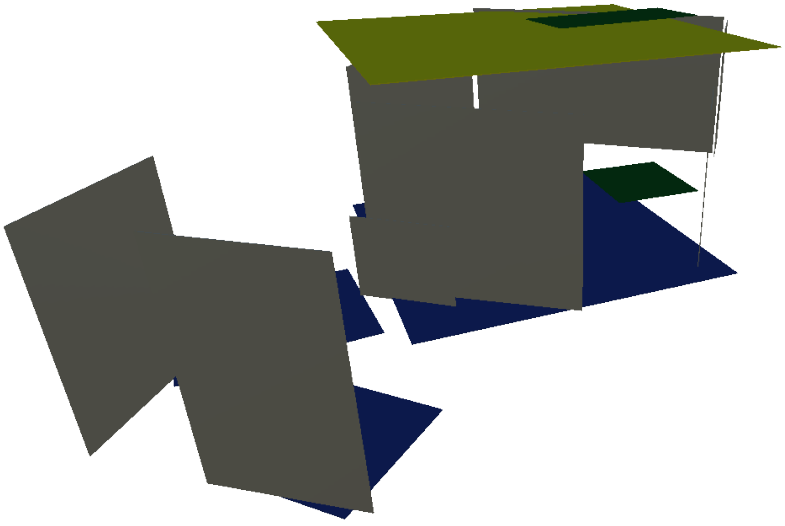


Figura 5.28: Planos obtenidos a partir del modelo anterior

En RoboTIC todo lo que puede ver el jugador se encuentra dentro del objeto PlaceableMap que cuenta con una instancia de la clase Placeable (ver Anexo C 5.97) y tiene afinidad por las superficies horizontales como suelos o mesas. Una vez que el objeto MainMenuLogic detecta que el procesamiento del espacio del jugador ha finalizado llama a SpaceCollectionManager (ver Anexo C 5.114) que se

encargará de colocarlo en una superficie adecuada en la posición más cercana posible a la cabeza del jugador. Decimos que una superficie es adecuada cuando tiene una orientación correcta y espacio suficiente para contener el objeto que se trata de colocar.

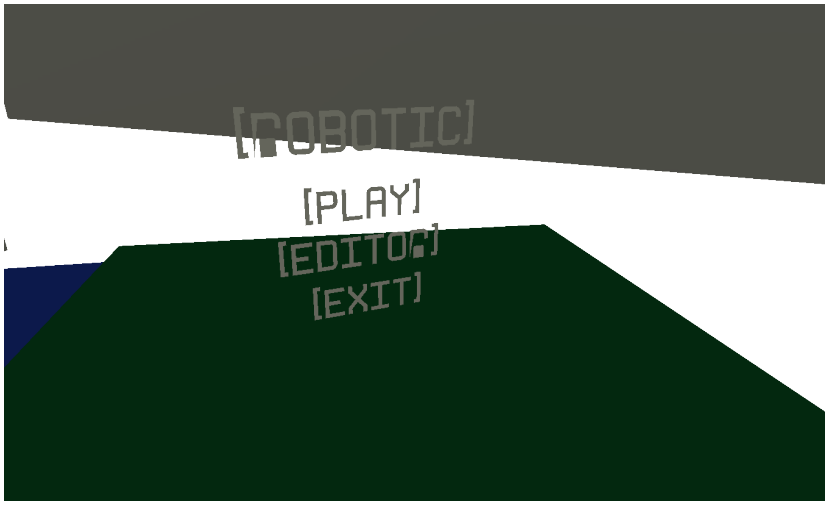


Figura 5.29: Objeto colocado sobre un plano afín

5.7.3. Interacción con los objetos

La interacción con los objetos del juego ha estado muy condicionada desde un principio por los gestos que soporta la plataforma a la que está destinado. A diferencia de en un videojuego tradicional cuyos controles basados en teclado y ratón o mando que pueden generar muchas entradas diferentes al usar las *HoloLens* todo se ve condicionado al gesto de pulsar (*tap*) y la dirección en la que mira el jugador (*gaze*). Si bien en principio puede parecer una limitación dicho conjunto reducido de gestos, a lo largo de la construcción se ha preferido orientarlo como algo sobre lo que generar una experiencia accesible y directa.

El proceso de interacción en RoboTIC comienza una vez que es conocido qué objeto está mirando el usuario. En la clase GazeManager (ver Anexo C 5.31) se encuentra el código que determina en cada fotograma si el jugador está mirando un objeto del juego. Esto funciona mediante el trazado de un rayo tomando como origen la posición de la cámara principal (ubicada en la cabeza del usuario) en dirección hacia delante, Unity proporciona un método para realizar esta función que no solo retorna si el rayo colisiona con un objeto sino que también ofrece información del punto en el que se ha colisionado, el GameObject con el que ha sido, etc.

Dentro del juego el punto hacia el que mira el jugador se representa con un pequeño cursor circular pues sería ciertamente difícil saber exactamente qué se está observando si no hubiera ningún tipo de *feedback*.

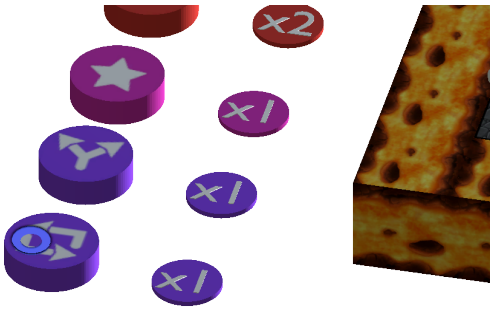


Figura 5.30: Objeto marcado con el cursor

Una vez se sabe qué GameObject está seleccionado es trivial mandarle un mensaje cada vez que el usuario hace el gesto *tap*. Unity implementa el objeto GestureRecognizer que genera un evento cuando detecta un gesto de los soportados que recoge la clase GestureManager (ver Anexo C 5.33).

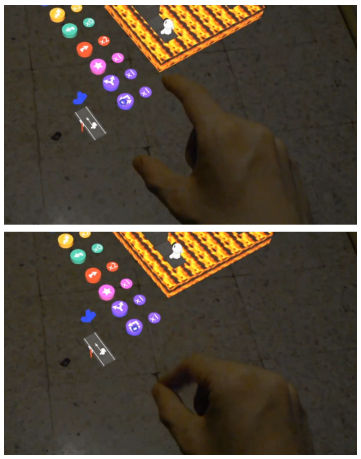


Figura 5.31: Usuario haciendo *tap* sobre un objeto

Para mandar un mensaje al objeto en concreto se utiliza el sistema de paso de mensajes de Unity que toma como argumento el nombre del método al que se pretende llamar en cualquiera de los scripts que forman parte del GameObject de destino. Además, también se pública un mensaje en el EventAggregator exista o no un objeto enfocado pues resulta de gran utilidad durante movimientos en los que se debe reaccionar a la entrada del usuario pese a que no esté apuntando directamente a un objeto en concreto.

Listado 5.16: Momento en el que se generan los mensajes tras recibir un tap

```
1  private void GestureRecognizer_Tapped(TappedEventArgs args)
2  {
3      EventAggregator.Instance.Publish<MsgSomethingTapped>(new ↩
           ↪ MsgSomethingTapped());
4      if (focusedObject != null)
5      {
6          focusedObject.SendMessage("OnSelect");
7      }
8  }
```

CAPÍTULO 6

CONCLUSIONES

En este capitulo se van a tratar las conclusiones a las que se ha llegado durante la realización de este trabajo además de costes del proyecto, estadísticas del repositorio y de rendimiento y la tabla de competencias que se han tratado durante el desarrollo del mismo.

6.1. COSTE ECONÓMICO

RoboTIC se ha desarrollado entre los días 4 de octubre de 2019 y 28 de julio de 2020. Si bien puede parecer un periodo muy largo hay que tener en cuenta que a la vez se han debido llevar a cabo prácticas y otras actividades académicas por lo que la dedicación no ha sido a jornada completa. En total se estiman unas 700 horas de implementación del software acompañadas de 150 de escritura de la memoria. Asumiendo un sueldo de 33,65€ la hora[1] (y no teniendo en cuenta el tiempo que ha llevado realizar la memoria) los costes han sido los siguientes:

Descripción	*Cantidad*	*Coste*
Sueldo desarrollador	1	23.555€
Microsoft Hololens	1	3.631€
Asus K551LN	1	759€
Total		27.945€

Tabla 6.1: Desglose del coste del proyecto

6.2. ESTADÍSTICAS DEL REPOSITORIO

Se han llevado a cabo 130 *commits* en el repositorio y la extensión final del código ha resultado en 10.880 lineas (sin contar lineas en blanco) de las cuales el 36,5 % son comentarios y distribuidas a lo largo de 104 archivos, todos ellos escritos en c#. Para contar las lineas se ha usado *cloc*[2] sobre la carpeta *Assets/Scripts*. En la siguiente tabla se pueden ver los números concretos:

Language	*Files*	*Blank*	*Comment*	*Code*
C#	104	1498	3970	6910
SUM:	104	1498	3970	6910

Tabla 6.2: Estadísticas de número de lineas

[1] https://www.payscale.com/research/US/Job=Unity_Developer/Salary
[2] https://github.com/AlDanial/cloc

El número de lineas del proyecto ha ido variando con el tiempo pero no siempre de forma creciente. A lo largo del tiempo de desarrollo de un programa complejo como es el caso de RoboTIC se pueden ir acumulando librerías y demás *assets* que realmente no son necesarios nunca más. En el gráfico se aprecia que en las ultimas etapas del desarrollo del juego se ha llevado a cabo un limpieza intensiva de todo aquello que se pudiera considerar innecesario. Este diagrama[3] tiene en cuenta todas las lineas del proyecto, incluyendo las bibliotecas y las autogeneradas.

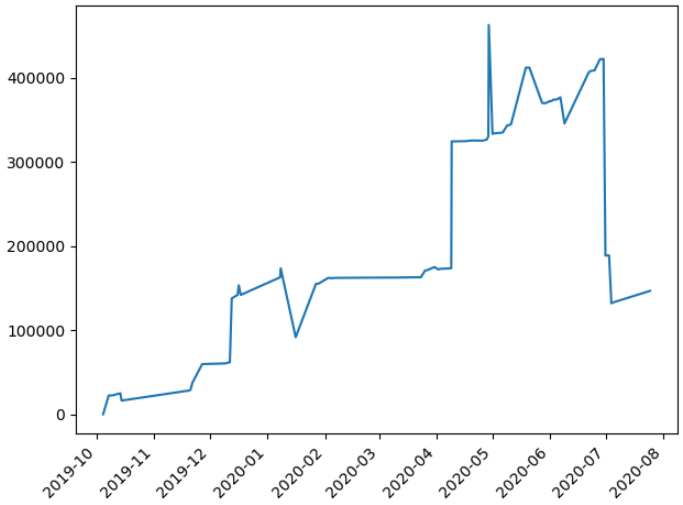


Figura 6.1: Número de lineas en el proyecto

La carga de trabajo ha sido considerable y se ha debido compaginar con el resto de las actividades académicas, lo que en los siguientes diagramas se refleja en los días en los que se han producido los *commits* pues si bien se le ha dedicado algo menos de tiempo al proyecto los fines de semana tampoco han habido una gran diferencia respecto al resto de días de la semana.

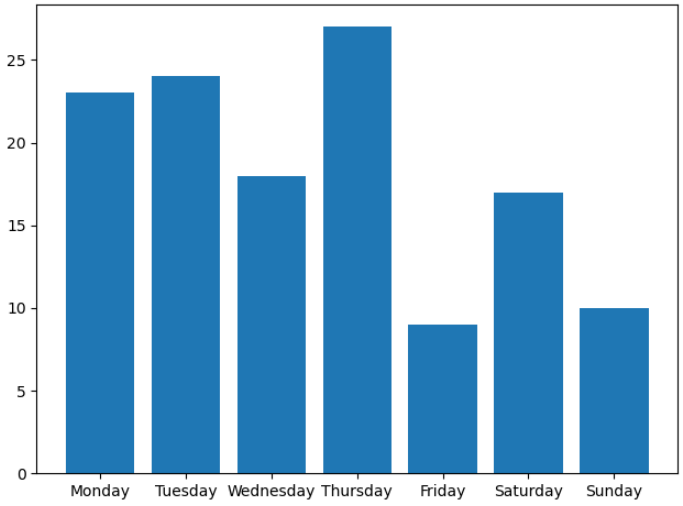


Figura 6.2: Número de commits por día de la semana

[3]Realizado con git-hammer https://github.com/asharov/git-hammer

En las horas de publicación de los commits se observa algo similar, con actividad a cualquier hora del día menos de madrugada.

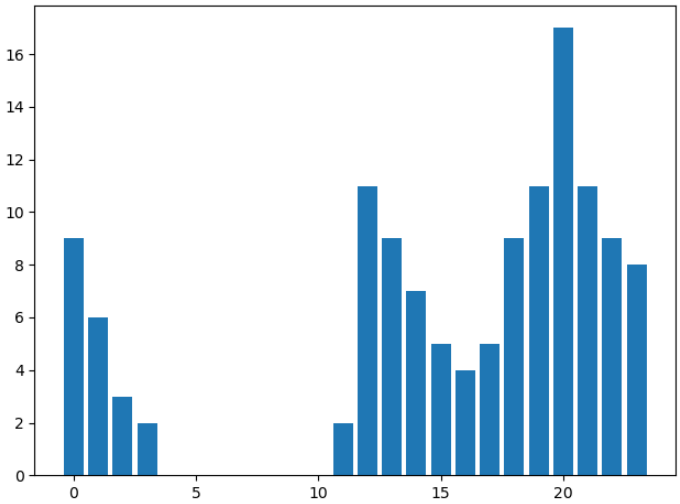


Figura 6.3: Commits según horas del día

6.3. ESTADÍSTICAS DE RENDIMIENTO

En un videojuego como RoboTIC el rendimiento es una preocupación importante pues se han de generar entre 30 y 60 fotogramas cada segundo para que el usuario no pierda la sensación de que el movimiento de los gráficos es fluido y no una sucesión de imágenes estáticas.

El instanciado de objetos en Unity puede ser un problema pues es una operación pesada y al estar los niveles de RoboTIC compuestos numerosos de bloques e items que se deben instanciar de forma dinámica podríamos estar ante un cuello de botella muy relevante.

Para comprobar que los requisitos de rendimiento se cumplan se han llevado a cabo una serie de experimentos consistentes en medir los fotogramas por segundo al cargar ciertos niveles en el sistema, navegar por la pantalla de selección de mapas, empezar a jugar el mapa cuyo número de bloques sea mayor y finalmente reiniciar la partida.

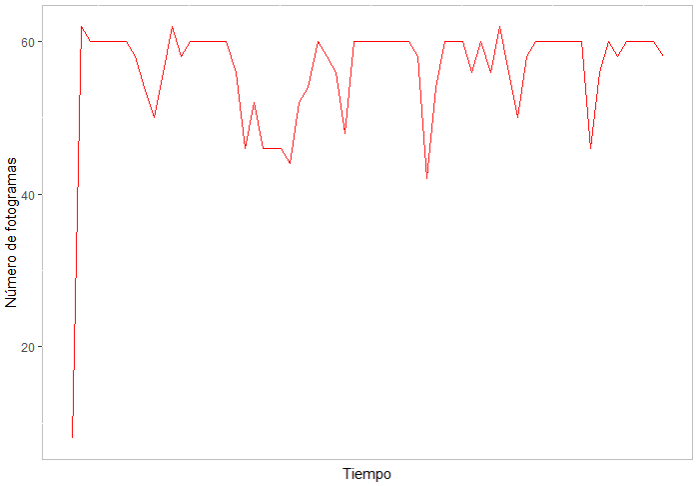


Figura 6.4: Rendimiento con 64 bloques

El primero de estos experimentos cuenta con un único nivel compuesto por 64 bloques. El primer pico descendente que se aprecia en la imagen corresponde con el momento en que la clase SpatialMappingObserver (ver sección 5.7.2) empieza a realizar su función. Este descenso de rendimiento inicial es apreciable en todos los tests que se han realizado. Se pueden ver otras caídas cuando se cargan los niveles y se empieza y reinicia la partida pero siempre sin bajar de los 30 cuadros por segundo.

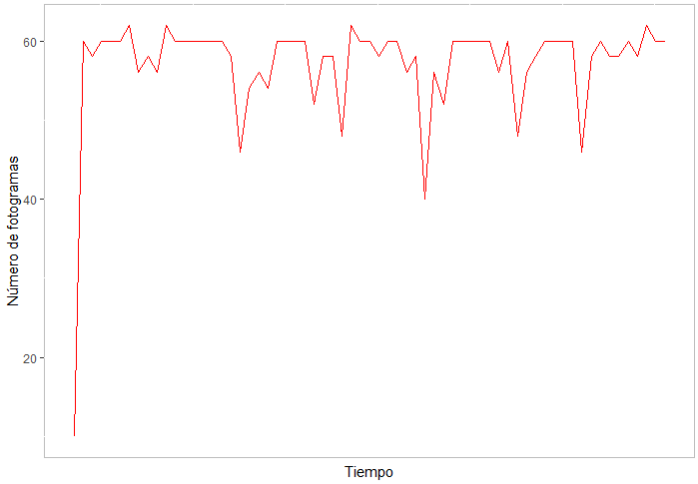


Figura 6.5: Rendimiento con 192 bloques

Para el segundo experimento se ha añadido un nivel de 128 bloques, sumando en total 192 que deberán cargarse en el sistema. El rendimiento es bastante similar.

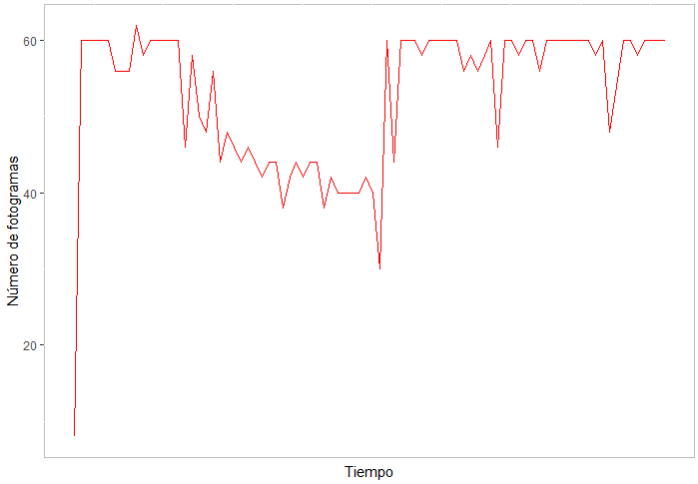


Figura 6.6: Rendimiento con 448 bloques

Ahora se añade un nivel de 256 bloques (tamaño máximo de nivel en RoboTIC). Con 448 bloques se produce un pico de perdida de rendimiento durante la carga de niveles mayor que anteriormente, aún así después de este proceso vuelve a números apropiados.

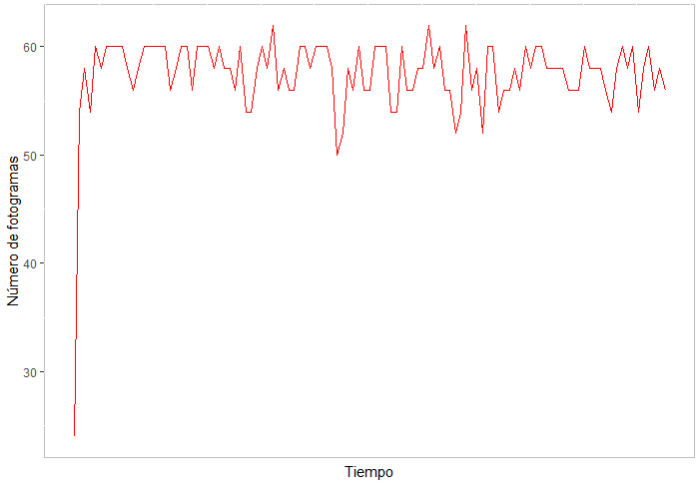


Figura 6.7: Rendimiento con 1728 bloques

Esta vez en lugar de un solo nivel se añaden a los existentes cinco nuevos conteniendo 256 bloques cada uno. Con 1728 bloques cargados el rendimiento es bueno manteniéndose siempre por encima de los 50 fotogramas por segundo. Se puede explicar que el rendimiento sufra de menos caídas con 1728 bloques que con 448 teniendo en cuenta que en un computador multitarea como el *headset* HoloLens pueden existir procesos en segundo plano acaparando recursos del sistema.

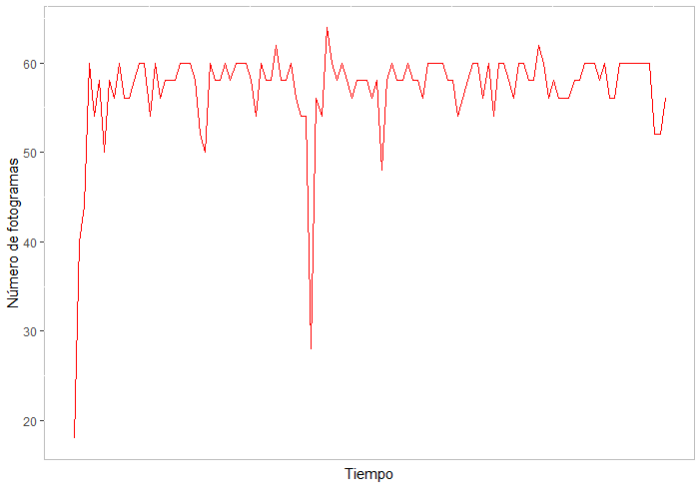


Figura 6.8: Rendimiento con 3776 bloques

En el último de los tests se ha aumentado el número de bloques a cargar a 3776 al introducir en el sistema 8 niveles más de 256 bloques cada uno. A pesar de una breve y visible caída de rendimiento este pronto se estabiliza.

A lo largo de estas pruebas se ha comprobado que RoboTIC puede soportar perfectamente un uso normal del mismo con un rendimiento adecuado pues si bien pueden producirse cierta disminución de fotogramas por segundo durante el escaneo del entorno del usuario y la carga de niveles se estabiliza una vez que ambos procesos han finalizado.

6.4. JUSTIFICACIÓN DE COMPETENCIAS ADQUIRIDAS

En esta sección se incluye una tabla con las competencias de la tecnología específica de computación que se han practicado y adquirido a lo largo de los meses de desarrollo de este trabajo de fin de grado.

Competencia	**Justificación**
[CM2] Capacidad para conocer los fundamentos teóricos de los lenguajes de programación y las técnicas de procesamiento léxico, sintáctico y semántico asociadas, y saber aplicarlas para la creación, diseño y procesamiento de lenguajes.	El objetivo de este proyecto es implementar un entorno virtual con un lenguaje de programación propio basado en metáforas visuales que habrá de ser diseñado y desarrollado para tal efecto.
[CM3] Capacidad para evaluar la complejidad computacional de un problema, conocer estrategias algorítmicas que puedan conducir a su resolución y recomendar, desarrollar e implementar aquella que garantice el mejor rendimiento de acuerdo con los requisitos establecidos.	En un sistema software como un videojuego es imprescindible un rendimiento estable y constante, más cuando partes de este se apoyan en la realidad aumentada, por lo que el conocimiento y desarrollo de estrategias algorítmicas rápidas y optimizadas es imperativo.
[CM6] Capacidad para desarrollar y evaluar sistemas interactivos y de presentación de información compleja y su aplicación a la resolución de problemas de diseño de interacción persona computadora.	Este proyecto es un sistema totalmente interactivo que recibe y transmite información directamente al usuario a través del paradigma de la realidad aumentada.

Tabla 6.3: Justificación de las competencias específicas abordadas en el TFG

6.5. CONCLUSIÓN DEL PROYECTO

Como se trató anteriormente en el capitulo objetivos, el objetivo general de este proyecto es desarrollar el prototipo de un juego serio con la finalidad de enseñar programación a escolares de una forma sencilla y directa usando realidad aumentada lo que ha sido cumplido en su totalidad. El sistema finalizado es completamente funcional en el *headset* Microsoft Hololens e incluye el editor de niveles del que también se hacia mención.

Entre los objetivos específicos del trabajo se encuentra el requisito de que el sistema debe ser fácil de usar al estar orientado a personas de corta edad sin experiencia previa en programación y además debiera tener un paradigma de interacción natural. Ambas cosas se han conseguido de forma interrelacionada gracias al uso de la realidad aumentada que permite al usuario relacionarse con el programa de una forma orgánica integrando las imágenes del videojuego en el mundo real y usando un solo tipo de gesto como forma de entrada.

Otro de los objetivos específicos requería que el proyecto fuera escalable y estuviera extensivamente documentado. Para solventar lo primero se ha optado por usar siempre que fuera posible un sistema de paso de mensajes que evitara interdependencias siempre que fuera posible y se ha organizado de una forma modular. La documentación extensiva también se ha llevado a cabo usando *Doxygen* y se han escrito descripciones de la mayoría de componentes del programa.

La posibilidad de ser portado a otras plataformas también ha sido un tema de preocupación a lo largo de estos meses y por ello se ha desarrollado sobre el motor Unity pues permite la exportación a múltiples dispositivos y sistemas operativos.

6.6. CONCLUSIÓN PERSONAL

Si bien los objetivos generales y específicos que se expusieron en la sección de objetivos han sido cumplidos de forma satisfactoria, a lo largo del proyecto surgieron ideas interesantes que por desgracia no han podido ser implementadas en la versión final. Entre ellas me gustaría citar las siguientes:

- Una versión de escritorio para que RoboTIC pudiera llegar a instituciones educativas y usuarios que no dispongan de un dispositivo HoloLens. Unity posee una característica[4] que permite compilar y ejecutar diferentes porciones de código fuente según la plataforma de destino lo que sería extremadamente útil para llevar a cabo esta idea en unos dos o tres meses.
- Manera visual de representar procedimientos. Esto podría hacerse en alrededor de un mes usando piezas que representen una llamada a un conjunto de carreteras permitiendo así crear "niveles dentro de niveles" lo que resultaría muy beneficioso para representar conceptos de la programación como el encapsulamiento.
- Un sistema multijugador sería especialmente acertado para un proyecto de este tipo pues abriría una serie de posibilidades enormes de cara a motivar a los usuarios, ya sea de forma competitiva o cooperativa. Unity incluye formas[5] de crear entornos multijugador lo que lo hace factible si bien sería algo más complejo de llevar a cabo se debería tener funcionando en unos tres meses.

Estas ideas son una pequeña muestra de objetivos que no han podido llegar al proyecto por falta de tiempo.

El trabajo del ingeniero informático, y de cualquier ingeniero, tiene que ver mucho con el tiempo. Si bien se enseña durante la carrera que “con tiempo y recursos infinitos cualquier proyecto es posible” nunca había entendido tan profundamente la importancia de esa frase hasta que he tenido que enfrentarme en solitario a un trabajo de la magnitud de este.

Llegué completamente verde al mundo de la programación de videojuegos y desde luego estos meses no han sido un camino de rosas debido a la cantidad de información que he tenido que asimilar sobre este campo, todos los problemas surgidos, exámenes, prácticas, la presión constante del paso del tiempo, etc, pero ahora que lo he acabado siento que estoy capacitado para ser ingeniero pues ya no siento todo como un conjunto de apuntes colgados en *moodle*: ahora es de verdad.

Estoy convencido de que este proyecto me ha aportado una base práctica conveniente que me ayudará a realizar mi profesión de forma satisfactoria en un futuro y además espero que sirva en la medida de lo posible para abrir las puertas a otras personas al mundo de la programación que tan importante es hoy en día.

[4] https://docs.unity3d.com/Manual/PlatformDependentCompilation.html
[5] https://docs.unity3d.com/Manual/UNet.html

ANEXOS

ANEXO A

PRUEBAS CON USUARIOS

En el capítulo de introducción se explicó que RoboTIC es una evolución del proyecto *"RoboTIC: a serious game based on augmented reality for learning programming"*[14] durante cuyo desarrollo se llevó a cabo una prueba con usuarios reales por lo que se ha considerado oportuno incluir, a modo de anexo, un resumen de la metodología y resultados de estas pruebas, así como las páginas del artículo donde se encuentra esta información en detalle por si se desea verificar. No obstante, realizar experimentos del mismo tipo con el proyecto que trata este TFG está considerado como trabajo futuro.

La prueba contó con la participación de 12 escolares (9 chicos y 3 chicas) cuya edad media se encontraba en el momento del experimento alrededor de los 12 años de edad que fueron divididos en 3 grupos iguales con el fin de evitar distracciones. Se realizaron dos encuestas a los participantes: una antes de probar el sistema (*pre-test*) y otra después (*post-test*).

La encuesta previa al uso del juego estaba compuesta de preguntas sobre información demográfica además de cuestiones generales sobre el conocimiento y el interés del participante en la programación y su opinión en temas relacionados con los videojuegos y los computadores.

El análisis de la primera encuesta arrojó que únicamente el 12,5 % de los usuarios había oído hablar de la programación de ordenadores o sabía para qué sirve y ninguno de ellos había programado previamente. A pesar de estos resultados el 33,33 % de los participantes manifestaba interés en dedicarse en un futuro a la creación de videojuegos u otros trabajos relacionados con la computación.

Durante la fase de uso del sistema los participantes debieron resolver, sin limite de tiempo, un nivel de RoboTIC que incluía todos los elementos relevantes del juego: instrucciones para definir el comportamiento del robot y condicionales y bucles para crear el flujo de ejecución del algoritmo. El problema al que se tuvieron que enfrentar los participantes debía resolverse mediante una combinación de los anteriores elementos.

Después de usar el sistema se realizó otra encuesta en la que fueron repetidas algunas cuestiones sobre programación de la primera para determinar si su opinión sobre el tema había cambiado de alguna forma durante la prueba del juego. La mayoría de las preguntas de este cuestionario se centraron en evaluar la percepción subjetiva del juego, su nivel de utilidad, facilidad de utilización e intención de uso.

Los resultados de esta segunda encuesta indicaron que tras usar RoboTIC el 66,67 % de los participantes pensaba que programar es divertido y el 62,5 % quería aprender a programar en el futuro o dedicarse a tareas de computación o desarrollo de videojuegos.

A continuación se incluyen las páginas de las que se ha extraído esta información.

wants to add to the sequence, and once he/she has finished, execute the sequence to try to make the robot reach the target. The user has total freedom to remove from the sequence those instructions that he/she does not consider useful or replace them with others, as long as he/she respects the limitation on the number of instructions of each type available for the level. In special cases where a control instruction is added from the palette, the player's focus will shift to the sequence of instructions to complete the parts required by that control instruction.

The control instructions regarding conditional sentences are represented by a bifurcation on the road, assuming the left lane as the sequence to be executed if the condition is met, while the right lane would contain those instructions that will only be executed if the condition is not met. The condition to be evaluated in this type of control instructions can be selected by the player from a list of cards with predefined conditions.

In the case of control instructions concerning loops, their visual representation would be made by means of several sections of road arranged in the form of a roundabout, thus reflecting the concept of repetition. These instructions are limited only to the repetition of a number of times of the body of the loop (right part). The number of repetitions can be increased by interacting directly on the signal.

4 Experimental results

In order to evaluate the proposal of this work, an experiment has been conducted with children where they tested the game in a real scenario of augmented reality with the Microsoft HoloLens visualization device and in the context of a level that addressed programming concepts related to instructions, conditional sentences and loops. This section describes the performance of the experiment, the usage scenario, the participants, the results obtained and the possible limitations faced. Discussion of the results and their relation to the research questions formulated are left to the reader in the Section 5.

4.1 Participants

This experiment involved 12 children (9 boys and 3 girls) from the youth center of Torralba in Calatrava (Ciudad Real, Spain). The average age of the participants was 12 years old. None of them knew the game or had tried it before. None of the organizers of the experiment knew the participants beforehand.

4.2 Method

To conduct this experiment, the experimental process of Wohlin et al. [51] was followed, formulating the experimental objective using the template provided in the Goal-Question-Metric (GQM) [52] as shown below:

*To **analyze** the RoboTIC serious game for learning programming **with the purpose of** evaluating the user experience and to know the subjective opinion **with regard to** ease of use, usefulness, intention, interest and motivation of this game, **from the point of view** of children attending the school **within the context of** an scenario where there is no background in programming knowledge nor augmented reality.*

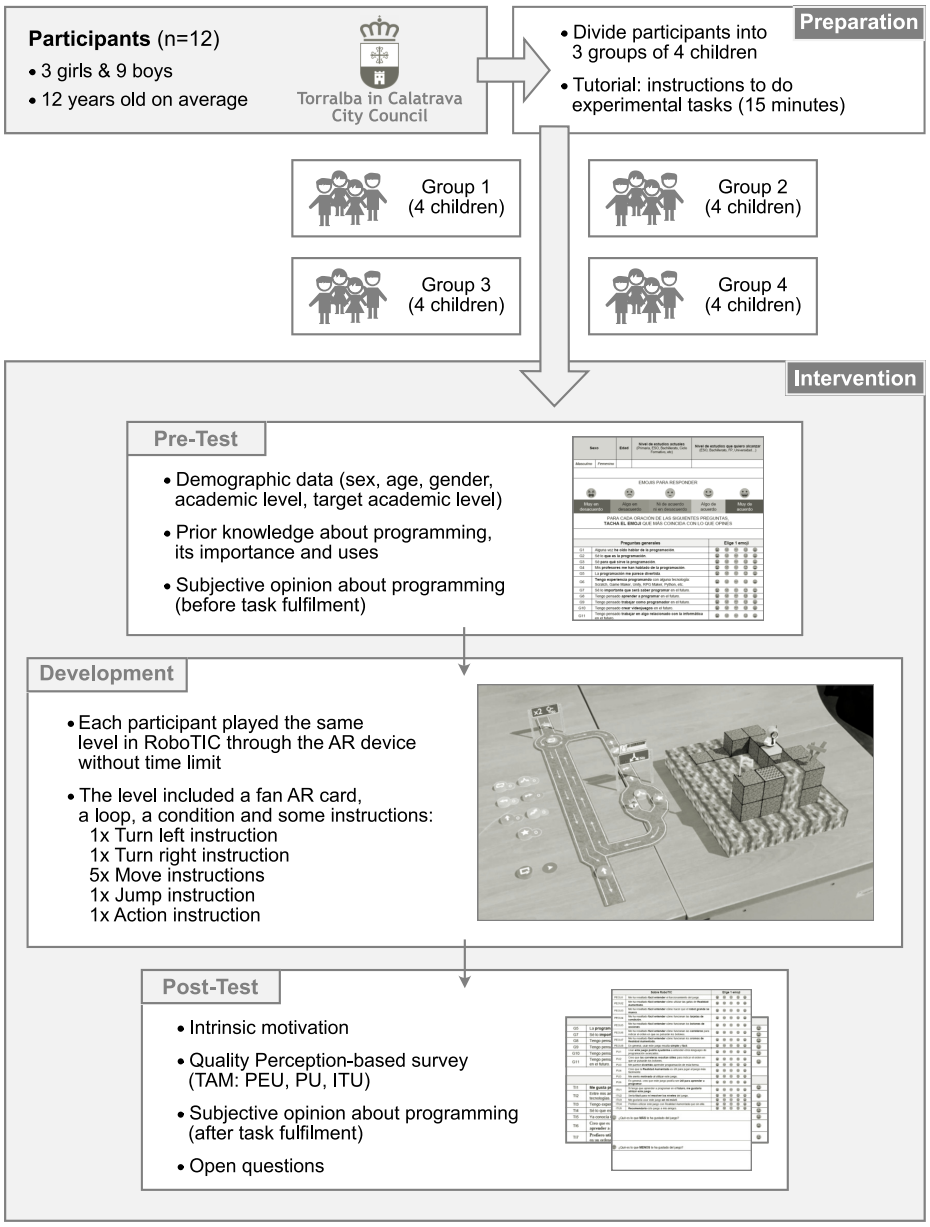


Fig. 4 Experimental design.

Fig. 4 graphically illustrates the experimental design followed in this study, describing the elements involved in each phase of the evaluation.

The experiment consisted of 2 phases; a first phase of preparation and another phase of intervention that included pre-test, development and post-test.

During the preparation phase the participants were divided into 3 groups of four children each given the reduced age of the participants, in order to isolate them in a room during the

Fig. 5 Photo taken during the preparation phase of the experiment.

experiment to reduce possible distractions and thus achieve a working group more focused in the experiment. In addition, a tutorial was given on the use of the game and the augmented reality device, explaining the possibilities of interaction, the functioning of the game and the objective they had to reach to solve the level. Fig. 5 shows a photo taken during this phase with the 4 members of the first group.

In the intervention phase, each working group performed three tasks of the experiment: i) pre-test, ii) development and iii) post-test. The total duration of the experiment for each working group was between 20 and 30 minutes, considering that each participant had to perform the development task individually. For both the pre-test and the post-test, the participants completed surveys in which several questions were asked in order to evaluate certain variables. In both tests, questions that were not open-ended had to be answered in a 5-point Likert scale. The answer options were replaced by *emojis* to make them easier to be understood by the participants [53] (1: *strongly disagree* being the angriest emoji to 5: *strongly agree* being the happiest emoji):

- *pre-test*: the participant's demographic information was requested, i.e. sex, age, current education level and desired future education level. In addition, some general questions were asked in the context of programming, in order to know their personal experience and opinion about video games, computers, and more specifically, about programming. In the Table 1 are included the questions of the questionnaire for this pre-test.
- *post-test*: some general programming questions were repeated again in order to determine whether they had changed their minds in any of their answers after the development phase. This test mainly included questions aimed at evaluating the participants' subjective perception of the game, inspired by the Technology Acceptance Model (TAM) [54] and analyzing the variables Perceived usefulness (PU), Perceived ease-of-use (PEOU) and Intention of use (ITU) of the model, as well as others aimed at learning about the participants' intrinsic motivation. Finally, some open-ended questions were included where participants could indicate what they liked most about the experience and what

they liked least. In the Table 2 are included the questions of the questionnaire for this post-test.

During the development phase, participants had to solve a RoboTIC level where various elements of the game were used, such as augmented reality stickers, instructions, use of loops and conditions (listed in Fig. 4). Thus, the level introduced a programming problem that had to be solved by a combination of the above elements (see video referenced in Section 2.1). The instructions allowed to define the sequence of commands to be executed at the level, while the loops and conditions allowed to define the execution flow.

Each participant had to put on the augmented reality device and start solving the level individually. The Microsoft HoloLens device allows streaming video via the web in order to see what the person wearing the device is seeing at that moment. This was used to see what the participants were doing to solve the level and how they interacted with the game.

Table 1 Items in the survey for the pre-test.

Item	Item statement
G1	I've ever heard of programming.
G2	I know what is programming.
G3	I know what programming is used for.
G4	My teachers have told me about programming.
G5	I find programming fun.
G6	I have experience programming with some technology: Scratch, Game Maker, Unity, RPG Maker, Python, etc.
G7	I know how important it will be to know how to program in the future.
G8	I intend to learn to program in the future.
G9	I intend to work as a programmer in the future.
G10	I intend to make video games in the future.
G11	I intend to work on something related to computer science in the future.

Table 2 Items in the survey for the *post-test*.

Item	Item statement
G5	I find programming fun.
G8	I intend to learn to program in the future.
G9	I intend to work as a programmer in the future.
G10	I intend to make video games in the future.
G11	I intend to work on something related to computer science in the future.
AR1	I have understood what is augmented reality.
AR2	I already knew the Microsoft HoloLens or other augmented reality device.
AR3	I think it's useful and fun to use augmented reality to learn how to program.
AR4	I prefer to use tools that use gestures and augmented reality rather than keyboard and mouse to learn how to program.
PEOU1	I found it easy to understand how the game works.
PEOU2	I found it easy to understand how to use the augmented reality device.
PEOU3	I found it easy to understand how to make the robot to move.
PEOU4	I found it easy to understand how the condition cards work.
PEOU5	I found it easy to understand how the action buttons work..
PEOU6	I have found it easy to understand how roads work to indicate the order in which buttons will be pressed.
PEOU7	I found it easy to understand how the augmented reality stickers work.
PEOU8	In general, using this game is simple and easy.
PU1	Using this game could help me understand other advanced programming languages.
PU2	I think roads are useful to indicate the order in which the buttons will be pressed.
PU3	I think it's fun to learn programming this way.
PU4	I think augmented aeality is useful for playing the game more easily.
PU5	I feel motivated to use this game.
PU6	Overall, I think this game could be useful for learning how to program.
ITU1	If I have to learn to program in the future, I would like to use this game.
ITU2	It would be easy for me to solve the levels of the game.
ITU3	I would like to use this game on my smartphone.
ITU4	I'd rather use this game with augmented reality than without it.
ITU5	I would recommend this game to my friends.

4.3 Results

First, the data obtained in the pre-test were analyzed considering the answers whose value on the Likert scale was greater or equal to 3, obtaining percentages of participants from the grouping of similar questions. This analysis concluded that 92% of the participants wanted to pursue higher education, only 12.5% had heard about programming, knew what it is or what it is used for (G1-G4 items) and none of them had previously programmed (G6 item), didn't want to dedicate to programming in the future (G8-G9 items), didn't know its importance (G7 item) or didn't think it was fun (G5 item), although 33.33% wanted to dedicate to creating video games or to computing in the future (G10-G11 items).

After analyzing again the same questions in the post-test part after the development phase and having been able to test a level of RoboTIC, it was found that now 66.67% of the participants thought programming was fun (item G5) and 62.5% plan to learn to program in the future, dedicate to computing or work creating video games. (item G8-G11).

Regarding the intrinsic motivation of the participants when using Augmented Reality tools, in the Table 3 the values for averages, standard deviations and medians of the related post-test items (ARx) are shown. The results obtained for items AR3 and AR4 show a mean score on the Likert scale above 4 indicating that participants were more predisposed to use augmented reality technologies to learn programming, even though they had never used an augmented reality device before (item AR2). The AR1 item is directly related to the explanation given to the participants of what augmented reality is previously in the preparation phase, demonstrating with a score on the Likert scale higher than 4 that they had understood the explanation.

In relation to the participants' subjective perception of ease of use (PEOU), utility (PU) and intention of use (ITU), the results obtained for the means, standard deviations and medians of the items of the post-test involved are shown in the Table 4, as well as the overall mean of each variable category. The results obtained show scores higher than 4 on the Likert scale for all cases except for the item PEOU4, whose standard deviation (0.79) is also far from that of the rest of the items. This may indicate that RoboTIC condition cards are one of the most difficult elements of the game to understand. On the contrary, the item PEOU3 presents a score of 5 with a standard deviation of 0.0, indicating a consensus among the participants on the understanding of the functioning of the movement of the main character from the instruction buttons. In particular, we would like to highlight the item PEOU6 related to the metaphor of roads and traffic signs, which allows us to validate the simplification of the notation ANGELA as suitable for the context of serious games.

Finally, the open-ended questions were answered in a variety of ways, focusing mainly on the fact that the participants liked the graphic aspect of the game, namely the instruction buttons, the roads and the movement of the mini-robot through them.

Table 3 Mean and median values for the intrinsic motivation regarding the AR items. Standard deviations are shown in parentheses.

Item	Mean	Median
AR1	4.33 (0.49)	4
AR2	1.08 (0.29)	1
AR3	4.58 (0.67)	5
AR4	4.42 (0.51)	4

Table 4 Mean and median values for the TAM framework variables; perceived ease-of-use (PEOU), perceived usefulness (PU) and intention of use (ITU). Standard deviations are shown in parentheses.

Item	Mean	Median	Mean (global)
PEOU1	4.17 (0.58)	4.0	4.35 (0.40)
PEOU2	4.50 (0.52)	4.5	
PEOU3	5.00 (0.00)	5.0	
PEOU4	3.58 (0.79)	3.0	
PEOU5	4.50 (0.52)	4.5	
PEOU6	4.33 (0.49)	4.0	
PEOU7	4.50 (0.52)	4.5	
PEOU8	4.25 (0.45)	4.0	
PU1	4.50 (0.52)	4.5	4.49 (0.19)
PU2	4.25 (0.45)	4.0	
PU3	4.67 (0.49)	5.0	
PU4	4.33 (0.49)	4.0	
PU5	4.75 (0.45)	5.0	
PU6	4.42 (0.51)	4.0	
ITU1	4.58 (0.51)	5.0	4.67 (0.13)
ITU2	4.67 (0.49)	5.0	
ITU3	4.75 (0.45)	5.0	
ITU4	4.50 (0.52)	4.5	
ITU5	4.83 (0.39)	5.0	

4.4 Study limitations

Despite the results obtained, there are some limitations and risks for the external validity of the study. In terms of the nature of the participants, given the small size of the sample treated, it has not been possible to conduct an exhaustive hypothesis validation, leading to a descriptive statistical analysis. Regarding the experimental task performed, although the level of play that the participants have tested is complete enough to include all the elements of the game and to deal with all the available programming concepts, experimental tasks that respect the expected learning flow are considered, so that the participants learn programming concepts incrementally.

5 Conclusions and future work

In this work we have presented RoboTIC, a serious game oriented to learning programming for children who have no coding skills. The different components that are integrated into the RoboTIC architecture have been introduced within the context of the learning possibilities it provides thanks to the use of augmented reality and the game mechanics designed to motivate students. This layered architecture is scalable enough to add new levels and multimedia resources that make possible to address new programming concepts and techniques.

The analysis of the results obtained after the evaluation of RoboTIC concludes answering the research questions posed at the beginning of this research work. Thus, with regard to the children's motivation and interest in programming (RQ1) it is concluded that the students have felt more motivated after playing RoboTIC and have been more receptive to learning programming. Regarding augmented reality, the study reveals that its use as an immersive technology improves the children's motivation towards programming, making learning more attractive (RQ2). Finally, the analysis of the subjective perception towards ease of use, usefulness and intention to use RoboTIC shows that the serious game proposed in this work is

ANEXO B

MANUAL DE USO DE ROBOTIC

En este anexo se recoge un breve manual de usuario de RoboTIC para mejorar la comprensión de la jugabilidad del mismo.

B.1. MENÚ PRINCIPAL

Figura B.1: Menú principal de RoboTIC

El menú principal es muy sencillo pues lo único que contiene son los botones [PLAY], [EDITOR] y [EXIT]. El usuario deberá hacer el gesto *tap* sobre el botón que quiera pulsar mientras mira al mismo.

- [PLAY]: este botón da acceso al menú de selección de niveles.
- [EDITOR]: abre el editor de niveles.
- [EXIT]: cierra la aplicación.

B.2. EL JUEGO

Si el jugador ha pulsado el botón [PLAY] se abrirá el menú de selección de niveles donde podrá elegir el que desee jugar.

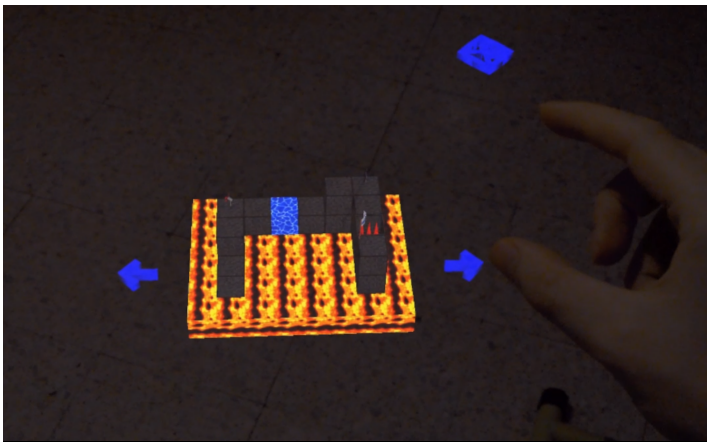


Figura B.2: Menú de selección de niveles de RoboTIC

Este menú se puede navegar usando las flechas dispuestas a la izquierda y la derecha del mapa actualmente seleccionado. El símbolo azul que se puede ver en la esquina superior derecha sirve para mover el escenario del juego de sitio, por ejemplo a otra superficie. Una vez elegido el nivel apropiado basta con hacer *tap* sobre el mismo para empezar a jugar.

Figura B.3: Escenario de juego

En la imagen se pueden ver, de izquierda a derecha, el tramo de carretera inicial, los botones de generación de carreteras y el nivel a superar con el robot. Al pulsar en un botón de generación de carreteras se pondrá un tramo en el lugar elegido si es posible (y si el número del contador de su derecha es mayor que cero). Después de colocar un tramo el número de su contador se decrementa en un unidad. Es posible seleccionar donde poner un tramo de carretera arrastrando la flecha azul. La flecha roja apunta a la posición válida más cercana.

Figura B.4: Seleccionando donde colocar una nueva carretera

A la derecha del espacio de juego se encuentran una serie de botones:

- [UNDO]: deshace las acciones del usuario, una por cada click.
- [PLAY]: ejecuta el algoritmo que haya escrito el usuario.
- [MENU]: vuelve al menú principal.
- [RESTART]: reinicia el nivel.

Figura B.5: Botones del escenario del juego

Una vez que el usuario esté contento con su sistema de carreteras debe pulsar el botón [PLAY] para ejecutar el nivel:

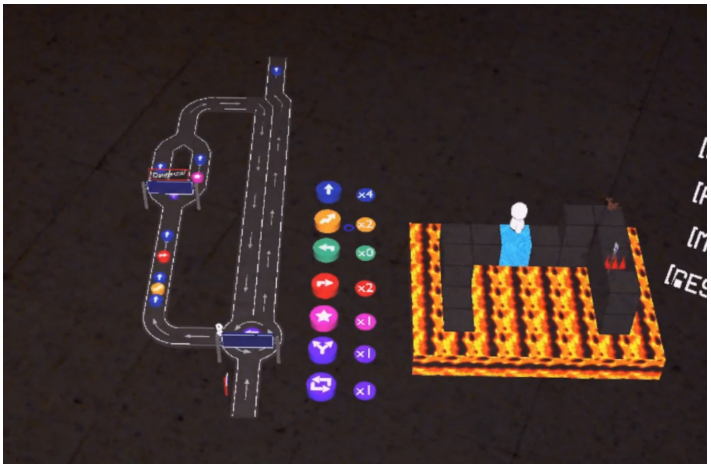


Figura B.6: Ejecución del juego en proceso

Dependiendo de si el robot llega o no a la meta se mostrará la pantalla de ganar o perder. En ambos casos se da al usuario la posibilidad de reiniciar el nivel o volver al menú.

Figura B.7: Pantallas de ganar y perder

B.3. EL EDITOR DE NIVELES

Tras elegir el botón correspondiente en el menú principal se llevará al usuario al editor de niveles.

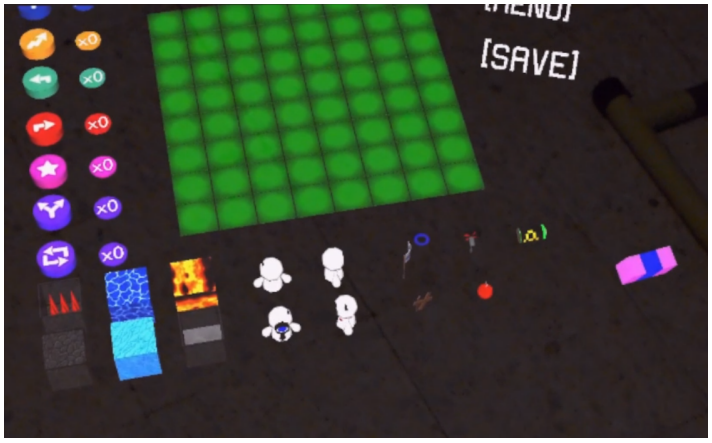


Figura B.8: Editor de niveles

En la parte izquierda de la imagen se encuentran los botones para añadir instrucciones acompañados cada uno de su contador correspondiente y puede haber hasta un máximo de 9 de cada instrucción.

Abajo se encuentran los bloques, el robot principal en cada una de sus orientaciones posibles y los items. Haciendo *tap* en alguno de estos objetos lo seleccionaremos y podremos añadirlo al mapa haciendo el mismo gesto sobre los cuadrados verdes. La goma de borrar de más a la derecha sirve para eliminar objetos del escenario. Una vez contento con el mapa se puede guardar con el botón [SAVE].

Figura B.9: Creación del nivel

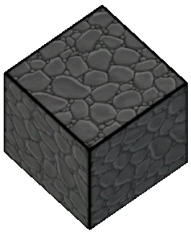
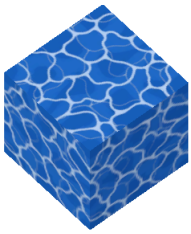
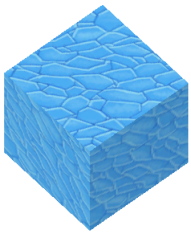
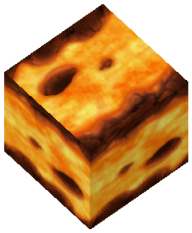
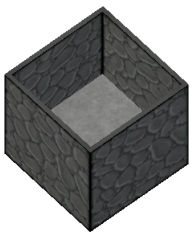
A la hora de crear los niveles hay algunas restricciones:

- Solo se pueden poner items y robots sobre bloques.
- Solo puede haber un robot.
- Debe haber una bandera para simbolizar la meta del nivel.
- El tamaño máximo del nivel es de 8 x 4 x 8 bloques.

B.4. BLOQUES

En la siguiente tabla se muestran los bloques que pueden componer los escenarios del juego acompañados de una pequeña descripción.

Tabla B.1: Bloques que aparecen en el juego

Bloque	Descripción
	El bloque Solid es la base de los niveles y como tal el jugador puede andar y saltar sobre él sin ningún tipo de penalización. Se puede destruir usando items con el efecto apropiado y se puede obtener al congelar Lava.
	El bloque Water representa agua y es peligroso para el jugador pues si intenta caminar sobre un bloque de este tipo se acabará la partida. Se puede congelar para formar un bloque Ice.
	El bloque Ice resulta de congelar el bloque Water y en esencia funciona como el bloque Solid y al igual que este puede ser tocado sin peligro y destruido con el item adecuado.
	El bloque Lava es peligroso y no se puede andar sobre él. Generalmente funciona igual que un bloque Water pero al ser congelado genera el bloque Solid.
	El bloque Spikes representa un agujero con pinchos al fondo por lo que provoca la muerte del robot si se pisa. Puede ser neutralizado o destruido con los items correspondientes.
	El bloque Lift es una plataforma que debe ser activada mediante el uso de items para que el robot pueda caminar por encima. Para aumentar las posibilidades en cuanto a formas de enfrentarse a un nivel también se permite que sea destruida.

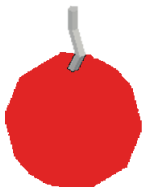
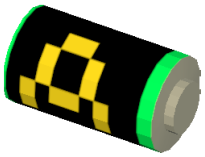
Bloque	Descripción
	Aunque invisible, el bloque NoBlock es imprescindible para el juego pues permite que el jugador camine a través de él y en general se puede encontrar en cualquier espacio en el que no haya otro bloque.

Tabla B.1: Bloques que aparecen en el juego

B.5. ITEMS

Los items que el usuario puede usar para superar los niveles se describen en la tabla que se muestra a continuación.

Tabla B.2: Items de RoboTIC

Item	Descripción
	El item Flag se usa para marcar la meta de un nivel. Todos los niveles deben tener una bandera y no puede ser tomada por el jugador.
	El item Bomb destruye ciertos bloques al ser usado en ellos, principalmente bloque sólidos como Solid o Ice.
	El item Activator activa los bloques con los que se usa (y son compatibles con él). Usando este item en el bloque Lift conseguiremos que suba la plataforma y el robot pueda pasar al otro lado.
	El item Fan congela los bloques líquidos, en este caso Water y Lava transformándolos en sólidos y permitiendo ser pisados por el jugador o destruidos con el objeto Bomb.

Item	Descripción
	El item Planks representa unas planchas de madera cuyo uso es ser colocadas en el bloque Spikes para convertirlo en seguro para el jugador.

Tabla B.2: Items de RoboTIC

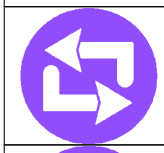
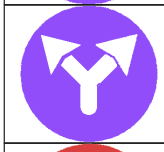
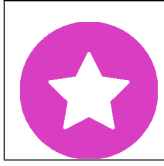
B.6. LOS TRAMOS DE CARRETERA

Se ha hablado anteriormente de los tramos de carretera pero ahora se concretará qué botones generan a cada tramo y su función específica.

B.6.1. Los botones

Estos botones se pueden encontrar tanto en la pantalla de juego principal (para generar carreteras) como en el editor (para cambiar el número de instrucciones de un mismo tipo disponibles).

Tabla B.3: Botones de RoboTIC

Botón	Descripción
	Este botón genera una estructura de tipo bucle.
	Este botón genera una estructura de tipo *if*.
	Este botón añade un tramo de carretera que hace girar al robot 90º a la derecha.
	Este botón añade un tramo de carretera que hace girar al robot 90º a la izquierda.
	Este botón añade un tramo que hace saltar al robot al bloque que tenga enfrente.
	Este botón añade un tramo de carretera que hace que el robot use el último objeto que ha recogido.

Botón	Descripción
	Este botón añade un tramo de carretera que hace que el robot avance al bloque que tiene justo enfrente.

Tabla B.3: Botones de RoboTIC

B.6.2. Tramo de funciones del robot

Los tramos de funciones del robot toman la forma de una pieza de carretera vertical de un solo carril con entre uno y tres botones dentro representando acciones del robot que se ejecutarán de forma secuencial.

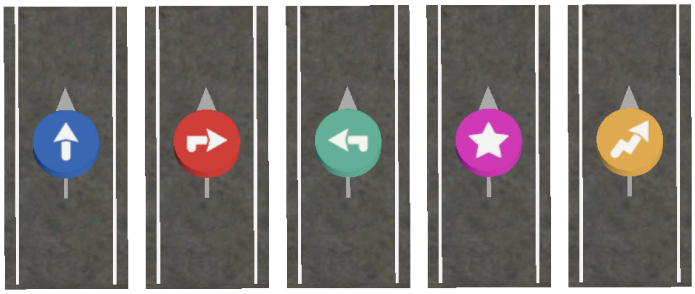


Figura B.10: Tramos con las diferentes acciones que puede llevar a cabo el robot

B.6.3. Tramo condicional

El lenguaje soporta estructuras condicionales en las cuales se permite seleccionar una carta de condición y dependiendo de si dicha condición se cumple o no en el bloque al que vaya a avanzar el robot el flujo de ejecución tomará un camino u otro. Para seleccionar una carta de condición concreta haremos *tap* en la carta en uso actualmente para desplegar todas las posibles y acto seguido repetiremos el gesto sobre la que queramos seleccionar para confirmar la elección.

Figura B.11: Estructura tipo *if* o condicional

Las cartas de condición que incluye el juego son las siguientes:

- Dangerous. En este caso la condición será verdadera si el bloque de enfrente del robot es peligroso de alguna manera, por ejemplo si es lava o agua y por tanto no debería pisarlo.
- Destructible. Si el bloque frente al jugador es destruible la condición será satisfecha.
- Freezable. Congelable, por ejemplo en el caso del agua que se puede congelar para formar hielo o de la lava para conseguir un bloque sólido.
- Usable. Esta condición será verdad si el jugador tiene en la posición más alta de su inventario un item que pueda ser usado con el bloque que se encuentra enfrente de él.
- Walkable. Verdad si el bloque de enfrente al robot se puede pisar de manera segura (hielo y bloque sólidos).

B.6.4. Tramo de bucle

En RoboTIC, el bucle representa una estructura de bucle *for* de la que el usuario puede elegir el número de iteraciones pulsando sobre el contador de la misma carretera, entre cero y nueve, y el robot iterará en ella hasta que el número de repeticiones restantes sea igual que cero.

Figura B.12: Estructura tipo bucle

ANEXO C

DOCUMENTACIÓN DEL PROGRAMA

La documentación del programa se ha generado usando Doxygen y a continuación se presenta una versión abreviada de la misma para evitar que este documento sea demasiado voluminoso. Aún así, cualquier persona interesada en consultar una versión más completa de esta documentación puede generarla a conveniencia. Especialmente interesantes son los miembros privados y los diagramas que por desgracia no se han podido incluir aquí pues superaban las 400 paginas de extensión lo cual sinceramente considero un gasto de papel innecesario.

También existe una página web que contiene la documentación del programa con todo tipo de detalles y diagramas que se puede consultar en el siguiente enlace:

Listado C.1: Enlace a la documentación de RoboTIC

```
1 https://ohespaco.github.io/RoboTIC/annotated.html
```

RoboTIC

Prototype 4

Generado por Doxygen 1.8.18

1. Indice de namespaces

1.1. Paquetes

Aquí van los paquetes con una breve descripción (si etá disponible):

2. Indice jerárquico

2.1. Jerarquía de la clase

Esta lista de herencias esta ordenada aproximadamente por orden alfabético:

3. Índice de clases

3.1. Lista de clases

Lista de las clases, estructuras, uniones e interfaces con una breve descripción:

4. Documentación de namespaces

4.1. Referencia del Namespace Academy

4.2. Referencia del Namespace Academy.HoloToolkit

4.3. Referencia del Namespace Academy.HoloToolkit.Unity

Clases

- class BasicCursor
- struct BoundedPlane
- class GazeManager

  GazeManager determines the location of the user's gaze, hit position and normals.
- class GestureManager

  GestureManager creates a gesture recognizer and signs up for a tap gesture. When a tap gesture is detected, GestureManager uses GazeManager to find the game object. GestureManager then sends a message to that game object.
- class HandsManager

  HandsDetected determines if the hand is currently detected or not.
- class ObjectSurfaceObserver
- struct OrientedBoundingBox
- class PlaneFinding
- class RemoveSurfaceVertices

  RemoveSurfaceVertices will remove any vertices from the Spatial Mapping Mesh that fall within the bounding volume. This can be used to create holes in the environment, or to help reduce triangle count after finding planes.
- class SpatialMappingManager

  The SpatialMappingManager class allows applications to use a SurfaceObserver or a stored Spatial Mapping mesh (loaded from a file). When an application loads a mesh file, the SurfaceObserver is stopped. Calling StartObserver() clears the stored mesh and enables real-time SpatialMapping updates.
- class SpatialMappingObserver

  The SpatialMappingObserver class encapsulates the SurfaceObserver into an easy to use object that handles managing the observed surfaces and the rendering of surface geometry.
- class SpatialMappingSource
- class SurfaceMeshesToPlanes

  SurfaceMeshesToPlanes will find and create planes based on the meshes returned by the SpatialMappingManager's Observer.
- class SurfacePlane

  The SurfacePlane class is used by SurfaceMeshesToPlanes to create different types of planes (walls, floors, tables, etc.) based on the Spatial Mapping data returned by the SpatialMappingManager's source. This script should be a component on the SufacePlane prefab, which is used by SurfaceMeshesToPlanes.

Enumeraciones

- enum ObserverStates { ObserverStates.Running = 0, ObserverStates.Stopped = 1 }

  Spatial Mapping Observer states.
- enum PlaneTypes {
  Wall = 0x1, **Floor** = 0x2, **Ceiling** = 0x4, **Table** = 0x8,
  Unknown = 0x10 }

  All possible plane types that a SurfacePlane can be.

4.3.1. Documentación de las enumeraciones

4.3.1.1. ObserverStates `enum Academy.HoloToolkit.Unity.ObserverStates [strong]`

Spatial Mapping Observer states.

Valores de enumeraciones

Running	The SurfaceObserver is currently running.
Stopped	The SurfaceObserver is currently idle.

4.3.1.2. PlaneTypes `enum Academy.HoloToolkit.Unity.PlaneTypes [strong]`

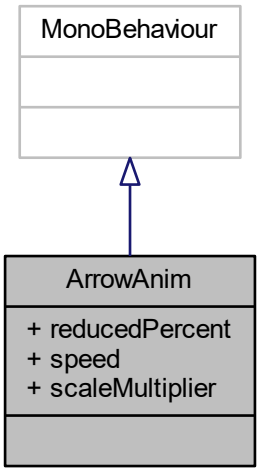
All possible plane types that a SurfacePlane can be.

5. Documentación de las clases

5.1. Referencia de la Clase ArrowAnim

Clase que lleva a cabo la animación de las flechas del mapa.

Diagrama de herencias de ArrowAnim



Atributos públicos

- float reducedPercent = 0.5f

  Porcentaje de tamaño que debe disminuir el objeto.
- float speed = 10f

  Velocidad de la animación.
- float scaleMultiplier = 20f

  Multiplicador de la escala.

5.1.1. Descripción detallada

Clase que lleva a cabo la animación de las flechas del mapa.

5.1.2. Documentación de los datos miembro

5.1.2.1. reducedPercent `float ArrowAnim.reducedPercent = 0.5f`

Porcentaje de tamaño que debe disminuir el objeto.

5.1.2.2. scaleMultiplier `float ArrowAnim.scaleMultiplier = 20f`

Multiplicador de la escala.

5.1.2.3. speed `float ArrowAnim.speed = 10f`

Velocidad de la animación.

La documentación para esta clase fue generada a partir del siguiente fichero:

- Assets/Scripts/MapMenu/ArrowAnim.cs

5.2. Referencia de la Clase AvailableInstructions

Guarda las instrucciones que tiene el jugador disponibles.

Atributos públicos

- int condition
  Condición.
- int loop
  Bucle.
- int turnRight
  Giro a la derecha.
- int turnLeft
  Giro a la izquierda.
- int jump
  Salto.
- int move
  Movimiento.
- int action
  Acción.

5.2.1. Descripción detallada

Guarda las instrucciones que tiene el jugador disponibles.

5.2.2. Documentación de los datos miembro

5.2.2.1. action `int AvailableInstructions.action`

Acción.

5.2.2.2. condition `int AvailableInstructions.condition`

Condición.

5.2.2.3. jump `int AvailableInstructions.jump`

Salto.

5.2.2.4. loop `int AvailableInstructions.loop`

Bucle.

5.2.2.5. move `int AvailableInstructions.move`

Movimiento.

5.2.2.6. turnLeft `int AvailableInstructions.turnLeft`

Giro a la izquierda.

5.2.2.7. turnRight `int AvailableInstructions.turnRight`

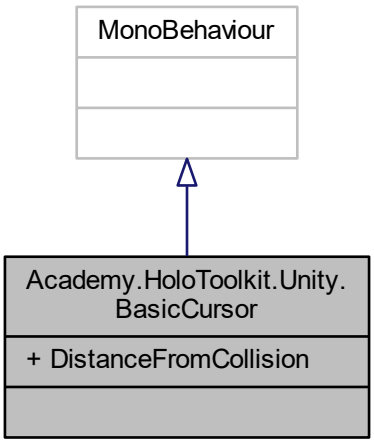
Giro a la derecha.

La documentación para esta clase fue generada a partir del siguiente fichero:

- Assets/Scripts/DataStructures/LevelData.cs

5.3. Referencia de la Clase Academy.HoloToolkit.Unity.BasicCursor

Diagrama de herencias de Academy.HoloToolkit.Unity.BasicCursor



Atributos públicos

- float **DistanceFromCollision** = 0.01f

5.3.1. Descripción detallada

1. Decides when to show the cursor.
2. Positions the cursor at the gazed location.
3. Rotates the cursor to match hologram normals.

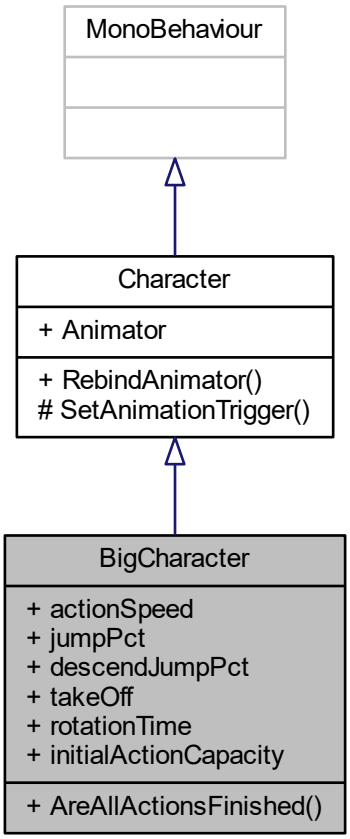
La documentación para esta clase fue generada a partir del siguiente fichero:

- Assets/HoloToolkit-SpatialMapping-230/Input/Scripts/BasicCursor.cs

5.4. Referencia de la Clase BigCharacter

La clase BigCharacter contiene .

Diagrama de herencias de BigCharacter



Métodos públicos

- bool AreAllActionsFinished ()

  Comprueba si han terminado todas las acciones del robot.

Atributos públicos

- float actionSpeed = 0.5f

  Velocidad a la que van las acciones.
- float jumpPct = 1f

  Porcentaje de la altura del salto hacia arriba respecto a la del bloque.
- float descendJumpPct = 0.3f

  Porcentaje de la altura del salto hacia abajo respecto a la altura total del salto.
- float takeOff = 0.8f

  Porcentaje del tiempo de salto que se pasa ascendiendo.
- float rotationTime = 1f

  Tiempo que dura la rotación.
- int initialActionCapacity = 20

  Capacidad inicial de la lista de acciones pendientes.

Otros miembros heredados

5.4.1. Descripción detallada

La clase BigCharacter contiene .

5.4.2. Documentación de las funciones miembro

5.4.2.1. AreAllActionsFinished() `bool BigCharacter.AreAllActionsFinished ( )`

Comprueba si han terminado todas las acciones del robot.

Devuelve

True si han terminado todas las acciones.

5.4.3. Documentación de los datos miembro

5.4.3.1. actionSpeed `float BigCharacter.actionSpeed = 0.5f`

Velocidad a la que van las acciones.

5.4.3.2. descendJumpPct `float BigCharacter.descendJumpPct = 0.3f`

Porcentaje de la altura del salto hacia abajo respecto a la altura total del salto.

5.4.3.3. initialActionCapacity `int BigCharacter.initialActionCapacity = 20`

Capacidad inicial de la lista de acciones pendientes.

5.4.3.4. jumpPct `float BigCharacter.jumpPct = 1f`

Porcentaje de la altura del salto hacia arriba respecto a la del bloque.

5.4.3.5. rotationTime `float BigCharacter.rotationTime = 1f`

Tiempo que dura la rotación.

5.4.3.6. takeOff `float BigCharacter.takeOff = 0.8f`

Porcentaje del tiempo de salto que se pasa ascendiendo.

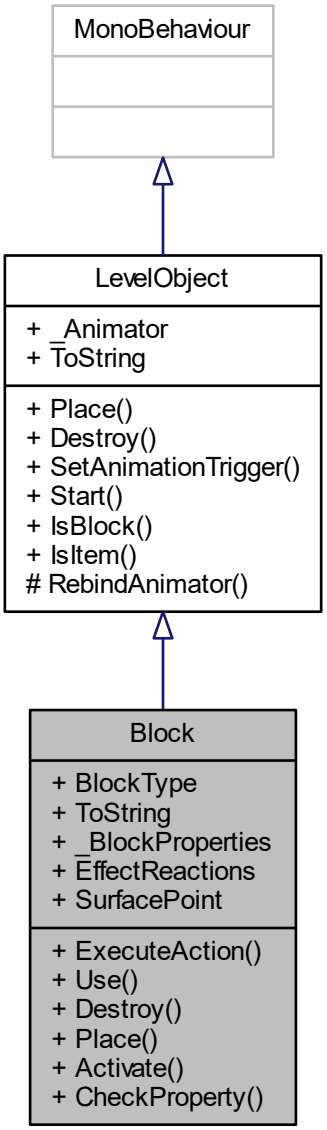
La documentación para esta clase fue generada a partir del siguiente fichero:

- Assets/Scripts/Character/BigCharacter.cs

5.5. Referencia de la Clase Block

Define la clase Block.

Diagrama de herencias de Block



Clases

- class EffectReaction

  La clase EffectReaction modela las reacciones que pueden tener los items sobre este bloque.

Tipos públicos

- enum BlockActions {
  Use, **Destroy**, **Place**, **Activate**,
  Rebind }

Enum de acciones que soporta el bloque.

- enum BlockProperties {
  Immaterial, **Walkable**, **Dangerous**, **Icy**,
  Destructible, **Usable**, **Freezable** }

  Enum de propiedades que pueden tener los bloques

Métodos públicos

- void ExecuteAction (BlockActions action)

  Ejecuta una acción.
- void Use ()

  Acción usar.
- override void Destroy ()

  Acción destruir.
- override void Place ()

  Acción de colocar el bloque.
- void Activate ()

  Acción de activar el bloque.
- bool CheckProperty (BlockProperties property)

  Comprueba que el bloque cumpla una propiedad.

Propiedades

- Blocks BlockType `[get]`

  Retorna el tipo del bloque.
- override string ToString `[get]`

  ToString
- BlockProperties[ ] _BlockProperties `[get, set]`

  Retorna o modifica las propiedades del bloque.
- EffectReaction[ ] EffectReactions `[get, set]`

  Retorna o modifica las reacciones del bloque a efectos.
- Vector3 SurfacePoint `[get]`

  Calcula el centro de la superficie del bloque.

Otros miembros heredados

5.5.1. Descripción detallada

Define la clase Block.

5.5.2. Documentación de las enumeraciones miembro de la clase

5.5.2.1. BlockActions `enum Block.BlockActions [strong]`

Enum de acciones que soporta el bloque.

5.5.2.2. BlockProperties `enum Block.BlockProperties [strong]`

Enum de propiedades que pueden tener los bloques

5.5.3. Documentación de las funciones miembro

5.5.3.1. Activate() `void Block.Activate ( )`

Acción de activar el bloque.

5.5.3.2. CheckProperty() `bool Block.CheckProperty ( BlockProperties property )`

Comprueba que el bloque cumpla una propiedad.

Parámetros

property	La propiedad BlockProperties.

Devuelve

True si la cumple, false si no bool.

5.5.3.3. Destroy() `override void Block.Destroy ( ) [virtual]`

Acción destruir.

Implementa LevelObject.

5.5.3.4. ExecuteAction() `void Block.ExecuteAction ( BlockActions action )`

Ejecuta una acción.

Parámetros

action	La acción a ejecutar BlockActions.

5.5.3.5. Place() override void Block.Place ( ) [virtual]

Acción de colocar el bloque.

Implementa LevelObject.

5.5.3.6. Use() void Block.Use ( )

Acción usar.

5.5.4. Documentación de propiedades

5.5.4.1. _BlockProperties BlockProperties [] Block._BlockProperties [get], [set]

Retorna o modifica las propiedades del bloque.

5.5.4.2. BlockType Blocks Block.BlockType [get]

Retorna el tipo del bloque.

5.5.4.3. EffectReactions EffectReaction [] Block.EffectReactions [get], [set]

Retorna o modifica las reacciones del bloque a efectos.

5.5.4.4. SurfacePoint Vector3 Block.SurfacePoint [get]

Calcula el centro de la superficie del bloque.

5.5.4.5. ToString override string Block.ToString [get]

ToString

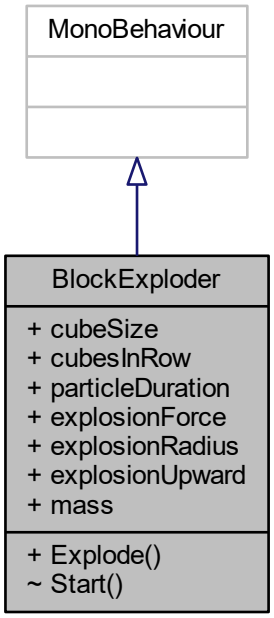
La documentación para esta clase fue generada a partir del siguiente fichero:

- Assets/Scripts/Blocks/Block.cs

5.6. Referencia de la Clase BlockExploder

Clase que hace explotar un bloque como una serie de cubos.

Diagrama de herencias de BlockExploder



Métodos públicos

- void Explode ()

  Hace explotar el bloque.

Atributos públicos

- float cubeSize = 0.2f

  Tamaño del cubo.
- int cubesInRow = 5

  Cubos en una fila.
- float particleDuration = 1f

  Duración de las partículas.
- float explosionForce = 50f

  Fuerza de la explosión.
- float explosionRadius = 4f

  Radio de la explosión.
- float explosionUpward = 0.4f

  Hace que la explosión mueva los objetos hacia arriba.
- float mass = 0.1f

  Masa de las partículas.

5.6.1. Descripción detallada

Clase que hace explotar un bloque como una serie de cubos.

5.6.2. Documentación de las funciones miembro

5.6.2.1. Explode() `void BlockExploder.Explode ( )`

Hace explotar el bloque.

5.6.3. Documentación de los datos miembro

5.6.3.1. cubesInRow `int BlockExploder.cubesInRow = 5`

Cubos en una fila.

5.6.3.2. cubeSize `float BlockExploder.cubeSize = 0.2f`

Tamaño del cubo.

5.6.3.3. explosionForce `float BlockExploder.explosionForce = 50f`

Fuerza de la explosión.

5.6.3.4. explosionRadius `float BlockExploder.explosionRadius = 4f`

Radio de la explosión.

5.6.3.5. explosionUpward `float BlockExploder.explosionUpward = 0.4f`

Hace que la explosión mueva los objetos hacia arriba.

5.6.3.6. mass `float BlockExploder.mass = 0.1f`

Masa de las partículas.

5.6.3.7. particleDuration `float BlockExploder.particleDuration = 1f`

Duración de las partículas.

La documentación para esta clase fue generada a partir del siguiente fichero:

- Assets/Scripts/Visual/BlockExploder.cs

5.7. Referencia de la Estructura Academy.HoloToolkit.Unity.BoundedPlane

Métodos públicos

- BoundedPlane (Transform xform)

  Builds the bounded plane to match the obb defined by xform

Atributos públicos

- Plane **Plane**
- OrientedBoundingBox **Bounds**
- float **Area**

5.7.1. Documentación del constructor y destructor

5.7.1.1. BoundedPlane() `Academy.HoloToolkit.Unity.BoundedPlane.BoundedPlane ( Transform xform )`

Builds the bounded plane to match the obb defined by xform

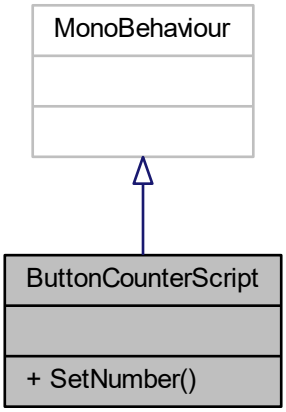
La documentación para esta estructura fue generada a partir del siguiente fichero:

- Assets/HoloToolkit-SpatialMapping-230/SpatialMapping/Scripts/PlaneFinding.cs

5.8. Referencia de la Clase ButtonCounterScript

Actúa como intermediario con el contador de un botón.

Diagrama de herencias de ButtonCounterScript



Métodos públicos

- int SetNumber (int number)

  Pone un número en ese contador.

5.8.1. Descripción detallada

Actúa como intermediario con el contador de un botón.

5.8.2. Documentación de las funciones miembro

5.8.2.1. SetNumber() `int ButtonCounterScript.SetNumber ( int number )`

Pone un número en ese contador.

Parámetros

number	El número a poner.

Devuelve

El número que se ha puesto.

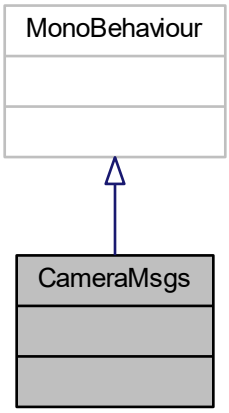
La documentación para esta clase fue generada a partir del siguiente fichero:

- Assets/Scripts/Roads/ButtonCounterScript.cs

5.9. Referencia de la Clase CameraMsgs

Pequeña clase que responde a las peticiones de mensajes que se hacen a la cámara CameraMsgs.

Diagrama de herencias de CameraMsgs



5.9.1. Descripción detallada

Pequeña clase que responde a las peticiones de mensajes que se hacen a la cámara CameraMsgs.

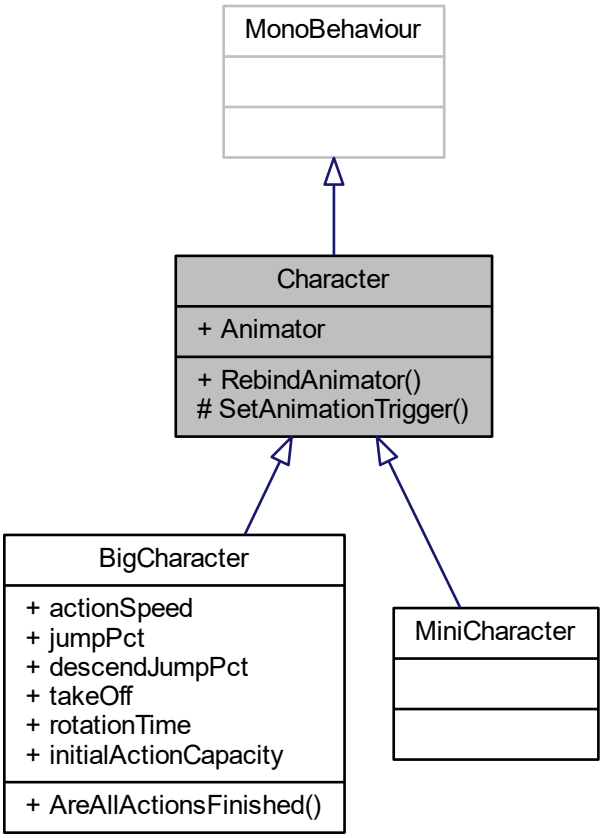
La documentación para esta clase fue generada a partir del siguiente fichero:

- Assets/Scripts/Camera/CameraMsgs.cs

5.10. Referencia de la Clase Character

Clase abstracta de la que descienden los robots.

Diagrama de herencias de Character



Métodos públicos

- void RebindAnimator ()

  Resetea el animator.

Métodos protegidos

- void SetAnimationTrigger (in string trigger)

  Ejecuta un trigger de animación.

Propiedades

- Animator Animator `[get]`

  Retorna el animator.

5.10.1. Descripción detallada

Clase abstracta de la que descienden los robots.

5.10.2. Documentación de las funciones miembro

5.10.2.1. RebindAnimator() `void Character.RebindAnimator ( )`

Resetea el animator.

5.10.2.2. SetAnimationTrigger() `void Character.SetAnimationTrigger ( in string trigger ) [protected]`

Ejecuta un trigger de animación.

Parámetros

trigger	El trigger.

5.10.3. Documentación de propiedades

5.10.3.1. Animator `Animator Character.Animator [get]`

Retorna el animator.

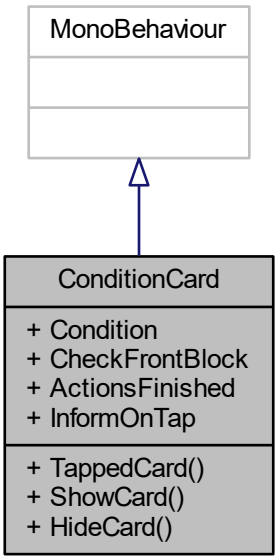
La documentación para esta clase fue generada a partir del siguiente fichero:

- Assets/Scripts/Character/Character.cs

5.11. Referencia de la Clase ConditionCard

Clase de las cartas de condición.

Diagrama de herencias de ConditionCard



Métodos públicos

- delegate void TappedCard (ConditionCard card)

  Delegate para informar de que la carta ha sido tocada.
- void ShowCard ()

  Despliega la carta.
- void HideCard ()

  Esconde la carta.

Propiedades

- BlockProperties Condition `[get]`

  Retorna la condición de la carta.
- bool CheckFrontBlock `[get]`

  Retorna un valor indicando si hay que comprobar el bloque de enfrente.
- bool ActionsFinished `[get]`

  Indica si las acciones han terminado.
- TappedCard InformOnTap `[get, set]`

  Añade métodos al delegado.

5.11.1. Descripción detallada

Clase de las cartas de condición.

5.11.2. Documentación de las funciones miembro

5.11.2.1. HideCard() `void ConditionCard.HideCard ( )`

Esconde la carta.

5.11.2.2. ShowCard() `void ConditionCard.ShowCard ( )`

Despliega la carta.

5.11.2.3. TappedCard() `delegate void ConditionCard.TappedCard ( ConditionCard card )`

Delegate para informar de que la carta ha sido tocada.

Parámetros

card	La carta tocada.

5.11.3. Documentación de propiedades

5.11.3.1. ActionsFinished `bool ConditionCard.ActionsFinished [get]`

Indica si las acciones han terminado.

5.11.3.2. CheckFrontBlock `bool ConditionCard.CheckFrontBlock [get]`

Retorna un valor indicando si hay que comprobar el bloque de enfrente.

5.11.3.3. Condition `BlockProperties ConditionCard.Condition [get]`

Retorna la condición de la carta.

5.11.3.4. InformOnTap `TappedCard ConditionCard.InformOnTap [get], [set]`

Añade métodos al delegado.

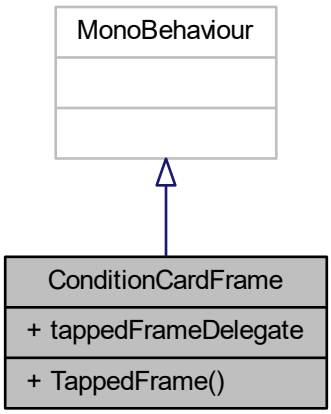
La documentación para esta clase fue generada a partir del siguiente fichero:

- Assets/Scripts/ConditionCards/ConditionCard.cs

5.12. Referencia de la Clase ConditionCardFrame

La clase ConditionCardFrame incorpora la parte física de la carta e informa de si se ha tocado.

Diagrama de herencias de ConditionCardFrame



Métodos públicos

- delegate void TappedFrame ()

  Delegado para informar de que se ha tocado la carta.

Atributos públicos

- TappedFrame tappedFrameDelegate

  Instancia del delegado para informar de que se ha tocado la carta.

5.12.1. Descripción detallada

La clase ConditionCardFrame incorpora la parte física de la carta e informa de si se ha tocado.

5.12.2. Documentación de las funciones miembro

5.12.2.1. TappedFrame() `delegate void ConditionCardFrame.TappedFrame ( )`

Delegado para informar de que se ha tocado la carta.

5.12.3. Documentación de los datos miembro

5.12.3.1. tappedFrameDelegate `TappedFrame ConditionCardFrame.tappedFrameDelegate`

Instancia del delegado para informar de que se ha tocado la carta.

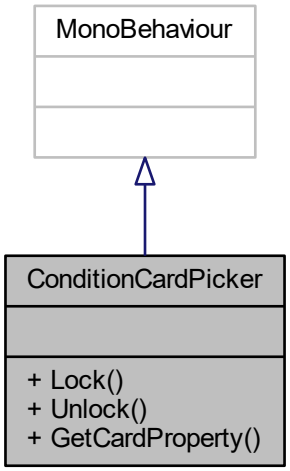
La documentación para esta clase fue generada a partir del siguiente fichero:

- Assets/Scripts/ConditionCards/ConditionCardFrame.cs

5.13. Referencia de la Clase ConditionCardPicker

Esta clase modela el objeto que permite elegir cartas de condición.

Diagrama de herencias de ConditionCardPicker



Métodos públicos

- void Lock ()

  Bloquea el picker para que no responda a las acciónes del usuario.
- void Unlock ()

  Desbloquea el picker para que responda a las acciónes del usuario.
- BlockProperties GetCardProperty ()

  Retorna la propiedad de la carta seleccionada.

5.13.1. Descripción detallada

Esta clase modela el objeto que permite elegir cartas de condición.

5.13.2. Documentación de las funciones miembro

5.13.2.1. GetCardProperty() `BlockProperties ConditionCardPicker.GetCardProperty ( )`

Retorna la propiedad de la carta seleccionada.

Devuelve

La propiedad BlockProperties.

5.13.2.2. Lock() `void ConditionCardPicker.Lock ( )`

Bloquea el picker para que no responda a las acciónes del usuario.

5.13.2.3. Unlock() `void ConditionCardPicker.Unlock ( )`

Desbloquea el picker para que responda a las acciónes del usuario.

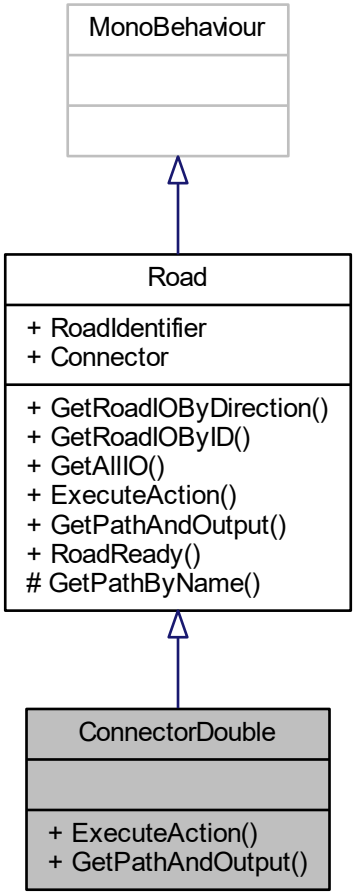
La documentación para esta clase fue generada a partir del siguiente fichero:

- Assets/Scripts/ConditionCards/ConditionCardPicker.cs

5.14. Referencia de la Clase ConnectorDouble

Conector doble.

Diagrama de herencias de ConnectorDouble



Métodos públicos

- override void ExecuteAction (in string[ ] args)

  Ejecuta una acción en base a una lista de argumentos.
- override bool GetPathAndOutput (in RoadInput input, out Path path, out RoadOutput output)

  Dado un input retorna el camino que inicia en él y el output en el que termina.

Otros miembros heredados

5.14.1. Descripción detallada

Conector doble.

5.14.2. Documentación de las funciones miembro

5.14.2.1. ExecuteAction() `override void ConnectorDouble.ExecuteAction ( in string[] args ) [virtual]`

Ejecuta una acción en base a una lista de argumentos.

Parámetros

args	Los argumentos.

Implementa Road.

5.14.2.2. GetPathAndOutput() `override bool ConnectorDouble.GetPathAndOutput ( in RoadInput input, out Path path, out RoadOutput output ) [virtual]`

Dado un input retorna el camino que inicia en él y el output en el que termina.

Parámetros

input	El input RoadInput.
path	El camino Path.
output	El output RoadOutput.

Devuelve

True si hay camino, false si no.

Implementa Road.

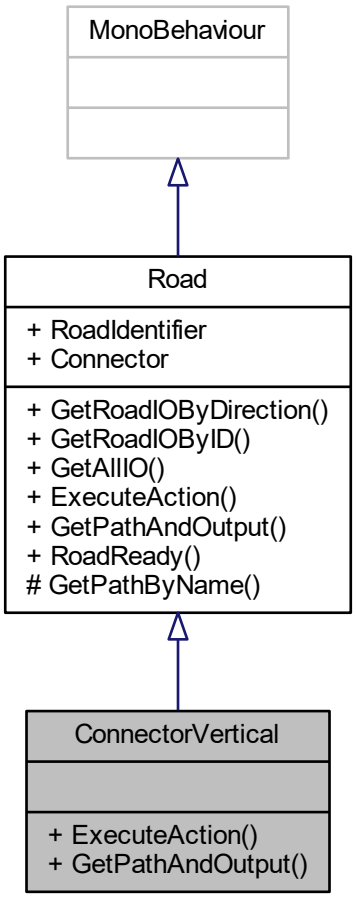
La documentación para esta clase fue generada a partir del siguiente fichero:

- Assets/Scripts/Roads/Roads/ConnectorDouble.cs

5.15. Referencia de la Clase ConnectorVertical

Conector vertical.

Diagrama de herencias de ConnectorVertical



Métodos públicos

- override void ExecuteAction (in string[ ] args)

  Ejecuta una acción en base a una lista de argumentos.
- override bool GetPathAndOutput (in RoadInput input, out Path path, out RoadOutput output)

  Dado un input retorna el camino que inicia en él y el output en el que termina.

Otros miembros heredados

5.15.1. Descripción detallada

Conector vertical.

5.15.2. Documentación de las funciones miembro

5.15.2.1. ExecuteAction()

```
override void ConnectorVertical.ExecuteAction (
    in string[] args )  [virtual]
```

Ejecuta una acción en base a una lista de argumentos.

Parámetros

args	Los argumentos.

Implementa Road.

5.15.2.2. GetPathAndOutput()

```
override bool ConnectorVertical.GetPathAndOutput (
    in RoadInput input,
    out Path path,
    out RoadOutput output )  [virtual]
```

Dado un input retorna el camino que inicia en él y el output en el que termina.

Parámetros

input	El input RoadInput.
path	El camino Path.
output	El output RoadOutput.

Devuelve

True si hay camino, false si no.

Implementa Road.

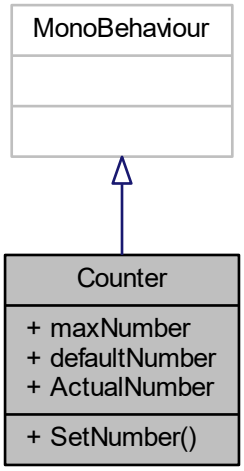
La documentación para esta clase fue generada a partir del siguiente fichero:

- Assets/Scripts/Roads/Roads/ConnectorVertical.cs

5.16. Referencia de la Clase Counter

Contador para el número de instrucciones restantes.

Diagrama de herencias de Counter



Métodos públicos

- int SetNumber (in int number)

  Pone un número en el contador. Si se pasa pone el número máximo y si es menor que cero pone el cero.

Atributos públicos

- int maxNumber = 9

  Número máximo.
- int defaultNumber = 0

  Número por defecto.

Propiedades

- int ActualNumber `[get]`

  Retorna el número actual.

5.16.1. Descripción detallada

Contador para el número de instrucciones restantes.

5.16.2. Documentación de las funciones miembro

5.16.2.1. SetNumber() `int Counter.SetNumber ( in int number )`

Pone un número en el contador. Si se pasa pone el número máximo y si es menor que cero pone el cero.

Parámetros

number	El número a poner.

Devuelve

El número que se ha puesto realmente.

5.16.3. Documentación de los datos miembro

5.16.3.1. defaultNumber `int Counter.defaultNumber = 0`

Número por defecto.

5.16.3.2. maxNumber `int Counter.maxNumber = 9`

Número máximo.

5.16.4. Documentación de propiedades

5.16.4.1. ActualNumber `int Counter.ActualNumber [get]`

Retorna el número actual.

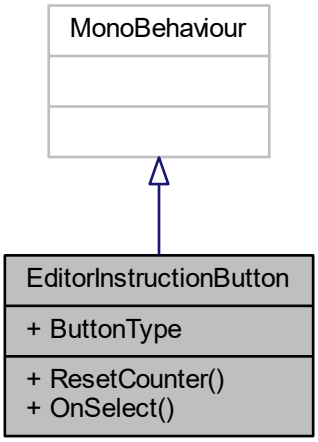
La documentación para esta clase fue generada a partir del siguiente fichero:

- Assets/Scripts/Visual/Counter.cs

5.17. Referencia de la Clase EditorInstructionButton

Clase usada en los botones de instrucciones del editor de niveles.

Diagrama de herencias de EditorInstructionButton



Métodos públicos

- void ResetCounter (MsgEditorResetAllCounters msg)

  Pone el contador a cero.
- void OnSelect ()

  Ejecuta la acción del botón cuando el usuario hace tap.

Propiedades

- Buttons ButtonType `[get]`

  Retorna el tipo de botón.

5.17.1. Descripción detallada

Clase usada en los botones de instrucciones del editor de niveles.

5.17.2. Documentación de las funciones miembro

5.17.2.1. OnSelect() `void EditorInstructionButton.OnSelect ( )`

Ejecuta la acción del botón cuando el usuario hace tap.

5.17.2.2. ResetCounter() `void EditorInstructionButton.ResetCounter ( MsgEditorResetAllCounters msg )`

Pone el contador a cero.

Parámetros

msg	El mensaje MsgEditorResetAllCounters.

5.17.3. Documentación de propiedades

5.17.3.1. ButtonType `Buttons EditorInstructionButton.ButtonType [get]`

Retorna el tipo de botón.

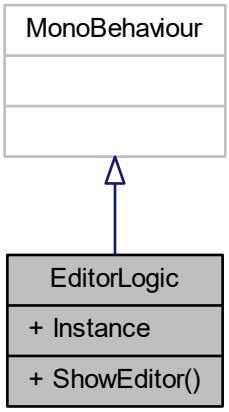
La documentación para esta clase fue generada a partir del siguiente fichero:

- Assets/Scripts/LevelEditor/EditorInstructionButton.cs

5.18. Referencia de la Clase EditorLogic

Clase que contiene toda la lógica del editor de niveles EditorLogic.

Diagrama de herencias de EditorLogic



Tipos públicos

- enum EditorToolType { **Item**, **Block**, **Player**, **Eraser** }

  Enum con la clase de herramientas que podemos usar.

Métodos públicos

- void ShowEditor ()

  Método para mostrar el editor.

Propiedades

- static EditorLogic Instance `[get]`

  Devuelve la instancia de la clase.

5.18.1. Descripción detallada

Clase que contiene toda la lógica del editor de niveles EditorLogic.

5.18.2. Documentación de las enumeraciones miembro de la clase

5.18.2.1. EditorToolType `enum EditorLogic.EditorToolType [strong]`

Enum con la clase de herramientas que podemos usar.

5.18.3. Documentación de las funciones miembro

5.18.3.1. ShowEditor() `void EditorLogic.ShowEditor ( )`

Método para mostrar el editor.

5.18.4. Documentación de propiedades

5.18.4.1. Instance `EditorLogic EditorLogic.Instance [static], [get]`

Devuelve la instancia de la clase.

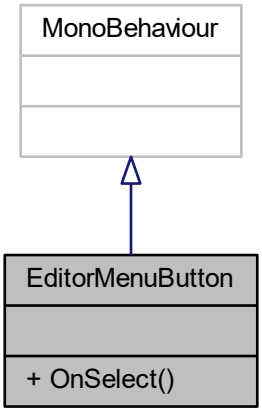
La documentación para esta clase fue generada a partir del siguiente fichero:

- Assets/Scripts/GameLogic/EditorLogic.cs

5.19. Referencia de la Clase EditorMenuButton

Manda un mensaje MsgEditorMenu cuando el editor hace tap sobre una instancia de esta clase.

Diagrama de herencias de EditorMenuButton



Métodos públicos

- void OnSelect ()

  Manda un mensaje MsgEditorMenu cuando el editor hace tap sobre una instancia de esta clase.

5.19.1. Descripción detallada

Manda un mensaje MsgEditorMenu cuando el editor hace tap sobre una instancia de esta clase.

5.19.2. Documentación de las funciones miembro

5.19.2.1. OnSelect() `void EditorMenuButton.OnSelect ( )`

Manda un mensaje MsgEditorMenu cuando el editor hace tap sobre una instancia de esta clase.

La documentación para esta clase fue generada a partir del siguiente fichero:

- Assets/Scripts/LevelEditor/EditorMenuButton.cs

5.20. Referencia de la Clase EditorObject

Las instancias de esta forman los objetos que guarda el editor de niveles.

Métodos públicos

- EditorObject (GameObject associatedGameobject, EditorToolType objectType, int objectIdentifier)

  Constructor de la clase.

Propiedades

- GameObject AssociatedGameobject `[get, set]`

  Retorna o cambia el GameObject asociado.
- EditorToolType ObjectType `[get, set]`

  Retorna o cambia el tipo de objeto.
- int ObjectIdentifier `[get, set]`

  Retorna o cambia el identificador del objeto.

5.20.1. Descripción detallada

Las instancias de esta forman los objetos que guarda el editor de niveles.

5.20.2. Documentación del constructor y destructor

5.20.2.1. EditorObject() `EditorObject.EditorObject (`

```
GameObject associatedGameobject,
EditorToolType objectType,
int objectIdentifier )
```

Constructor de la clase.

Parámetros

associatedGameobject	GameObject que tiene asociado.
objectType	Tipo de objeto (Item, Block, Player).
objectIdentifier	Parámetro extra.

5.20.3. Documentación de propiedades

5.20.3.1. AssociatedGameobject `GameObject EditorObject.AssociatedGameobject [get], [set]`

Retorna o cambia el GameObject asociado.

5.20.3.2. ObjectIdentifier `int EditorObject.ObjectIdentifier [get], [set]`

Retorna o cambia el identificador del objeto.

5.20.3.3. ObjectType `EditorToolType EditorObject.ObjectType [get], [set]`

Retorna o cambia el tipo de objeto.

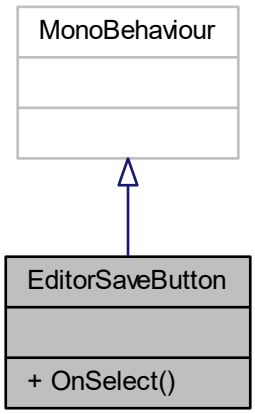
La documentación para esta clase fue generada a partir del siguiente fichero:

- Assets/Scripts/LevelEditor/EditorObject.cs

5.21. Referencia de la Clase EditorSaveButton

Manda un mensaje MsgEditorSaveMap cuando el editor hace tap sobre una instancia de esta clase.

Diagrama de herencias de EditorSaveButton



Métodos públicos

- void OnSelect ()

  Manda un mensaje MsgEditorSaveMap cuando el editor hace tap sobre una instancia de esta clase.

5.21.1. Descripción detallada

Manda un mensaje MsgEditorSaveMap cuando el editor hace tap sobre una instancia de esta clase.

5.21.2. Documentación de las funciones miembro

5.21.2.1. OnSelect() `void EditorSaveButton.OnSelect ( )`

Manda un mensaje MsgEditorSaveMap cuando el editor hace tap sobre una instancia de esta clase.

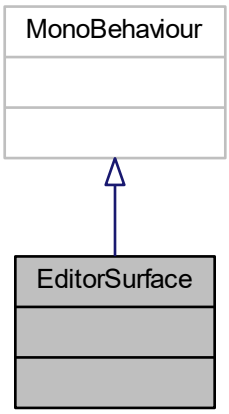
La documentación para esta clase fue generada a partir del siguiente fichero:

- Assets/Scripts/LevelEditor/EditorSaveButton.cs

5.22. Referencia de la Clase EditorSurface

Las instancias de esta clase se usan para generar una superficie en la que el usuario puede poner bloques y objetos.

Diagrama de herencias de EditorSurface



5.22.1. Descripción detallada

Las instancias de esta clase se usan para generar una superficie en la que el usuario puede poner bloques y objetos.

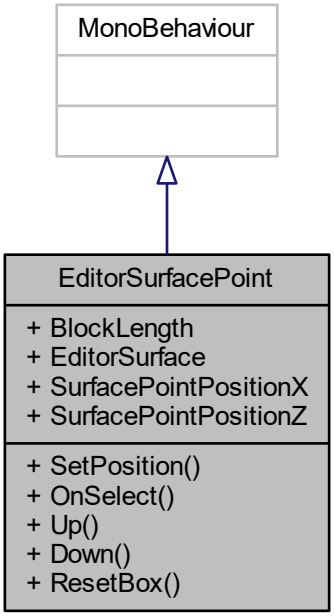
La documentación para esta clase fue generada a partir del siguiente fichero:

- Assets/Scripts/LevelEditor/EditorSurface.cs

5.23. Referencia de la Clase EditorSurfacePoint

La clase EditorSurfacePoint representa al objeto físico en el que los usuarios harán click al tocar la superficie del editor de niveles.

Diagrama de herencias de EditorSurfacePoint



Métodos públicos

- void SetPosition (int x, int z)

  Indica a qué posición x,z corresponde este objeto.
- void OnSelect ()

  Llamado cuando el usuario hace tap.
- void Up ()

  Sube la colisión.
- void Down ()

  Baja la colisión.
- void ResetBox ()

  Resetea la colisión.

Propiedades

- float BlockLength `[get, set]`

  Retorna la longitud de los bloques.
- Transform EditorSurface `[get, set]`

  Retorna la transformación de la superficie del editor.
- int SurfacePointPositionX `[get]`

  Retorna la posición X.
- int SurfacePointPositionZ `[get]`

  Retorna la posición Z.

5.23.1. Descripción detallada

La clase EditorSurfacePoint representa al objeto físico en el que los usuarios harán click al tocar la superficie del editor de niveles.

5.23.2. Documentación de las funciones miembro

5.23.2.1. Down() `void EditorSurfacePoint.Down ( )`

Baja la colisión.

5.23.2.2. OnSelect() `void EditorSurfacePoint.OnSelect ( )`

Llamado cuando el usuario hace tap.

5.23.2.3. ResetBox() `void EditorSurfacePoint.ResetBox ( )`

Resetea la colisión.

5.23.2.4. SetPosition() `void EditorSurfacePoint.SetPosition (`

```
int x,
int z )
```

Indica a qué posición x,z corresponde este objeto.

Parámetros

x	Coordenada x.
z	Coordenada z.

5.23.2.5. Up() `void EditorSurfacePoint.Up ( )`

Sube la colisión.

5.23.3. Documentación de propiedades

5.23.3.1. BlockLength `float EditorSurfacePoint.BlockLength [get], [set]`

Retorna la longitud de los bloques.

5.23.3.2. EditorSurface `Transform EditorSurfacePoint.EditorSurface [get], [set]`

Retorna la transformación de la superficie del editor.

5.23.3.3. SurfacePointPositionX `int EditorSurfacePoint.SurfacePointPositionX [get]`

Retorna la posición X.

5.23.3.4. SurfacePointPositionZ `int EditorSurfacePoint.SurfacePointPositionZ [get]`

Retorna la posición Z.

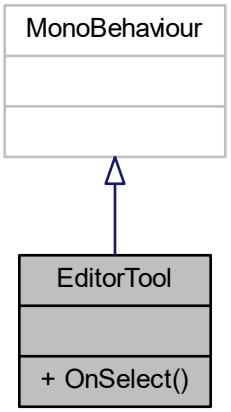
La documentación para esta clase fue generada a partir del siguiente fichero:

- Assets/Scripts/LevelEditor/EditorSurfacePoint.cs

5.24. Referencia de la Clase EditorTool

Esta clase contiene el tipo de herramienta del editor e informa que se ha seleccionado cuando se hace click.

Diagrama de herencias de EditorTool



Métodos públicos

- void OnSelect ()

  Manda un mensaje MsgEditorToolSelected al hacer tap en el objeto.

5.24.1. Descripción detallada

Esta clase contiene el tipo de herramienta del editor e informa que se ha seleccionado cuando se hace click.

5.24.2. Documentación de las funciones miembro

5.24.2.1. OnSelect() `void EditorTool.OnSelect ( )`

Manda un mensaje MsgEditorToolSelected al hacer tap en el objeto.

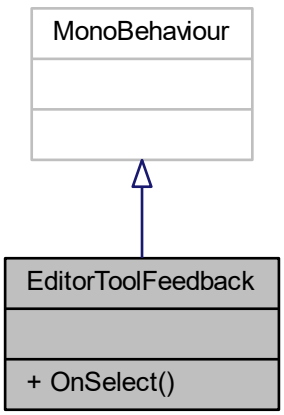
La documentación para esta clase fue generada a partir del siguiente fichero:

- Assets/Scripts/LevelEditor/EditorTool.cs

5.25. Referencia de la Clase EditorToolFeedback

Ejecuta una pequeña animación cuando se hace tap en un objeto.

Diagrama de herencias de EditorToolFeedback



Métodos públicos

- void OnSelect ()

  Ejecuta la animación cuando el usuario hace tap.

5.25.1. Descripción detallada

Ejecuta una pequeña animación cuando se hace tap en un objeto.

5.25.2. Documentación de las funciones miembro

5.25.2.1. OnSelect() `void EditorToolFeedback.OnSelect ( )`

Ejecuta la animación cuando el usuario hace tap.

La documentación para esta clase fue generada a partir del siguiente fichero:

- Assets/Scripts/LevelEditor/EditorToolFeedback.cs

5.26. Referencia de la Clase Block.EffectReaction

La clase EffectReaction modela las reacciones que pueden tener los items sobre este bloque.

Atributos públicos

- Items[ ] compatibleItems = new Items[0]

  Items compatibles con este efecto. Si hay 0 compatible items se activara con cualquier item.
- Effects effect

  El efecto en cuestión.
- bool replaceBlock = false

  ¿Hay que cambiar este bloque por otro tras ejecutar el efecto?
- Blocks block

  Por qué bloque se cambiará.
- BlockActions[ ] actionsToExecute = new BlockActions[0]

  Acciones a ejecutar durante el efecto.
- BlockProperties[ ] newProperties = new BlockProperties[0]

  Nuevas propiedades del bloque.
- string[ ] animationTriggers = new string[0]

  Triggers de animación que se pueden ejecutar.

5.26.1. Descripción detallada

La clase EffectReaction modela las reacciones que pueden tener los items sobre este bloque.

5.26.2. Documentación de los datos miembro

5.26.2.1. actionsToExecute `BlockActions [] Block.EffectReaction.actionsToExecute = new BlockActions[0]`

Acciones a ejecutar durante el efecto.

5.26.2.2. animationTriggers `string [] Block.EffectReaction.animationTriggers = new string[0]`

Triggers de animación que se pueden ejecutar.

5.26.2.3. block `Blocks Block.EffectReaction.block`

Por qué bloque se cambiará.

5.26.2.4. compatibleItems `Items [] Block.EffectReaction.compatibleItems = new Items[0]`

Items compatibles con este efecto. Si hay 0 compatible items se activara con cualquier item.

5.26.2.5. effect `Effects Block.EffectReaction.effect`

El efecto en cuestión.

5.26.2.6. newProperties `BlockProperties [] Block.EffectReaction.newProperties = new BlockProperties[0]`

Nuevas propiedades del bloque.

5.26.2.7. replaceBlock `bool Block.EffectReaction.replaceBlock = false`

¿Hay que cambiar este bloque por otro tras ejecutar el efecto?

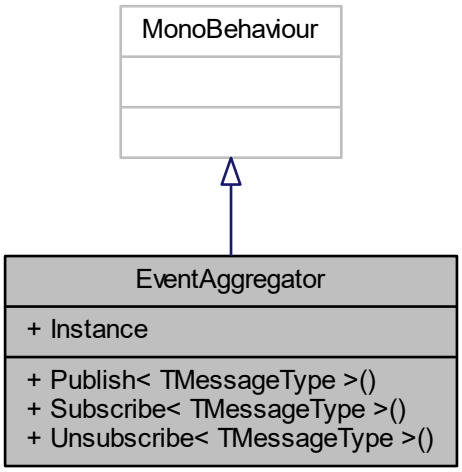
La documentación para esta clase fue generada a partir del siguiente fichero:

- Assets/Scripts/Blocks/Block.cs

5.27. Referencia de la Clase EventAggregator

Clase EventAggregator usada como un punto central en el que los objetos del juego pueden públicar mensajes y suscribirse para recibir los tipos de mensajes que les interesen. Tomado y adaptado de https://www.↩ c-sharpcorner.com/UploadFile/pranayamr/publisher-or-subscriber-pattern-with-event-or-del

Diagrama de herencias de EventAggregator



Métodos públicos

- void Publish< TMessageType > (TMessageType message)

  Método para públicar mensajes
- Subscription< TMessageType > Subscribe< TMessageType > (Action< TMessageType > action)

  Procedimiento para suscribirse a los mensajes deseados.
- void Unsubscribe< TMessageType > (Subscription< TMessageType > subscription)

  Metodo para desuscribirse.

Propiedades

- static EventAggregator Instance `[get]`

  Getter de la instancia de la clase.

5.27.1. Descripción detallada

Clase EventAggregator usada como un punto central en el que los objetos del juego pueden públicar mensajes y suscribirse para recibir los tipos de mensajes que les interesen. Tomado y adaptado de https://www.↩ c-sharpcorner.com/UploadFile/pranayamr/publisher-or-subscriber-pattern-with-event-or-del

5.27.2. Documentación de las funciones miembro

5.27.2.1. Publish< TMessageType >() `void EventAggregator.Publish< TMessageType > ( TMessageType message )`

Método para públicar mensajes

Parámetros del template

TMessageType	.

Parámetros

message	El mensaje a publicarTMessageType.

5.27.2.2. Subscribe< TMessageType >() `Subscription<TMessageType> EventAggregator.Subscribe< TMessageType > ( Action< TMessageType > action )`

Procedimiento para suscribirse a los mensajes deseados.

Parámetros del template

TMessageType	.

Parámetros

action	Metodo al que se va a llamar cuando se reciba un mensaje Action<TMessageType>.

Devuelve

Token de suscripcion Subscription<TMessageType>.

5.27.2.3. Unsubscribe< TMessageType >() `void EventAggregator.Unsubscribe< TMessageType > ( Subscription< TMessageType > subscription )`

Metodo para desuscribirse.

Parámetros del template

TMessageType	.

Parámetros

subscription	La suscripción Subscription<TMessageType>.

5.27.3. Documentación de propiedades

5.27.3.1. Instance `EventAggregator EventAggregator.Instance [static], [get]`

Getter de la instancia de la clase.

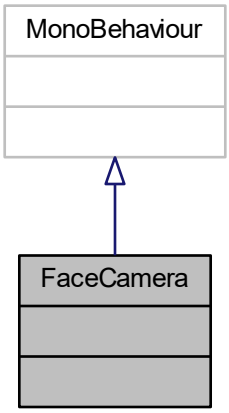
La documentación para esta clase fue generada a partir del siguiente fichero:

- Assets/Scripts/MessageHub/EventAggregator.cs

5.28. Referencia de la Clase FaceCamera

Hace que el objeto que lo tiene mire siempre a la cámara principal.

Diagrama de herencias de FaceCamera



5.28.1. Descripción detallada

Hace que el objeto que lo tiene mire siempre a la cámara principal.

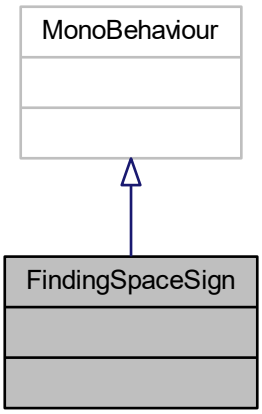
La documentación para esta clase fue generada a partir del siguiente fichero:

- Assets/Scripts/Visual/FaceCamera.cs

5.29. Referencia de la Clase FindingSpaceSign

Cartel que indica que se está buscando el espacio del jugador.

Diagrama de herencias de FindingSpaceSign



5.29.1. Descripción detallada

Cartel que indica que se está buscando el espacio del jugador.

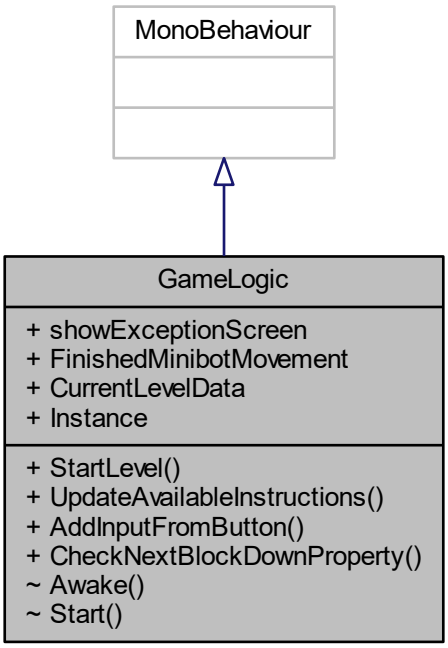
La documentación para esta clase fue generada a partir del siguiente fichero:

- Assets/Scripts/Visual/FindingSpaceSign.cs

5.30. Referencia de la Clase GameLogic

La clase GameLogic contiene la lógica del robot que se mueve por el mapa y determina cuando se gana o se pierde la partida, etc.

Diagrama de herencias de GameLogic



Métodos públicos

- void StartLevel (LevelData levelData, GameObject mapParent)

  Da comienzo a un nivel del juego.
- void UpdateAvailableInstructions ()

  Publica un mensaje para que los contadores de las instrucciones restantes actualicen su número.
- void AddInputFromButton (Buttons button)

  Ejecuta las acciones apropiadas en respuesta a la pulsación de un botón.
- bool CheckNextBlockDownProperty (BlockProperties property)

  Comprueba si el bloque de enfrente y abajo cumple una propiedad.

Atributos públicos

- bool showExceptionScreen = false

  Determina si se debe mostrar la pantalla de excepción cuando el jugador pierde el nivel.

Propiedades

- bool FinishedMinibotMovement `[get, set]`

  Determina si el robot de las carreteras ha acabado de moverse.
- LevelData CurrentLevelData `[get, set]`

  Estructura de datos que contiene el nivel actual.
- static GameLogic Instance `[get]`

  Retorna la instancia de la clase.

5.30.1. Descripción detallada

La clase GameLogic contiene la lógica del robot que se mueve por el mapa y determina cuando se gana o se pierde la partida, etc.

5.30.2. Documentación de las funciones miembro

5.30.2.1. AddInputFromButton() `void GameLogic.AddInputFromButton ( Buttons button )`

Ejecuta las acciones apropiadas en respuesta a la pulsación de un botón.

Parámetros

button	El botón pulsado Buttons.

5.30.2.2. CheckNextBlockDownProperty() `bool GameLogic.CheckNextBlockDownProperty ( BlockProperties property )`

Comprueba si el bloque de enfrente y abajo cumple una propiedad.

Parámetros

property	La propiedadBlockProperties.

Devuelve

True si la cumple, false si no bool.

5.30.2.3. StartLevel() `void GameLogic.StartLevel ( LevelData levelData, GameObject mapParent )`

Da comienzo a un nivel del juego.

Parámetros

levelData	Los datos del nivelLevelData.
mapParent	El padre de los objetos del nivelGameObject.

5.30.2.4. UpdateAvailableInstructions() `void GameLogic.UpdateAvailableInstructions ( )`

Publica un mensaje para que los contadores de las instrucciones restantes actualicen su número.

5.30.3. Documentación de los datos miembro

5.30.3.1. showExceptionScreen `bool GameLogic.showExceptionScreen = false`

Determina si se debe mostrar la pantalla de excepción cuando el jugador pierde el nivel.

5.30.4. Documentación de propiedades

5.30.4.1. CurrentLevelData `LevelData GameLogic.CurrentLevelData [get], [set]`

Estructura de datos que contiene el nivel actual.

5.30.4.2. FinishedMinibotMovement `bool GameLogic.FinishedMinibotMovement [get], [set]`

Determina si el robot de las carreteras ha acabado de moverse.

5.30.4.3. Instance `GameLogic GameLogic.Instance [static], [get]`

Retorna la instancia de la clase.

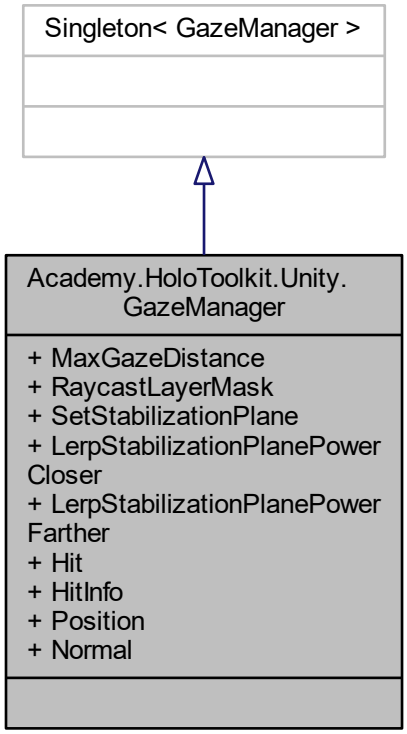
La documentación para esta clase fue generada a partir del siguiente fichero:

- Assets/Scripts/GameLogic/GameLogic.cs

5.31. Referencia de la Clase Academy.HoloToolkit.Unity.GazeManager

GazeManager determines the location of the user's gaze, hit position and normals.

Diagrama de herencias de Academy.HoloToolkit.Unity.GazeManager



Atributos públicos

- float **MaxGazeDistance** = 15.0f
- LayerMask **RaycastLayerMask** = Physics.DefaultRaycastLayers
- bool **SetStabilizationPlane** = true
- float **LerpStabilizationPlanePowerCloser** = 4.0f
- float **LerpStabilizationPlanePowerFarther** = 7.0f

Propiedades

- bool Hit `[get]`

  Physics.Raycast result is true if it hits a hologram.
- RaycastHit HitInfo `[get]`

  HitInfo property gives access to RaycastHit public members.
- Vector3 Position `[get]`

  Position of the intersection of the user's gaze and the holograms in the scene.
- Vector3 Normal `[get]`

  RaycastHit Normal direction.

5.31.1. Descripción detallada

GazeManager determines the location of the user's gaze, hit position and normals.

5.31.2. Documentación de propiedades

5.31.2.1. Hit `bool Academy.HoloToolkit.Unity.GazeManager.Hit [get]`

Physics.Raycast result is true if it hits a hologram.

5.31.2.2. HitInfo `RaycastHit Academy.HoloToolkit.Unity.GazeManager.HitInfo [get]`

HitInfo property gives access to RaycastHit public members.

5.31.2.3. Normal `Vector3 Academy.HoloToolkit.Unity.GazeManager.Normal [get]`

RaycastHit Normal direction.

5.31.2.4. Position `Vector3 Academy.HoloToolkit.Unity.GazeManager.Position [get]`

Position of the intersection of the user's gaze and the holograms in the scene.

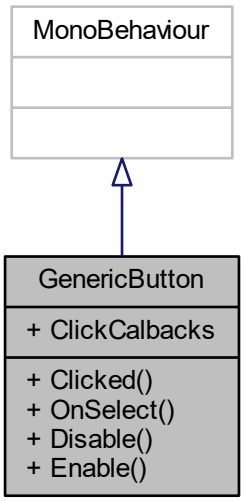
La documentación para esta clase fue generada a partir del siguiente fichero:

- Assets/HoloToolkit-SpatialMapping-230/Input/Scripts/GazeManager.cs

5.32. Referencia de la Clase GenericButton

Representa un botón genérico.

Diagrama de herencias de GenericButton



Métodos públicos

- delegate void Clicked ()

  Declaración del delegado que se llamará cuando se haga tap.
- void OnSelect ()

  OnSelect.
- void Disable ()

  Desactiva el botón.
- void Enable ()

  Activa el botón.

Propiedades

- Clicked ClickCalbacks `[get, set]`

  Añade un callback.

5.32.1. Descripción detallada

Representa un botón genérico.

5.32.2. Documentación de las funciones miembro

5.32.2.1. Clicked() `delegate void GenericButton.Clicked ( )`

Declaración del delegado que se llamará cuando se haga tap.

5.32.2.2. Disable() `void GenericButton.Disable ( )`

Desactiva el botón.

5.32.2.3. Enable() `void GenericButton.Enable ( )`

Activa el botón.

5.32.2.4. OnSelect() `void GenericButton.OnSelect ( )`

OnSelect.

5.32.3. Documentación de propiedades

5.32.3.1. ClickCalbacks `Clicked GenericButton.ClickCalbacks [get], [set]`

Añade un callback.

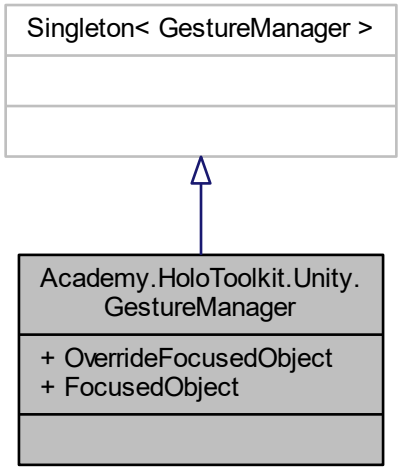
La documentación para esta clase fue generada a partir del siguiente fichero:

- Assets/Scripts/Visual/GenericButton.cs

5.33. Referencia de la Clase Academy.HoloToolkit.Unity.GestureManager

GestureManager creates a gesture recognizer and signs up for a tap gesture. When a tap gesture is detected, GestureManager uses GazeManager to find the game object. GestureManager then sends a message to that game object.

Diagrama de herencias de Academy.HoloToolkit.Unity.GestureManager



Propiedades

- GameObject OverrideFocusedObject `[get, set]`

  To select even when a hologram is not being gazed at, set the override focused object. If its null, then the gazed at object will be selected.
- GameObject FocusedObject `[get]`

  Gets the currently focused object, or null if none.

5.33.1. Descripción detallada

GestureManager creates a gesture recognizer and signs up for a tap gesture. When a tap gesture is detected, GestureManager uses GazeManager to find the game object. GestureManager then sends a message to that game object.

5.33.2. Documentación de propiedades

5.33.2.1. FocusedObject `GameObject Academy.HoloToolkit.Unity.GestureManager.FocusedObject [get]`

Gets the currently focused object, or null if none.

5.33.2.2. OverrideFocusedObject `GameObject Academy.HoloToolkit.Unity.GestureManager.OverrideFocusedObject [get], [set]`

To select even when a hologram is not being gazed at, set the override focused object. If its null, then the gazed at object will be selected.

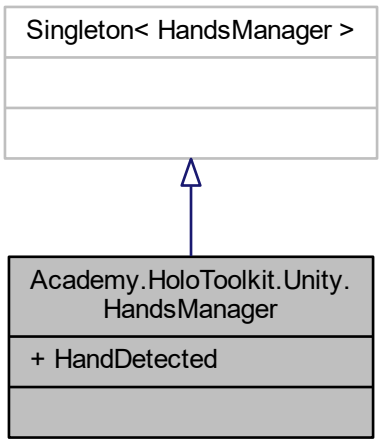
La documentación para esta clase fue generada a partir del siguiente fichero:

- Assets/HoloToolkit-SpatialMapping-230/Input/Scripts/GestureManager.cs

5.34. Referencia de la Clase Academy.HoloToolkit.Unity.HandsManager

HandsDetected determines if the hand is currently detected or not.

Diagrama de herencias de Academy.HoloToolkit.Unity.HandsManager



Propiedades

- bool HandDetected `[get]`

  HandDetected tracks the hand detected state. Returns true if the list of tracked hands is not empty.

5.34.1. Descripción detallada

HandsDetected determines if the hand is currently detected or not.

5.34.2. Documentación de propiedades

5.34.2.1. HandDetected `bool Academy.HoloToolkit.Unity.HandsManager.HandDetected [get]`

HandDetected tracks the hand detected state. Returns true if the list of tracked hands is not empty.

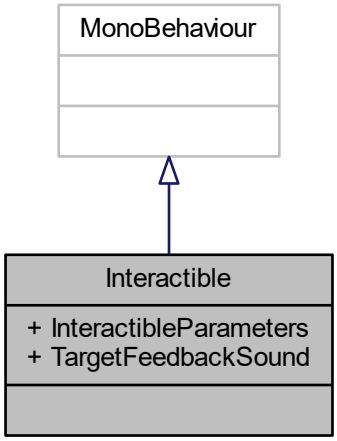
La documentación para esta clase fue generada a partir del siguiente fichero:

- Assets/HoloToolkit-SpatialMapping-230/Input/Scripts/HandsManager.cs

5.35. Referencia de la Clase Interactible

Esta clase marca un objeto de tal forma que se pueda interactuar con el mismo cuando el usuario lo mira.

Diagrama de herencias de Interactible



Atributos públicos

- InteractibleParameters **InteractibleParameters**
- AudioClip **TargetFeedbackSound**

5.35.1. Descripción detallada

Esta clase marca un objeto de tal forma que se pueda interactuar con el mismo cuando el usuario lo mira.

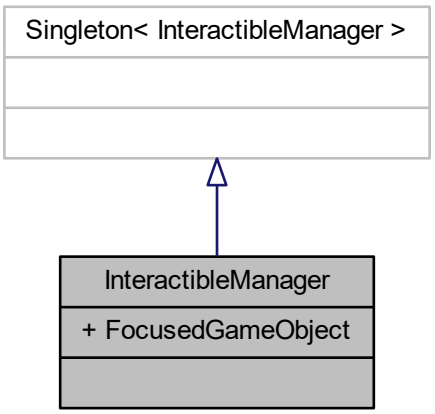
La documentación para esta clase fue generada a partir del siguiente fichero:

- Assets/Scripts/Hololens/Interactible.cs

5.36. Referencia de la Clase InteractibleManager

InteractibleManager contiene el objeto al que el usuario está mirando.

Diagrama de herencias de InteractibleManager



Propiedades

- GameObject **FocusedGameObject** `[get]`

5.36.1. Descripción detallada

InteractibleManager contiene el objeto al que el usuario está mirando.

La documentación para esta clase fue generada a partir del siguiente fichero:

- Assets/Scripts/Hololens/InteractibleManager.cs

5.37. Referencia de la Clase InteractibleParameters

Atributos públicos

- bool **Scrollable** = true
- bool **Placeable** = true

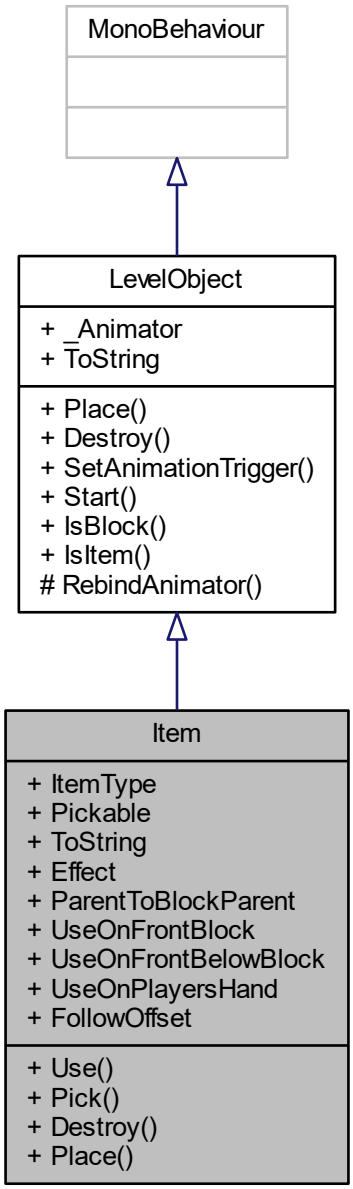
La documentación para esta clase fue generada a partir del siguiente fichero:

- Assets/Scripts/Hololens/Interactible.cs

5.38. Referencia de la Clase Item

Define la clase Item.

Diagrama de herencias de Item



Métodos públicos

- void Use ()

  Acción de usar el item.
- void Pick (Transform transformToFollow, Vector3 followOffset)

Acción de tomar el item.

- override void Destroy ()

  Acción de destruir el bloque.
- override void Place ()

  Acción de colocar el bloque.

Propiedades

- Items ItemType `[get]`

  Retorna el tipo del item de esta instancia.
- bool Pickable `[get]`

  Indica si el item se puede recoger o no.
- override string ToString `[get]`

  ToString.
- Effects Effect `[get]`

  Efecto que provoca este item.
- bool ParentToBlockParent `[get]`

  Indica si al usarlo se tiene que convertir en hijo del bloque sobre el que se usa.
- bool UseOnFrontBlock `[get, set]`

  Indica si hay que usarlo en el bloque de enfrente.
- bool UseOnFrontBelowBlock `[get, set]`

  Indica si hay que usarlo en el bloque de enfrente y abajo
- bool UseOnPlayersHand `[get, set]`

  ¿Se tiene que usar en la mano del jugador?
- Vector3 FollowOffset `[get, set]`

  Distancia a la que tiene que seguir al jugador.

Otros miembros heredados

5.38.1. Descripción detallada

Define la clase Item.

5.38.2. Documentación de las funciones miembro

5.38.2.1. Destroy() `override void Item.Destroy ( ) [virtual]`

Acción de destruir el bloque.

Implementa LevelObject.

5.38.2.2. Pick()

```
void Item.Pick (
    Transform transformToFollow,
    Vector3 followOffset )
```

Acción de tomar el item.

Parámetros

transformToFollow	Objetivo que tiene que seguir el item Transform.
followOffset	Distancia a la que seguir al objetivo Vector3.

5.38.2.3. Place() `override void Item.Place ( ) [virtual]`

Acción de colocar el bloque.

Implementa LevelObject.

5.38.2.4. Use() `void Item.Use ( )`

Acción de usar el item.

5.38.3. Documentación de propiedades

5.38.3.1. Effect `Effects Item.Effect [get]`

Efecto que provoca este item.

5.38.3.2. FollowOffset `Vector3 Item.FollowOffset [get], [set]`

Distancia a la que tiene que seguir al jugador.

5.38.3.3. ItemType `Items Item.ItemType [get]`

Retorna el tipo del item de esta instancia.

5.38.3.4. ParentToBlockParent `bool Item.ParentToBlockParent [get]`

Indica si al usarlo se tiene que convertir en hijo del bloque sobre el que se usa.

5.38.3.5. Pickable `bool Item.Pickable [get]`

Indica si el item se puede recoger o no.

5.38.3.6. ToString `override string Item.ToString [get]`

ToString.

5.38.3.7. UseOnFrontBelowBlock `bool Item.UseOnFrontBelowBlock [get], [set]`

Indica si hay que usarlo en el bloque de enfrente y abajo

5.38.3.8. UseOnFrontBlock `bool Item.UseOnFrontBlock [get], [set]`

Indica si hay que usarlo en el bloque de enfrente.

5.38.3.9. UseOnPlayersHand `bool Item.UseOnPlayersHand [get], [set]`

¿Se tiene que usar en la mano del jugador?

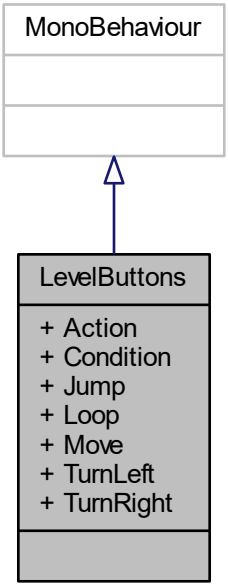
La documentación para esta clase fue generada a partir del siguiente fichero:

- Assets/Scripts/Blocks/Item.cs

5.39. Referencia de la Clase LevelButtons

Manager de los botones de colocar carreteras.

Diagrama de herencias de LevelButtons



Tipos públicos

- enum Buttons {
  Action = 0, **Condition** = 1, **Jump** = 2, **Loop** = 3,
  Move = 4, **Play** = 5, **Restart** = 6, **TurnLeft** = 7,
  TurnRight = 8, **Undo** = 9, **MapMenu** = 10, **Undefined** = 999 }

  Tipos de botones aceptados.

Atributos públicos

- ButtonCounterScript Action

  Contador del botón.
- ButtonCounterScript Condition

  Contador del botón.
- ButtonCounterScript Jump

  Contador del botón.
- ButtonCounterScript Loop

  Contador del botón.
- ButtonCounterScript Move

  Contador del botón.
- ButtonCounterScript TurnLeft

  Contador del botón.
- ButtonCounterScript TurnRight

  Contador del botón.

5.39.1. Descripción detallada

Manager de los botones de colocar carreteras.

5.39.2. Documentación de las enumeraciones miembro de la clase

5.39.2.1. Buttons `enum LevelButtons.Buttons [strong]`

Tipos de botones aceptados.

5.39.3. Documentación de los datos miembro

5.39.3.1. Action `ButtonCounterScript LevelButtons.Action`

Contador del botón.

5.39.3.2. Condition `ButtonCounterScript LevelButtons.Condition`

Contador del botón.

5.39.3.3. Jump `ButtonCounterScript LevelButtons.Jump`

Contador del botón.

5.39.3.4. Loop `ButtonCounterScript LevelButtons.Loop`

Contador del botón.

5.39.3.5. Move `ButtonCounterScript LevelButtons.Move`

Contador del botón.

5.39.3.6. TurnLeft `ButtonCounterScript LevelButtons.TurnLeft`

Contador del botón.

5.39.3.7. TurnRight `ButtonCounterScript LevelButtons.TurnRight`

Contador del botón.

La documentación para esta clase fue generada a partir del siguiente fichero:

- Assets/Scripts/Roads/LevelButtons.cs

5.40. Referencia de la Clase LevelData

La clase LevelData es la estructura que contiene todos los datos de un nivel.

Métodos públicos

- LevelData Clone ()

  Hace una copia profunda del objeto.

Atributos públicos

- string levelName

  Nombre del nivel.
- List< int > levelSize

  Tamaño del nivel.
- List< int > playerPos

  Posición inicial del jugador.
- int playerOrientation

  Orientación inicial del jugador como un entero (0,1,2,3).
- List< int > goal

  Meta del mapa.
- AvailableInstructions availableInstructions

  Guarda el número de instrucciones que le quedan al jugador.
- List< int > mapAndItems

  Los bloques y los items como una lista de ints.

5.40.1. Descripción detallada

La clase LevelData es la estructura que contiene todos los datos de un nivel.

5.40.2. Documentación de las funciones miembro

5.40.2.1. Clone() `LevelData LevelData.Clone ( )`

Hace una copia profunda del objeto.

Devuelve

Retorna un clon del objeto.

5.40.3. Documentación de los datos miembro

5.40.3.1. availableInstructions `AvailableInstructions LevelData.availableInstructions`

Guarda el número de instrucciones que le quedan al jugador.

5.40.3.2. goal `List<int> LevelData.goal`

Meta del mapa.

5.40.3.3. levelName `string LevelData.levelName`

Nombre del nivel.

5.40.3.4. levelSize `List<int> LevelData.levelSize`

Tamaño del nivel.

5.40.3.5. mapAndItems `List<int> LevelData.mapAndItems`

Los bloques y los items como una lista de ints.

5.40.3.6. playerOrientation `int LevelData.playerOrientation`

Orientación inicial del jugador como un entero (0,1,2,3).

5.40.3.7. playerPos `List<int> LevelData.playerPos`

Posición inicial del jugador.

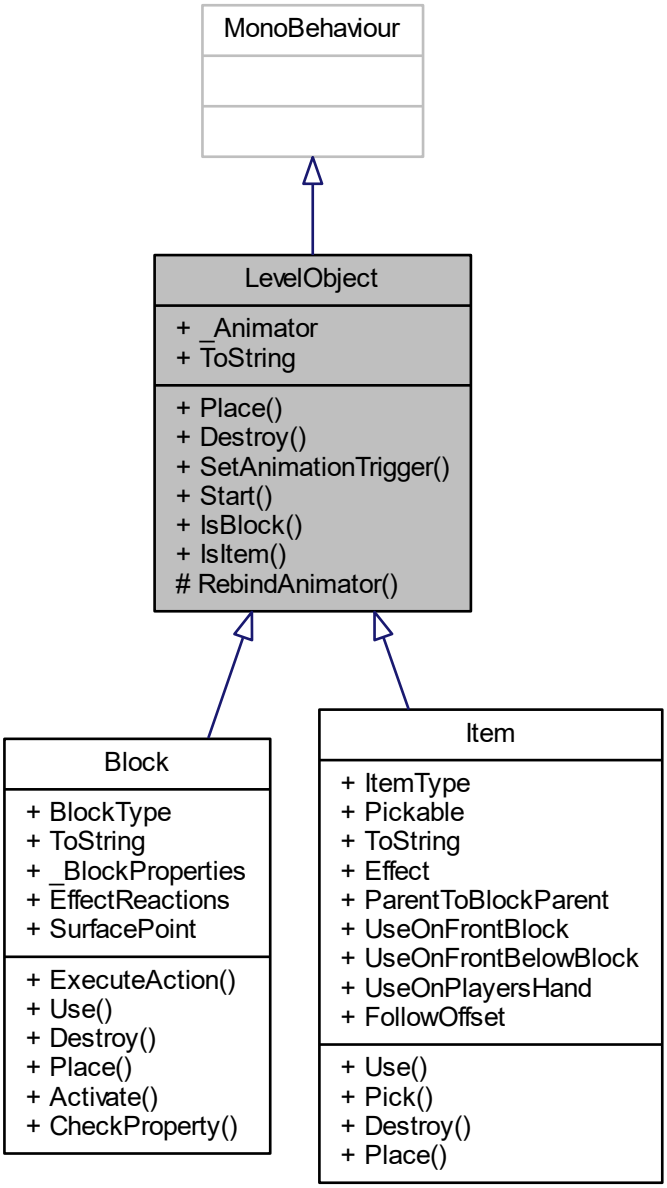
La documentación para esta clase fue generada a partir del siguiente fichero:

- Assets/Scripts/DataStructures/LevelData.cs

5.41. Referencia de la Clase LevelObject

Define la clase LevelObject que son los objetos del juego tales como items y bloques.

Diagrama de herencias de LevelObject



Tipos públicos

- enum Blocks {
  NoBlock = 0, **WaterBlock** = 1, **LavaBlock** = 2, **SolidBlock** = 3,
  LiftBlock = 4, **SpikesBlock** = 5, **IceBlock** = 6 }

  Identificadores de los bloques de juego.
- enum Items {
  PlankItem = 25, **FanItem** = 26, **FlagItem** = 27, **ActivatorItem** = 28,
  BombItem = 29 }

  Identificadores de los items.
- enum Effects { **None**, **Freeze**, **Destroy**, **Activate** }

  Identificadores de los efectos de un item.

Métodos públicos

- abstract void Place ()

  Acción de colocar el bloque.
- abstract void Destroy ()

  Acción de destruir el bloque.
- void SetAnimationTrigger (in string trigger)

  Ejecuta un trigger en el animator del objeto.
- void Start ()

  Start.
- bool IsBlock ()

  ¿Es un bloque?
- bool IsItem ()

  ¿Es un item?

Métodos protegidos

- void RebindAnimator ()

  Resetea el animator.

Propiedades

- Animator _Animator `[get]`

  Retorna el animator.
- abstract new string ToString `[get]`

  ToString.

5.41.1. Descripción detallada

Define la clase LevelObject que son los objetos del juego tales como items y bloques.

5.41.2. Documentación de las enumeraciones miembro de la clase

5.41.2.1. Blocks enum LevelObject.Blocks [strong]

Identificadores de los bloques de juego.

5.41.2.2. Effects enum LevelObject.Effects [strong]

Identificadores de los efectos de un item.

5.41.2.3. Items enum LevelObject.Items [strong]

Identificadores de los items.

5.41.3. Documentación de las funciones miembro

5.41.3.1. Destroy() abstract void LevelObject.Destroy ( ) [pure virtual]

Acción de destruir el bloque.

Implementado en Block y Item.

5.41.3.2. IsBlock() bool LevelObject.IsBlock ( )

¿Es un bloque?

Devuelve

True si lo es, false si no bool.

5.41.3.3. IsItem() bool LevelObject.IsItem ( )

¿Es un item?

Devuelve

True si lo es, false si no bool.

5.41.3.4. Place() `abstract void LevelObject.Place ( ) [pure virtual]`

Acción de colocar el bloque.

Implementado en Block y Item.

5.41.3.5. RebindAnimator() `void LevelObject.RebindAnimator ( ) [protected]`

Resetea el animator.

5.41.3.6. SetAnimationTrigger() `void LevelObject.SetAnimationTrigger ( in string trigger )`

Ejecuta un trigger en el animator del objeto.

Parámetros

trigger	El trigger string.

5.41.3.7. Start() `void LevelObject.Start ( )`

Start.

5.41.4. Documentación de propiedades

5.41.4.1. _Animator `Animator LevelObject._Animator [get]`

Retorna el animator.

5.41.4.2. ToString `abstract new string LevelObject.ToString [get]`

ToString.

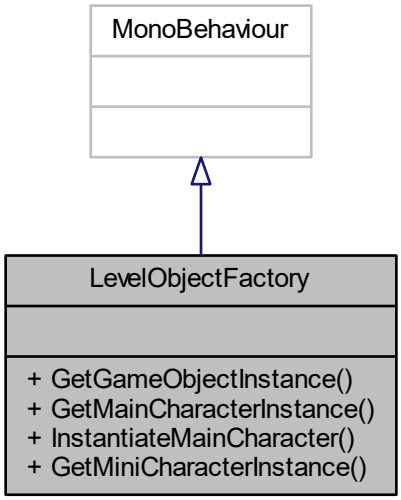
La documentación para esta clase fue generada a partir del siguiente fichero:

- Assets/Scripts/Blocks/LevelObject.cs

5.42. Referencia de la Clase LevelObjectFactory

Define la clase LevelObjectFactory que instancia robots, bloques y objetos.

Diagrama de herencias de LevelObjectFactory



Métodos públicos

- LevelObject GetGameObjectInstance (in int id)

  Genera la instancia de un objeto dado su ID.
- void GetMainCharacterInstance (MsgRenderMainCharacter msg)

  Retorna la instancia del robot principal.
- GameObject InstantiateMainCharacter ()

  Retorna la instancia del robot principal.
- GameObject GetMiniCharacterInstance ()

  Retorna la instancia del robot pequeño.

5.42.1. Descripción detallada

Define la clase LevelObjectFactory que instancia robots, bloques y objetos.

5.42.2. Documentación de las funciones miembro

5.42.2.1. GetGameObjectInstance() `LevelObject LevelObjectFactory.GetGameObjectInstance ( in int id )`

Genera la instancia de un objeto dado su ID.

Parámetros

id	El id int.

Devuelve

El LevelObject generado.

5.42.2.2. GetMainCharacterInstance() `void LevelObjectFactory.GetMainCharacterInstance ( MsgRenderMainCharacter msg )`

Retorna la instancia del robot principal.

Parámetros

msg	El mensaje MsgRenderMainCharacter.

5.42.2.3. GetMiniCharacterInstance() `GameObject LevelObjectFactory.GetMiniCharacterInstance ( )`

Retorna la instancia del robot pequeño.

Devuelve

El GameObject generado.

5.42.2.4. InstantiateMainCharacter() `GameObject LevelObjectFactory.InstantiateMainCharacter ( )`

Retorna la instancia del robot principal.

Devuelve

El GameObject generado.

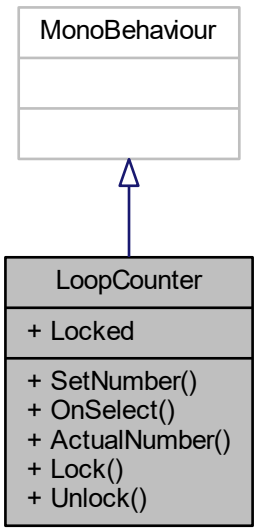
La documentación para esta clase fue generada a partir del siguiente fichero:

- Assets/Scripts/Blocks/LevelObjectFactory.cs

5.43. Referencia de la Clase LoopCounter

Contador del número de iteraciones de un loop.

Diagrama de herencias de LoopCounter



Métodos públicos

- int SetNumber (int number)

  Pone el número que se le indica si es posible, aunque si es menor que cero se pone a cero y si es mayor que el número máximo se pone a ese.
- void OnSelect ()

  OnSelect.
- int ActualNumber ()

  Retorna el número actual.
- void Lock ()

  Bloquea el contador.
- void Unlock ()

  Desbloquea el contador.

Propiedades

- bool Locked `[get]`

  Retorna si está bloqueado.

5.43.1. Descripción detallada

Contador del número de iteraciones de un loop.

5.43.2. Documentación de las funciones miembro

5.43.2.1. ActualNumber() `int LoopCounter.ActualNumber ( )`

Retorna el número actual.

Devuelve

El número actual.

5.43.2.2. Lock() `void LoopCounter.Lock ( )`

Bloquea el contador.

5.43.2.3. OnSelect() `void LoopCounter.OnSelect ( )`

OnSelect.

5.43.2.4. SetNumber() `int LoopCounter.SetNumber ( int number )`

Pone el número que se le indica si es posible, aunque si es menor que cero se pone a cero y si es mayor que el número máximo se pone a ese.

Parámetros

number	El número a poner.

Devuelve

El número que se ha puesto.

5.43.2.5. Unlock() `void LoopCounter.Unlock ( )`

Desbloquea el contador.

5.43.3. Documentación de propiedades

5.43.3.1. Locked `bool LoopCounter.Locked [get]`

Retorna si está bloqueado.

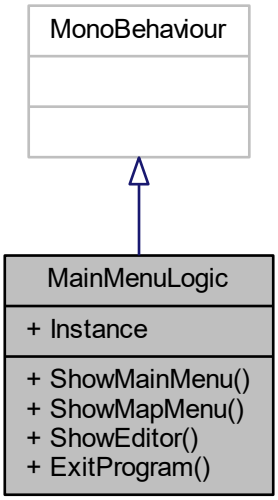
La documentación para esta clase fue generada a partir del siguiente fichero:

- Assets/Scripts/Roads/LoopCounter.cs

5.44. Referencia de la Clase MainMenuLogic

La clase MainMenuLogic contiene la lógica del menú principal.

Diagrama de herencias de MainMenuLogic



Métodos públicos

- void ShowMainMenu ()

  Lanza el menú principal.
- void ShowMapMenu ()

  Lanza el menú de mapas.
- void ShowEditor ()

  Lanza el editor.
- void ExitProgram ()

  Sale del programa.

Propiedades

- static MainMenuLogic Instance `[get]`

  Retorna la instancia de la clase.

5.44.1. Descripción detallada

La clase MainMenuLogic contiene la lógica del menú principal.

5.44.2. Documentación de las funciones miembro

5.44.2.1. ExitProgram() `void MainMenuLogic.ExitProgram ( )`

Sale del programa.

5.44.2.2. ShowEditor() `void MainMenuLogic.ShowEditor ( )`

Lanza el editor.

5.44.2.3. ShowMainMenu() `void MainMenuLogic.ShowMainMenu ( )`

Lanza el menú principal.

5.44.2.4. ShowMapMenu() `void MainMenuLogic.ShowMapMenu ( )`

Lanza el menú de mapas.

5.44.3. Documentación de propiedades

5.44.3.1. Instance `MainMenuLogic MainMenuLogic.Instance [static], [get]`

Retorna la instancia de la clase.

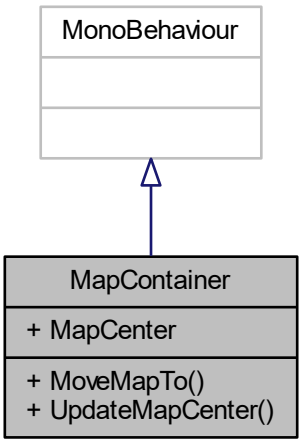
La documentación para esta clase fue generada a partir del siguiente fichero:

- Assets/Scripts/GameLogic/MainMenuLogic.cs

5.45. Referencia de la Clase MapContainer

Contenedor para los niveles del juego con algunas utilidades.

Diagrama de herencias de MapContainer



Métodos públicos

- void MoveMapTo (in Vector3 mapCenterPos, float surfaceY, float blockLength)

  Pone el mapa en el punto requerido y haciendo que quede sobre la coordenada y especificada.
- void UpdateMapCenter (List< int > mapSize, float blockLength)

  Actualiza el punto central del mapa.

Propiedades

- Vector3 MapCenter `[get]`

  Retorna o cambia el centro del mapa.

5.45.1. Descripción detallada

Contenedor para los niveles del juego con algunas utilidades.

5.45.2. Documentación de las funciones miembro

5.45.2.1. MoveMapTo() `void MapContainer.MoveMapTo (`

```
in Vector3 mapCenterPos,
float surfaceY,
float blockLength )
```

Pone el mapa en el punto requerido y haciendo que quede sobre la coordenada y especificada.

Parámetros

mapCenterPos	Donde hay que poner el centro del mapa.
surfaceY	Superficie sobre la que se asienta el mapa.
blockLength	Longitud de los bloques del mapa.

5.45.2.2. UpdateMapCenter() `void MapContainer.UpdateMapCenter (`

```
List< int > mapSize,
float blockLength )
```

Actualiza el punto central del mapa.

Parámetros

mapSize	Tamaño del mapa.
blockLength	Longitud de los bloques del mapa.float.

5.45.3. Documentación de propiedades

5.45.3.1. MapCenter `Vector3 MapContainer.MapCenter [get]`

Retorna o cambia el centro del mapa.

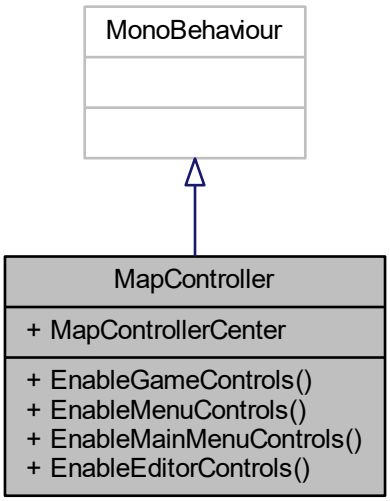
La documentación para esta clase fue generada a partir del siguiente fichero:

- Assets/Scripts/MapMenu/MapContainer.cs

5.46. Referencia de la Clase MapController

Contiene una serie de métodos utiles para activar los controles de las diferentes pantallas del juego.

Diagrama de herencias de MapController



Métodos públicos

- void EnableGameControls ()

  Activa los controles del modo juego.
- void EnableMenuControls ()

  Activa los controles del menú de mapas.
- void EnableMainMenuControls ()

  Activa los controles de menú principal.
- void EnableEditorControls ()

  Activa los controles del editor.

Propiedades

- Vector3 MapControllerCenter `[get]`

  Retorna el punto donde deben aparecer los mapas.

5.46.1. Descripción detallada

Contiene una serie de métodos utiles para activar los controles de las diferentes pantallas del juego.

5.46.2. Documentación de las funciones miembro

5.46.2.1. EnableEditorControls() `void MapController.EnableEditorControls ( )`

Activa los controles del editor.

5.46.2.2. EnableGameControls() `void MapController.EnableGameControls ( )`

Activa los controles del modo juego.

5.46.2.3. EnableMainMenuControls() `void MapController.EnableMainMenuControls ( )`

Activa los controles de menú principal.

5.46.2.4. EnableMenuControls() `void MapController.EnableMenuControls ( )`

Activa los controles del menú de mapas.

5.46.3. Documentación de propiedades

5.46.3.1. MapControllerCenter `Vector3 MapController.MapControllerCenter [get]`

Retorna el punto donde deben aparecer los mapas.

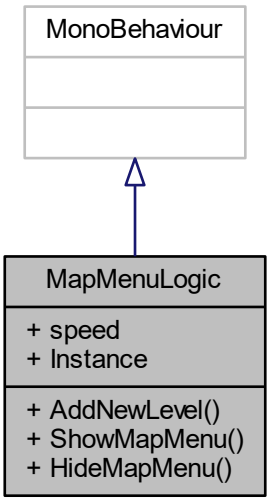
La documentación para esta clase fue generada a partir del siguiente fichero:

- Assets/Scripts/MapMenu/MapController.cs

5.47. Referencia de la Clase MapMenuLogic

Esta clase MapMenuLogic contiene la lógica del menú de mapas del juego.

Diagrama de herencias de MapMenuLogic



Métodos públicos

- void AddNewLevel (LevelData newLevel)

  Añade un nuevo nivel a la lista.
- void ShowMapMenu ()

  Muestra el menú de niveles.
- void HideMapMenu ()

  Esconde el menú de niveles.

Atributos públicos

- float speed = 1f

  Velocidad a la que se ejecutan las acciones.

Propiedades

- static MapMenuLogic Instance `[get]`

  Retorna la instancia de la clase.

5.47.1. Descripción detallada

Esta clase MapMenuLogic contiene la lógica del menú de mapas del juego.

5.47.2. Documentación de las funciones miembro

5.47.2.1. AddNewLevel() `void MapMenuLogic.AddNewLevel ( LevelData newLevel )`

Añade un nuevo nivel a la lista.

Parámetros

newLevel	The newLevelLevelData.

5.47.2.2. HideMapMenu() `void MapMenuLogic.HideMapMenu ( )`

Esconde el menú de niveles.

5.47.2.3. ShowMapMenu() `void MapMenuLogic.ShowMapMenu ( )`

Muestra el menú de niveles.

5.47.3. Documentación de los datos miembro

5.47.3.1. speed `float MapMenuLogic.speed = 1f`

Velocidad a la que se ejecutan las acciones.

5.47.4. Documentación de propiedades

5.47.4.1. Instance `MapMenuLogic MapMenuLogic.Instance [static], [get]`

Retorna la instancia de la clase.

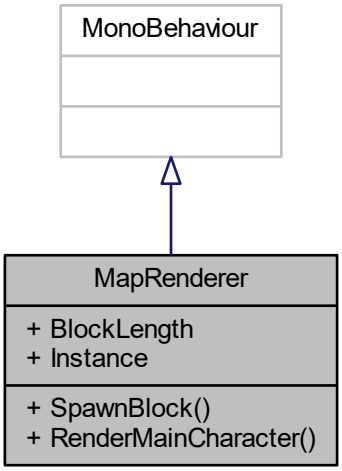
La documentación para esta clase fue generada a partir del siguiente fichero:

- Assets/Scripts/GameLogic/MapMenuLogic.cs

5.48. Referencia de la Clase MapRenderer

La clase MapRenderer genera el mapa a partir de los datos que se le pasen.

Diagrama de herencias de MapRenderer



Métodos públicos

- LevelObject SpawnBlock (int blockToSpawn)

  Genera un bloque por su id.
- GameObject RenderMainCharacter ()

  Genera el robot principal.

Propiedades

- float BlockLength `[get]`

  Retorna la longitud del bloque.
- static MapRenderer Instance `[get]`

  Retorna la instancia de la clase.

5.48.1. Descripción detallada

La clase MapRenderer genera el mapa a partir de los datos que se le pasen.

5.48.2. Documentación de las funciones miembro

5.48.2.1. RenderMainCharacter() `GameObject MapRenderer.RenderMainCharacter ( )`

Genera el robot principal.

Devuelve

El GameObject generado.

5.48.2.2. SpawnBlock() `LevelObject MapRenderer.SpawnBlock ( int blockToSpawn )`

Genera un bloque por su id.

Parámetros

blockToSpawn	El id del bloque int.

Devuelve

El LevelObject generado.

5.48.3. Documentación de propiedades

5.48.3.1. BlockLength `float MapRenderer.BlockLength [get]`

Retorna la longitud del bloque.

5.48.3.2. Instance `MapRenderer MapRenderer.Instance [static], [get]`

Retorna la instancia de la clase.

La documentación para esta clase fue generada a partir del siguiente fichero:

- Assets/Scripts/Blocks/MapRenderer.cs

5.49. Referencia de la Estructura Academy.HoloToolkit.Unity.PlaneFinding.MeshData

PlaneFinding is an expensive task that should not be run from Unity's main thread as it will stall the thread and cause a frame rate dip. Instead, the PlaneFinding APIs should be exclusively called from background threads. Unfortunately, Unity's built-in data types (such as MeshFilter) are not thread safe and cannot be accessed from background threads. The MeshData struct exists to work-around this limitation. When you want to find planes in a collection of MeshFilter objects, start by constructing a list of MeshData structs from those MeshFilters. You can then take the resulting list of MeshData structs, and safely pass it to the FindPlanes() API from a background thread.

Métodos públicos

- **MeshData** (MeshFilter meshFilter)

Atributos públicos

- Matrix4x4 **Transform**
- Vector3[ ] **Verts**
- Vector3[ ] **Normals**
- Int32[ ] **Indices**

5.49.1. Descripción detallada

PlaneFinding is an expensive task that should not be run from Unity's main thread as it will stall the thread and cause a frame rate dip. Instead, the PlaneFinding APIs should be exclusively called from background threads. Unfortunately, Unity's built-in data types (such as MeshFilter) are not thread safe and cannot be accessed from background threads. The MeshData struct exists to work-around this limitation. When you want to find planes in a collection of MeshFilter objects, start by constructing a list of MeshData structs from those MeshFilters. You can then take the resulting list of MeshData structs, and safely pass it to the FindPlanes() API from a background thread.

La documentación para esta estructura fue generada a partir del siguiente fichero:

- Assets/HoloToolkit-SpatialMapping-230/SpatialMapping/Scripts/PlaneFinding.cs

5.50. Referencia de la Estructura Academy.HoloToolkit.Unity.PlaneFinding.DLLImports.MeshData

Atributos públicos

- Matrix4x4 **transform**
- Int32 **vertCount**
- Int32 **indexCount**
- IntPtr **verts**
- IntPtr **normals**
- IntPtr **indices**

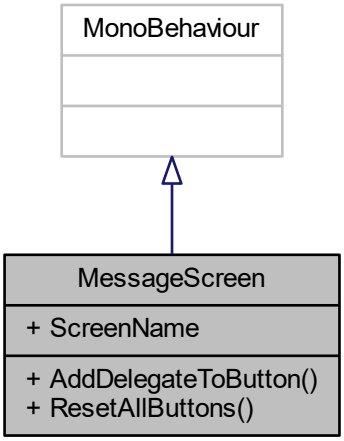
La documentación para esta estructura fue generada a partir del siguiente fichero:

- Assets/HoloToolkit-SpatialMapping-230/SpatialMapping/Scripts/PlaneFinding.cs

5.51. Referencia de la Clase MessageScreen

La clase MessageScreen representa una pantalla para mostrar mensajes al usuario.

Diagrama de herencias de MessageScreen



Métodos públicos

- void AddDelegateToButton (string bType, OnMessageScreenButtonPressed bDelegate)

  Hace que el botón con nombre bType llame al delegado bDelegate cuando se hace click en él.
- void ResetAllButtons ()

  Resetea los delegados de los botones.

Propiedades

- string ScreenName `[get]`

  Retorna el nombre de la pantalla.

5.51.1. Descripción detallada

La clase MessageScreen representa una pantalla para mostrar mensajes al usuario.

5.51.2. Documentación de las funciones miembro

5.51.2.1. AddDelegateToButton() `void MessageScreen.AddDelegateToButton ( string bType, OnMessageScreenButtonPressed bDelegate )`

Hace que el botón con nombre bType llame al delegado bDelegate cuando se hace click en él.

Parámetros

bType	Nombre del botón.
bDelegate	Método al que llamar cuando se pulse el botón.

5.51.2.2. ResetAllButtons() `void MessageScreen.ResetAllButtons ( )`

Resetea los delegados de los botones.

5.51.3. Documentación de propiedades

5.51.3.1. ScreenName `string MessageScreen.ScreenName [get]`

Retorna el nombre de la pantalla.

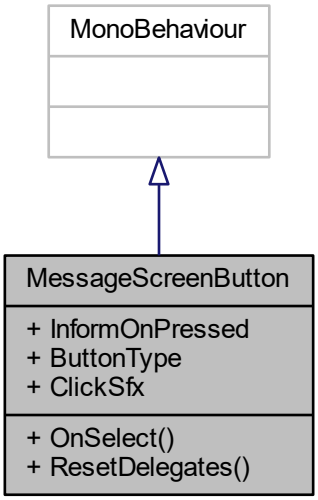
La documentación para esta clase fue generada a partir del siguiente fichero:

- Assets/Scripts/MessageScreen/MessageScreen.cs

5.52. Referencia de la Clase MessageScreenButton

Clase para los botones de <see cref="MessageScreen".

Diagrama de herencias de MessageScreenButton



Métodos públicos

- void OnSelect ()

  OnSelect.
- void ResetDelegates ()

  Pone el delegado a null.

Propiedades

- OnMessageScreenButtonPressed InformOnPressed `[get, set]`

  Para suscribirse al delegado.
- string ButtonType `[get, set]`

  Retorna o cambia el tipo de botón.
- AudioClip **ClickSfx** `[get, set]`

5.52.1. Descripción detallada

Clase para los botones de <see cref="MessageScreen".

5.52.2. Documentación de las funciones miembro

5.52.2.1. OnSelect() `void MessageScreenButton.OnSelect ( )`

OnSelect.

5.52.2.2. ResetDelegates() `void MessageScreenButton.ResetDelegates ( )`

Pone el delegado a null.

5.52.3. Documentación de propiedades

5.52.3.1. ButtonType `string MessageScreenButton.ButtonType [get], [set]`

Retorna o cambia el tipo de botón.

5.52.3.2. InformOnPressed `OnMessageScreenButtonPressed MessageScreenButton.InformOnPressed [get], [set]`

Para suscribirse al delegado.

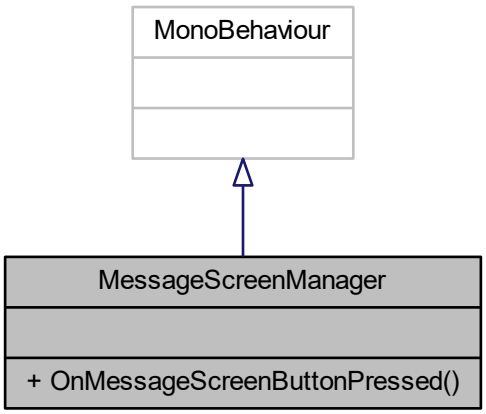
La documentación para esta clase fue generada a partir del siguiente fichero:

- Assets/Scripts/MessageScreen/MessageScreenButton.cs

5.53. Referencia de la Clase MessageScreenManager

Manager para objetos del tipo MessageScreen.

Diagrama de herencias de MessageScreenManager



Métodos públicos

- delegate void OnMessageScreenButtonPressed ()

  Delegado para cuando se pulsa un botón.

5.53.1. Descripción detallada

Manager para objetos del tipo MessageScreen.

5.53.2. Documentación de las funciones miembro

5.53.2.1. OnMessageScreenButtonPressed() `delegate void MessageScreenManager.OnMessageScreen↩ ButtonPressed ( )`

Delegado para cuando se pulsa un botón.

La documentación para esta clase fue generada a partir del siguiente fichero:

- Assets/Scripts/MessageScreen/MessageScreenManager.cs

5.54. Referencia de la Clase MessageWarehouse

Clase MessageWarehouse que actua como un "almacen" de mensajes para poder recibirlos de forma diferida. Muy usado en corrutinas.

Métodos públicos

- MessageWarehouse (EventAggregator eventAggregator)

  Crea una nueva instancia de la clase MessageWarehouse.
- bool IsResponseReceived< MessageType, ResponseType > (in MessageType msg, out ResponseType response)

  Comprueba si se ha recibido una respuesta a un mensaje.
- void PublishMsgAndWaitForResponse< MessageType, ResponseType > (MessageType msg)

  Sirve para publicar un mensaje.

5.54.1. Descripción detallada

Clase MessageWarehouse que actua como un "almacen" de mensajes para poder recibirlos de forma diferida. Muy usado en corrutinas.

5.54.2. Documentación del constructor y destructor

5.54.2.1. MessageWarehouse() `MessageWarehouse.MessageWarehouse ( EventAggregator eventAggregator )`

Crea una nueva instancia de la clase MessageWarehouse.

Parámetros

eventAggregator	El eventAggregatorEventAggregator.

5.54.3. Documentación de las funciones miembro

5.54.3.1. IsResponseReceived< MessageType, ResponseType >() bool MessageWarehouse.IsResponseReceived< MessageType, ResponseType > (

```
            in MessageType msg,
            out ResponseType response )
```

Comprueba si se ha recibido una respuesta a un mensaje.

Parámetros del template

MessageType	.
ResponseType	.

Parámetros

msg	El mensajeMessageType.
response	La respuestaResponseType.

Devuelve

bool True si se ha recibido, false si no.

5.54.3.2. PublishMsgAndWaitForResponse< MessageType, ResponseType >() void MessageWarehouse.PublishMsgAndWaitForResponse< MessageType, ResponseType > (

```
            MessageType msg )
```

Sirve para publicar un mensaje.

Parámetros del template

MessageType	.
ResponseType	.

Parámetros

msg	El mensaje MessageType.

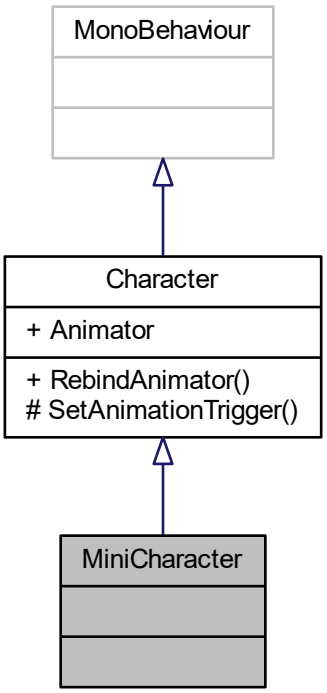
La documentación para esta clase fue generada a partir del siguiente fichero:

- Assets/Scripts/MessageHub/MessageWarehouse.cs

5.55. Referencia de la Clase MiniCharacter

Define la clase MiniCharacter del robot que se mueve por las carreteras.

Diagrama de herencias de MiniCharacter



Otros miembros heredados

5.55.1. Descripción detallada

Define la clase MiniCharacter del robot que se mueve por las carreteras.

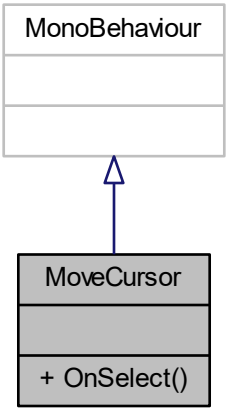
La documentación para esta clase fue generada a partir del siguiente fichero:

- Assets/Scripts/Character/MiniCharacter.cs

5.56. Referencia de la Clase MoveCursor

Esta clase manda un mensaje para ejecutar el método OnPlace() de su padre cuando se hace tap en ella.

Diagrama de herencias de MoveCursor



Métodos públicos

- void OnSelect ()
  OnSelect.

5.56.1. Descripción detallada

Esta clase manda un mensaje para ejecutar el método OnPlace() de su padre cuando se hace tap en ella.

5.56.2. Documentación de las funciones miembro

5.56.2.1. OnSelect() `void MoveCursor.OnSelect ( )`

OnSelect.

La documentación para esta clase fue generada a partir del siguiente fichero:

- Assets/Scripts/MapMenu/MoveCursor.cs

5.57. Referencia de la Clase MsgBigCharacterAllActionsFinished

Mensaje para preguntar si el robot grande ha finalizado sus acciones.

5.57.1. Descripción detallada

Mensaje para preguntar si el robot grande ha finalizado sus acciones.

La documentación para esta clase fue generada a partir del siguiente fichero:

- Assets/Scripts/MessageHub/Messages/MsgBigCharacterAllActionsFinished.cs

5.58. Referencia de la Clase MsgBigRobotAction

Mensaje para añadir una acción al robot principal.

Tipos públicos

- enum BigRobotActions {
  Jump, **Move**, **TurnLeft**, **TurnRight**,
  Win, **Lose** }
  Acciones disponibles.

Métodos públicos

- MsgBigRobotAction (BigRobotActions action, Vector3 target)
  Constructor del mensaje.

Propiedades

- BigRobotActions Action `[get]`
  Retorna la acción.
- Vector3 Target `[get]`
  Retorna el objetivo.

5.58.1. Descripción detallada

Mensaje para añadir una acción al robot principal.

5.58.2. Documentación de las enumeraciones miembro de la clase

5.58.2.1. BigRobotActions `enum MsgBigRobotAction.BigRobotActions [strong]`

Acciones disponibles.

5.58.3. Documentación del constructor y destructor

5.58.3.1. MsgBigRobotAction() `MsgBigRobotAction.MsgBigRobotAction (`

```
BigRobotActions action,
Vector3 target )
```

Constructor del mensaje.

Parámetros

action	La acción.
target	Posición objetivo.

5.58.4. Documentación de propiedades

5.58.4.1. Action `BigRobotActions MsgBigRobotAction.Action [get]`

Retorna la acción.

5.58.4.2. Target `Vector3 MsgBigRobotAction.Target [get]`

Retorna el objetivo.

La documentación para esta clase fue generada a partir del siguiente fichero:

- Assets/Scripts/MessageHub/Messages/MsgBigRobotAction.cs

5.59. Referencia de la Clase MsgBigRobotIdle

Mensaje para preguntar si el robot principal está parado.

5.59.1. Descripción detallada

Mensaje para preguntar si el robot principal está parado.

La documentación para esta clase fue generada a partir del siguiente fichero:

- Assets/Scripts/MessageHub/Messages/MsgBigRobotIdle.cs

5.60. Referencia de la Clase MsgBlockLength

Mensaje para pedir la longitud de un bloque.

5.60.1. Descripción detallada

Mensaje para pedir la longitud de un bloque.

La documentación para esta clase fue generada a partir del siguiente fichero:

- Assets/Scripts/MessageHub/Messages/MsgBlockLength.cs

5.61. Referencia de la Clase MsgDisableAllButtons

Mensaje para desactivar todos los botones.

5.61.1. Descripción detallada

Mensaje para desactivar todos los botones.

La documentación para esta clase fue generada a partir del siguiente fichero:

- Assets/Scripts/MessageHub/Messages/MsgDisableAllButtons.cs

5.62. Referencia de la Clase MsgEditorAvailableInstructionsChanged

Mensaje para cambiar el contador de número de instrucciones disponibles.

Métodos públicos

- MsgEditorAvailableInstructionsChanged (Buttons button, int numberOfInstrucions)
  Constructor de la clase.

Propiedades

- Buttons Button `[get, set]`
  Retorna el tipo de botón.
- int NumberOfInstructions `[get, set]`
  Retorna el número de instrucciones.

5.62.1. Descripción detallada

Mensaje para cambiar el contador de número de instrucciones disponibles.

5.62.2. Documentación del constructor y destructor

5.62.2.1. MsgEditorAvailableInstructionsChanged() `MsgEditorAvailableInstructionsChanged.Msg↩EditorAvailableInstructionsChanged (`

```
            Buttons button,
            int numberOfInstrucions )
```

Constructor de la clase.

Parámetros

button	El botón que hay que cambiar.
numberOfInstrucions	Número de instrucciones.

5.62.3. Documentación de propiedades

5.62.3.1. Button `Buttons MsgEditorAvailableInstructionsChanged.Button [get], [set]`

Retorna el tipo de botón.

5.62.3.2. NumberOfInstructions `int MsgEditorAvailableInstructionsChanged.NumberOfInstructions [get], [set]`

Retorna el número de instrucciones.

La documentación para esta clase fue generada a partir del siguiente fichero:

- Assets/Scripts/MessageHub/Messages/MsgEditorAvailableInstructionsChanged.cs

5.63. Referencia de la Clase MsgEditorMapSize

Pide la longitud del mapa x,y,z.

5.63.1. Descripción detallada

Pide la longitud del mapa x,y,z.

La documentación para esta clase fue generada a partir del siguiente fichero:

- Assets/Scripts/MessageHub/Messages/MsgEditorMapSize.cs

5.64. Referencia de la Clase MsgEditorMenu

Mensaje para el botón de volver al menú del editor.

5.64.1. Descripción detallada

Mensaje para el botón de volver al menú del editor.

La documentación para esta clase fue generada a partir del siguiente fichero:

- Assets/Scripts/MessageHub/Messages/MsgEditorMenu.cs

5.65. Referencia de la Clase MsgEditorResetAllCounters

Indica al editor que debe resetear todos los contadores.

5.65.1. Descripción detallada

Indica al editor que debe resetear todos los contadores.

La documentación para esta clase fue generada a partir del siguiente fichero:

- Assets/Scripts/MessageHub/Messages/MsgEditorResetAllCounters.cs

5.66. Referencia de la Clase MsgEditorSaveMap

Indica al editor que debe guardar el mapa.

5.66.1. Descripción detallada

Indica al editor que debe guardar el mapa.

La documentación para esta clase fue generada a partir del siguiente fichero:

- Assets/Scripts/MessageHub/Messages/MsgEditorSaveMap.cs

5.67. Referencia de la Clase MsgEditorSurfaceTapped

Mensaje para avisar de que se ha tocado en la superficie del editor.

Métodos públicos

- MsgEditorSurfaceTapped (EditorSurfacePoint tappedPoint)
  Inicializa una instancia del mensaje.

Propiedades

- EditorSurfacePoint TappedPoint `[get]`
  Retorna el punto tocado.

5.67.1. Descripción detallada

Mensaje para avisar de que se ha tocado en la superficie del editor.

5.67.2. Documentación del constructor y destructor

5.67.2.1. MsgEditorSurfaceTapped() `MsgEditorSurfaceTapped.MsgEditorSurfaceTapped ( EditorSurfacePoint tappedPoint )`

Inicializa una instancia del mensaje.

Parámetros

tappedPoint	El punto sobre el que se ha tocado.

5.67.3. Documentación de propiedades

5.67.3.1. TappedPoint `EditorSurfacePoint MsgEditorSurfaceTapped.TappedPoint [get]`

Retorna el punto tocado.

La documentación para esta clase fue generada a partir del siguiente fichero:

- Assets/Scripts/MessageHub/Messages/MsgEditorSurfaceTapped.cs

5.68. Referencia de la Clase MsgEditorToolSelected

Mensaje para seleccionar una herramienta del editor.

Métodos públicos

- MsgEditorToolSelected (EditorToolType toolType, int toolIdentifier)
  Crea una instancia del mensaje.

Propiedades

- EditorToolType ToolType `[get, set]`
  Retorna el tipo de herramienta.
- int ToolIdentifier `[get, set]`
  Retorna el parámetro extra.

5.68.1. Descripción detallada

Mensaje para seleccionar una herramienta del editor.

5.68.2. Documentación del constructor y destructor

5.68.2.1. MsgEditorToolSelected() `MsgEditorToolSelected.MsgEditorToolSelected ( EditorToolType toolType, int toolIdentifier )`

Crea una instancia del mensaje.

Parámetros

toolType	Tipo de herramienta.
toolIdentifier	Parámetro extra.

5.68.3. Documentación de propiedades

5.68.3.1. ToolIdentifier `int MsgEditorToolSelected.ToolIdentifier [get], [set]`

Retorna el parámetro extra.

5.68.3.2. ToolType `EditorToolType MsgEditorToolSelected.ToolType [get], [set]`

Retorna el tipo de herramienta.

La documentación para esta clase fue generada a partir del siguiente fichero:

- Assets/Scripts/MessageHub/Messages/MsgEditorToolSelected.cs

5.69. Referencia de la Clase MsgEnableAllButtons

Activa todos los botones.

5.69.1. Descripción detallada

Activa todos los botones.

La documentación para esta clase fue generada a partir del siguiente fichero:

- Assets/Scripts/MessageHub/Messages/MsgEnableAllButtons.cs

5.70. Referencia de la Clase MsgEnableButton

Mensaje para activar un botón.

Métodos públicos

- MsgEnableButton (Buttons button)

  Inicializa la clase.

Atributos públicos

- Buttons button
  El botón.

5.70.1. Descripción detallada

Mensaje para activar un botón.

5.70.2. Documentación del constructor y destructor

5.70.2.1. MsgEnableButton() `MsgEnableButton.MsgEnableButton ( Buttons button )`

Inicializa la clase.

Parámetros

button	El botón a activar.

5.70.3. Documentación de los datos miembro

5.70.3.1. button `Buttons MsgEnableButton.button`

El botón.

La documentación para esta clase fue generada a partir del siguiente fichero:

- Assets/Scripts/MessageHub/Messages/MsgEnableButton.cs

5.71. Referencia de la Clase MsgFindingSpace

Inicia o para el mensaje de "finding your space..."

Métodos públicos

- MsgFindingSpace (bool isFindingSpace)
  Crea una instancia del mensaje.

Atributos públicos

- bool isFindingSpace

  True si lo inicia, false si lo para.

5.71.1. Descripción detallada

Inicia o para el mensaje de "finding your space..."

5.71.2. Documentación del constructor y destructor

5.71.2.1. MsgFindingSpace() `MsgFindingSpace.MsgFindingSpace ( bool isFindingSpace )`

Crea una instancia del mensaje.

Parámetros

isFindingSpace	¿Está buscando el espacio?

5.71.3. Documentación de los datos miembro

5.71.3.1. isFindingSpace `bool MsgFindingSpace.isFindingSpace`

True si lo inicia, false si lo para.

La documentación para esta clase fue generada a partir del siguiente fichero:

- Assets/Scripts/MessageHub/Messages/MsgFindingSpace.cs

5.72. Referencia de la Clase MsgGetMainCameraTransform

Mensaje para pedir la transformada de la cámara principal.

5.72.1. Descripción detallada

Mensaje para pedir la transformada de la cámara principal.

La documentación para esta clase fue generada a partir del siguiente fichero:

- Assets/Scripts/MessageHub/Messages/MsgGetMainCameraTransform.cs

5.73. Referencia de la Clase MsgHideAllScreens

Mensaje para esconder todas las pantallas de aviso.

5.73.1. Descripción detallada

Mensaje para esconder todas las pantallas de aviso.

La documentación para esta clase fue generada a partir del siguiente fichero:

- Assets/Scripts/MessageHub/Messages/MsgHideAllScreens.cs

5.74. Referencia de la Clase MsgPlaceCharacter

Mensaje para colocar al robot principal.

Métodos públicos

- MsgPlaceCharacter (Vector3 position, Vector3 rotation, Transform newParent)
  Inicializa la clase.

Propiedades

- Vector3 Position `[get]`
  Retorna la posición.
- Vector3 Rotation `[get]`
  Retorna la rotación.
- Transform NewParent `[get]`
  Retorna el padre.

5.74.1. Descripción detallada

Mensaje para colocar al robot principal.

5.74.2. Documentación del constructor y destructor

5.74.2.1. MsgPlaceCharacter() `MsgPlaceCharacter.MsgPlaceCharacter (`

```
Vector3 position,
Vector3 rotation,
Transform newParent )
```

Inicializa la clase.

Parámetros

position	Posición.
rotation	Rotación.
newParent	El nuevo padre.

5.74.3. Documentación de propiedades

5.74.3.1. NewParent `Transform MsgPlaceCharacter.NewParent [get]`

Retorna el padre.

5.74.3.2. Position `Vector3 MsgPlaceCharacter.Position [get]`

Retorna la posición.

5.74.3.3. Rotation `Vector3 MsgPlaceCharacter.Rotation [get]`

Retorna la rotación.

La documentación para esta clase fue generada a partir del siguiente fichero:

- Assets/Scripts/MessageHub/Messages/MsgPlaceCharacter.cs

5.75. Referencia de la Clase MsgPlaySfx

Reproduce un SFX.

Métodos públicos

- MsgPlaySfx (AudioClip clip, float volume)
  Inicializa la clase.

Atributos públicos

- AudioClip clip
  El clip
- float volume
  El volumen.

5.75.1. Descripción detallada

Reproduce un SFX.

5.75.2. Documentación del constructor y destructor

5.75.2.1. MsgPlaySfx() `MsgPlaySfx.MsgPlaySfx (`

```
AudioClip clip,
float volume )
```

Inicializa la clase.

Parámetros

clip	El clip.
volume	El volumen.

5.75.3. Documentación de los datos miembro

5.75.3.1. clip `AudioClip MsgPlaySfx.clip`

El clip

5.75.3.2. volume `float MsgPlaySfx.volume`

El volumen.

La documentación para esta clase fue generada a partir del siguiente fichero:

- Assets/Scripts/MessageHub/Messages/MsgPlaySfx.cs

5.76. Referencia de la Clase MsgPlaySfxAtPoint

Reproduce un SFX en un punto.

Métodos públicos

- MsgPlaySfxAtPoint (AudioClip clip, float volume, Vector3 point)

  Inicializa el mensaje.

Atributos públicos

- AudioClip clip

  El clip.
- float volume

  El volumen.
- Vector3 point

  Punto donde tiene que reproducirse.

5.76.1. Descripción detallada

Reproduce un SFX en un punto.

5.76.2. Documentación del constructor y destructor

5.76.2.1. MsgPlaySfxAtPoint() `MsgPlaySfxAtPoint.MsgPlaySfxAtPoint (`

```
AudioClip clip,
float volume,
Vector3 point )
```

Inicializa el mensaje.

Parámetros

clip	El clip.
volume	El volumen.
point	La posición.

5.76.3. Documentación de los datos miembro

5.76.3.1. clip `AudioClip MsgPlaySfxAtPoint.clip`

El clip.

5.76.3.2. point `Vector3 MsgPlaySfxAtPoint.point`

Punto donde tiene que reproducirse.

5.76.3.3. volume `float MsgPlaySfxAtPoint.volume`

El volumen.

La documentación para esta clase fue generada a partir del siguiente fichero:

- Assets/Scripts/MessageHub/Messages/MsgPlaySfxAtPoint.cs

5.77. Referencia de la Clase MsgRenderMainCharacter

Mensaje para instanciar el robot principal.

5.77.1. Descripción detallada

Mensaje para instanciar el robot principal.

La documentación para esta clase fue generada a partir del siguiente fichero:

- Assets/Scripts/MessageHub/Messages/MsgRenderMainCharacter.cs

5.78. Referencia de la Clase MsgRenderMapAndItems

Mensaje para renderizar un nivel.

Métodos públicos

- MsgRenderMapAndItems (List< int > mapAndItems, List< int > levelSize, List< int > goal, GameObject mapParent)

  Inicializa el mensaje.

Propiedades

- List< int > MapAndItems `[get]`

  Retorna la lista de items y bloques.
- List< int > LevelSize `[get]`

  Retorna el tamaño del nivel.
- List< int > Goal `[get]`

  Retorna la meta.
- GameObject MapParent `[get]`

  Retorna el padre del mapa.

5.78.1. Descripción detallada

Mensaje para renderizar un nivel.

5.78.2. Documentación del constructor y destructor

5.78.2.1. MsgRenderMapAndItems() `MsgRenderMapAndItems.MsgRenderMapAndItems (`

```
List< int > mapAndItems,
List< int > levelSize,
List< int > goal,
GameObject mapParent )
```

Inicializa el mensaje.

Parámetros

mapAndItems	Lista de bloques e items.
levelSize	Tamaño del nivel.
goal	Meta del nivel.
mapParent	Padre del nivel.

5.78.3. Documentación de propiedades

5.78.3.1. Goal `List<int> MsgRenderMapAndItems.Goal [get]`

Retorna la meta.

5.78.3.2. LevelSize `List<int> MsgRenderMapAndItems.LevelSize [get]`

Retorna el tamaño del nivel.

5.78.3.3. MapAndItems `List<int> MsgRenderMapAndItems.MapAndItems [get]`

Retorna la lista de items y bloques.

5.78.3.4. MapParent `GameObject MsgRenderMapAndItems.MapParent [get]`

Retorna el padre del mapa.

La documentación para esta clase fue generada a partir del siguiente fichero:

- Assets/Scripts/MessageHub/Messages/MsgRenderMapAndItems.cs

5.79. Referencia de la Clase MsgResetEditorSurface

Resetea la superficie del editor.

5.79.1. Descripción detallada

Resetea la superficie del editor.

La documentación para esta clase fue generada a partir del siguiente fichero:

- Assets/Scripts/MessageHub/Messages/MsgResetEditorSurface.cs

5.80. Referencia de la Clase MsgSetAvInstructions

Cambia los contadores de instrucciones restantes.

Métodos públicos

- MsgSetAvInstructions (AvailableInstructions avInst)
  Inicializa la clase.

Atributos públicos

- AvailableInstructions avInst
  Instrucciones restantes.

5.80.1. Descripción detallada

Cambia los contadores de instrucciones restantes.

5.80.2. Documentación del constructor y destructor

5.80.2.1. MsgSetAvInstructions() `MsgSetAvInstructions.MsgSetAvInstructions (AvailableInstructions avInst )`

Inicializa la clase.

Parámetros

avInst	Instrucciones restantes.

5.80.3. Documentación de los datos miembro

5.80.3.1. avInst `AvailableInstructions MsgSetAvInstructions.avInst`

Instrucciones restantes.

La documentación para esta clase fue generada a partir del siguiente fichero:

- Assets/Scripts/MessageHub/Messages/MsgSetAvInstructions.cs

5.81. Referencia de la Clase MsgShowScreen

Mensaje para mostrar una pantalla de información.

Métodos públicos

- MsgShowScreen (string screenName, Tuple< string, OnMessageScreenButtonPressed >[] listOfActions)

  Inicializa el mensaje.
- MsgShowScreen (string screenName, float seconds)

  Inicializa el mensaje.

Atributos públicos

- string screenName

  Nombre de la pantalla.
- Tuple< string, OnMessageScreenButtonPressed >[] listOfActions

  Lista de tuplas (nombre de botón, acción).
- float seconds = -1

  Segundos que debe estar la pantalla activa.

5.81.1. Descripción detallada

Mensaje para mostrar una pantalla de información.

5.81.2. Documentación del constructor y destructor

5.81.2.1. MsgShowScreen() [1/2] MsgShowScreen.MsgShowScreen (

```
string screenName,
Tuple< string, OnMessageScreenButtonPressed >[] listOfActions )
```

Inicializa el mensaje.

Parámetros

screenName	Nombre de la pantalla.
listOfActions	Lista de tuplas (nombre de botón, acción).

5.81.2.2. MsgShowScreen() [2/2] MsgShowScreen.MsgShowScreen (

```
string screenName,
float seconds )
```

Inicializa el mensaje.

Parámetros

screenName	Nombre de la pantalla.
seconds	Segundos que debe estar activa.

5.81.3. Documentación de los datos miembro

5.81.3.1. listOfActions `Tuple<string, OnMessageScreenButtonPressed> [] MsgShowScreen.listOfActions`

Lista de tuplas (nombre de botón, acción).

5.81.3.2. screenName `string MsgShowScreen.screenName`

Nombre de la pantalla.

5.81.3.3. seconds `float MsgShowScreen.seconds = -1`

Segundos que debe estar la pantalla activa.

La documentación para esta clase fue generada a partir del siguiente fichero:

- Assets/Scripts/MessageHub/Messages/MsgShowScreen.cs

5.82. Referencia de la Clase MsgSomethingTapped

Indica que el usuario ha hecho tap.

5.82.1. Descripción detallada

Indica que el usuario ha hecho tap.

La documentación para esta clase fue generada a partir del siguiente fichero:

- Assets/Scripts/MessageHub/Messages/MsgSomethingTapped.cs

5.83. Referencia de la Clase MsgStartRoadMovement

Mensaje para que el robot pequeño empiece a moverse.

Métodos públicos

- MsgStartRoadMovement (RoadInput input, RoadOutput output)

  Inicializa una instancia del mensaje.

Atributos públicos

- RoadInput input

  Input inicial.
- RoadOutput output

  Output final.

5.83.1. Descripción detallada

Mensaje para que el robot pequeño empiece a moverse.

5.83.2. Documentación del constructor y destructor

5.83.2.1. MsgStartRoadMovement() `MsgStartRoadMovement.MsgStartRoadMovement (`

```
RoadInput input,
RoadOutput output )
```

Inicializa una instancia del mensaje.

Parámetros

input	El input.
output	El output.

5.83.3. Documentación de los datos miembro

5.83.3.1. input `RoadInput MsgStartRoadMovement.input`

Input inicial.

5.83.3.2. output `RoadOutput MsgStartRoadMovement.output`

Output final.

La documentación para esta clase fue generada a partir del siguiente fichero:

- Assets/Scripts/MessageHub/Messages/MsgStartRoadMovement.cs

5.84. Referencia de la Clase MsgStopMovement

Mensaje para parar el movimiento del robot pequeño.

5.84.1. Descripción detallada

Mensaje para parar el movimiento del robot pequeño.

La documentación para esta clase fue generada a partir del siguiente fichero:

- Assets/Scripts/MessageHub/Messages/MsgStopMovement.cs

5.85. Referencia de la Clase MsgTakeItem

Mensaje para recoger un item.

Métodos públicos

- MsgTakeItem (Item item, int numberOfItems)
  Constructor del mensaje.

Atributos públicos

- Item item
  El item.
- int numberOfItems
  Número de items en el inventario.

5.85.1. Descripción detallada

Mensaje para recoger un item.

5.85.2. Documentación del constructor y destructor

5.85.2.1. MsgTakeItem() `MsgTakeItem.MsgTakeItem (`

```
Item item,
int numberOfItems )
```

Constructor del mensaje.

Parámetros

item	El item.
numberOfItems	Número de items en el inventario.

5.85.3. Documentación de los datos miembro

5.85.3.1. item `Item MsgTakeItem.item`

El item.

5.85.3.2. numberOfItems `int MsgTakeItem.numberOfItems`

Número de items en el inventario.

La documentación para esta clase fue generada a partir del siguiente fichero:

- Assets/Scripts/MessageHub/Messages/MsgTakeItem.cs

5.86. Referencia de la Clase MsgUseItem

Mensaje que indica al robot que debe usar un item.

Métodos públicos

- MsgUseItem (Block frontBlock, EffectReaction reaction, LevelObject replaceBlock, Vector3 itemPos, Item item, Stack< Item > inventory)

  Inicializa el mensaje.

Atributos públicos

- Block frontBlock

  Bloque sobre el que usar el item.
- EffectReaction reaction

  Reacción del item.
- LevelObject replaceBlock

  ¿Hay que cambiar el bloque por otro?
- Vector3 itemPos

  Posición del item.
- Item item

  El item.
- Stack< Item > inventory

  Inventario del robot.

5.86.1. Descripción detallada

Mensaje que indica al robot que debe usar un item.

5.86.2. Documentación del constructor y destructor

5.86.2.1. MsgUseItem() `MsgUseItem.MsgUseItem (`

```
Block frontBlock,
EffectReaction reaction,
LevelObject replaceBlock,
Vector3 itemPos,
Item item,
Stack< Item > inventory )
```

Inicializa el mensaje.

Parámetros

frontBlock	Bloque sobre el que usar el item.
reaction	Reacción del item.
replaceBlock	¿Hay que cambiar el bloque por otro?.
itemPos	Posición del item.
item	El item.
inventory	Inventario del robot.

5.86.3. Documentación de los datos miembro

5.86.3.1. frontBlock `Block MsgUseItem.frontBlock`

Bloque sobre el que usar el item.

5.86.3.2. inventory `Stack<Item> MsgUseItem.inventory`

Inventario del robot.

5.86.3.3. item `Item MsgUseItem.item`

El item.

5.86.3.4. itemPos `Vector3 MsgUseItem.itemPos`

Posición del item.

5.86.3.5. reaction `EffectReaction MsgUseItem.reaction`

Reacción del item.

5.86.3.6. replaceBlock `LevelObject MsgUseItem.replaceBlock`

¿Hay que cambiar el bloque por otro?

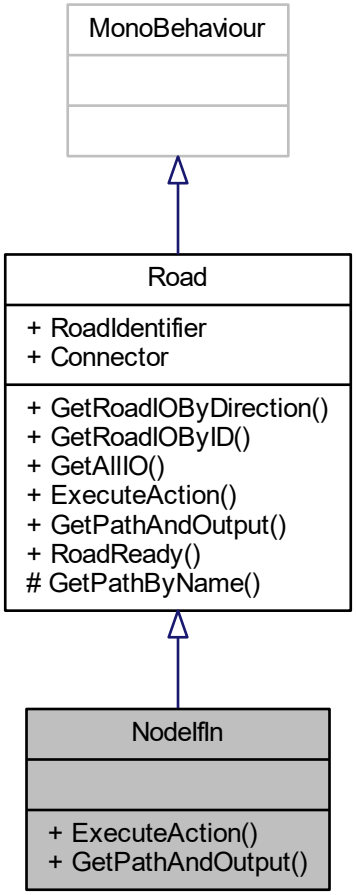
La documentación para esta clase fue generada a partir del siguiente fichero:

- Assets/Scripts/MessageHub/Messages/MsgUseItem.cs

5.87. Referencia de la Clase NodeIfIn

Clase de entrada al if.

Diagrama de herencias de NodeIfIn



Métodos públicos

- override void ExecuteAction (in string[ ] args)

  Ejecuta una acción en base a una lista de argumentos.
- override bool GetPathAndOutput (in RoadInput input, out Path path, out RoadOutput output)

  Dado un input retorna el camino que inicia en él y el output en el que termina.

Otros miembros heredados

5.87.1. Descripción detallada

Clase de entrada al if.

5.87.2. Documentación de las funciones miembro

5.87.2.1. ExecuteAction() `override void NodeIfIn.ExecuteAction ( in string[] args ) [virtual]`

Ejecuta una acción en base a una lista de argumentos.

Parámetros

args	Los argumentos.

Implementa Road.

5.87.2.2. GetPathAndOutput() `override bool NodeIfIn.GetPathAndOutput ( in RoadInput input, out Path path, out RoadOutput output ) [virtual]`

Dado un input retorna el camino que inicia en él y el output en el que termina.

Parámetros

input	El input RoadInput.
path	El camino Path.
output	El output RoadOutput.

Devuelve

True si hay camino, false si no.

Implementa Road.

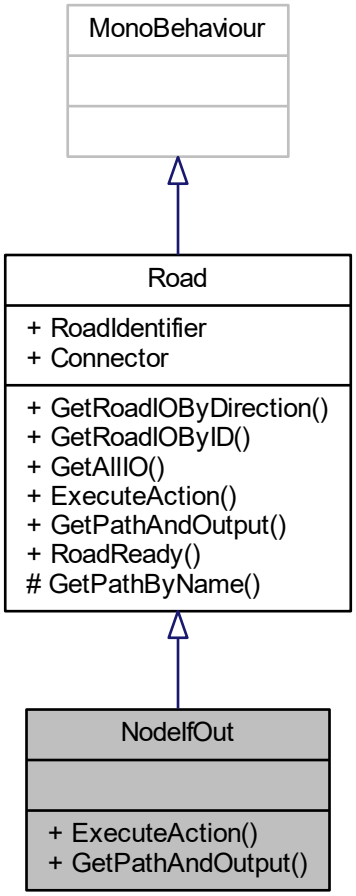
La documentación para esta clase fue generada a partir del siguiente fichero:

- Assets/Scripts/Roads/Roads/NodeIfIn.cs

5.88. Referencia de la Clase NodeIfOut

Clase de la carretera de salida de un if.

Diagrama de herencias de NodeIfOut



Métodos públicos

- override void ExecuteAction (in string[ ] args)

  Ejecuta una acción en base a una lista de argumentos.
- override bool GetPathAndOutput (in RoadInput input, out Path path, out RoadOutput output)

  Dado un input retorna el camino que inicia en él y el output en el que termina.

Otros miembros heredados

5.88.1. Descripción detallada

Clase de la carretera de salida de un if.

5.88.2. Documentación de las funciones miembro

5.88.2.1. ExecuteAction() `override void NodeIfOut.ExecuteAction ( in string[] args ) [virtual]`

Ejecuta una acción en base a una lista de argumentos.

Parámetros

args	Los argumentos.

Implementa Road.

5.88.2.2. GetPathAndOutput() `override bool NodeIfOut.GetPathAndOutput ( in RoadInput input, out Path path, out RoadOutput output ) [virtual]`

Dado un input retorna el camino que inicia en él y el output en el que termina.

Parámetros

input	El input RoadInput.
path	El camino Path.
output	El output RoadOutput.

Devuelve

True si hay camino, false si no.

Implementa Road.

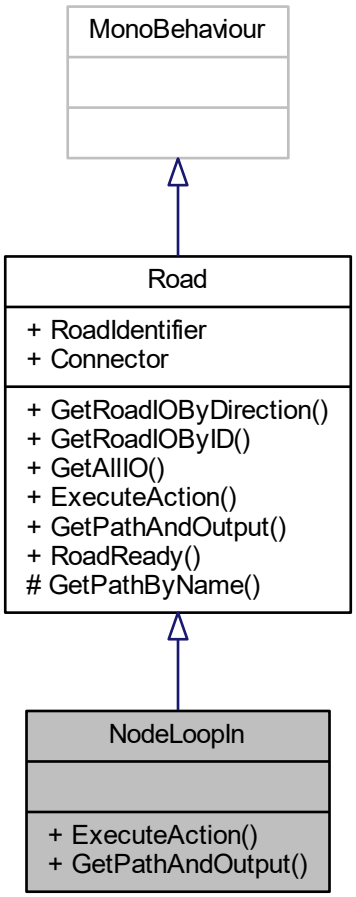
La documentación para esta clase fue generada a partir del siguiente fichero:

- Assets/Scripts/Roads/Roads/NodeIfOut.cs

5.89. Referencia de la Clase NodeLoopIn

Clase de la carretera de entrada al bucle.

Diagrama de herencias de NodeLoopIn



Métodos públicos

- override void ExecuteAction (in string[ ] args)

  Ejecuta una acción en base a una lista de argumentos.
- override bool GetPathAndOutput (in RoadInput input, out Path path, out RoadOutput output)

  Dado un input retorna el camino que inicia en él y el output en el que termina.

Otros miembros heredados

5.89.1. Descripción detallada

Clase de la carretera de entrada al bucle.

5.89.2. Documentación de las funciones miembro

5.89.2.1. ExecuteAction() `override void NodeLoopIn.ExecuteAction ( in string[] args ) [virtual]`

Ejecuta una acción en base a una lista de argumentos.

Parámetros

args	Los argumentos.

Implementa Road.

5.89.2.2. GetPathAndOutput() `override bool NodeLoopIn.GetPathAndOutput ( in RoadInput input, out Path path, out RoadOutput output ) [virtual]`

Dado un input retorna el camino que inicia en él y el output en el que termina.

Parámetros

input	El input RoadInput.
path	El camino Path.
output	El output RoadOutput.

Devuelve

True si hay camino, false si no.

Implementa Road.

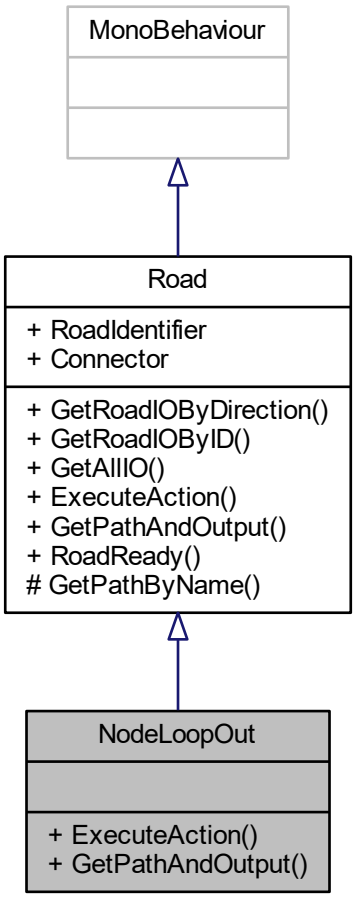
La documentación para esta clase fue generada a partir del siguiente fichero:

- Assets/Scripts/Roads/Roads/NodeLoopIn.cs

5.90. Referencia de la Clase NodeLoopOut

Clase de la carretera de entrada al bucle.

Diagrama de herencias de NodeLoopOut



Métodos públicos

- override void ExecuteAction (in string[ ] args)

  Ejecuta una acción en base a una lista de argumentos.
- override bool GetPathAndOutput (in RoadInput input, out Path path, out RoadOutput output)

  Dado un input retorna el camino que inicia en él y el output en el que termina.

Otros miembros heredados

5.90.1. Descripción detallada

Clase de la carretera de entrada al bucle.

5.90.2. Documentación de las funciones miembro

5.90.2.1. ExecuteAction() override void NodeLoopOut.ExecuteAction (in string[] *args*) [virtual]

Ejecuta una acción en base a una lista de argumentos.

Parámetros

args	Los argumentos.

Implementa Road.

5.90.2.2. GetPathAndOutput() override bool NodeLoopOut.GetPathAndOutput (in RoadInput *input*, out Path *path*, out RoadOutput *output*) [virtual]

Dado un input retorna el camino que inicia en él y el output en el que termina.

Parámetros

input	El input RoadInput.
path	El camino Path.
output	El output RoadOutput.

Devuelve

True si hay camino, false si no.

Implementa Road.

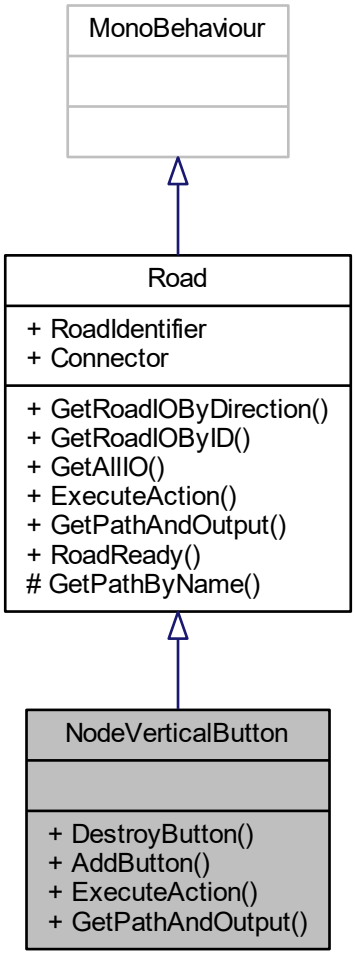
La documentación para esta clase fue generada a partir del siguiente fichero:

- Assets/Scripts/Roads/Roads/NodeLoopOut.cs

5.91. Referencia de la Clase NodeVerticalButton

Clase de la carretera con botones.

Diagrama de herencias de NodeVerticalButton



Métodos públicos

- bool DestroyButton (in VerticalButton button)

  Destruye un botón.
- bool AddButton (string buttonName, RoadIO io, out VerticalButton spwButton)

  Crea un botón si es posible.
- override void ExecuteAction (in string[ ] args)

  Ejecuta una acción en base a una lista de argumentos.
- override bool GetPathAndOutput (in RoadInput input, out Path path, out RoadOutput output)

  Dado un input retorna el camino que inicia en él y el output en el que termina.

Otros miembros heredados

5.91.1. Descripción detallada

Clase de la carretera con botones.

5.91.2. Documentación de las funciones miembro

5.91.2.1. AddButton() `bool NodeVerticalButton.AddButton (`

```
string buttonName,
RoadIO io,
out VerticalButton spwButton )
```

Crea un botón si es posible.

Parámetros

buttonName	Nombre del botón.
io	IO más cercana al botón.
spwButton	El botón creado.

Devuelve

True si lo ha podido crear, false si no.

5.91.2.2. DestroyButton() `bool NodeVerticalButton.DestroyButton (`

```
in VerticalButton button )
```

Destruye un botón.

Parámetros

button	El botón VerticalButton.

Devuelve

True si ha podido destruirlo, false si no.

5.91.2.3. ExecuteAction() `override void NodeVerticalButton.ExecuteAction (`

```
in string[] args )  [virtual]
```

Ejecuta una acción en base a una lista de argumentos.

Parámetros

args	Los argumentos.

Implementa Road.

5.91.2.4. GetPathAndOutput()

```
override bool NodeVerticalButton.GetPathAndOutput (
        in RoadInput input,
        out Path path,
        out RoadOutput output )  [virtual]
```

Dado un input retorna el camino que inicia en él y el output en el que termina.

Parámetros

input	El input RoadInput.
path	El camino Path.
output	El output RoadOutput.

Devuelve

True si hay camino, false si no.

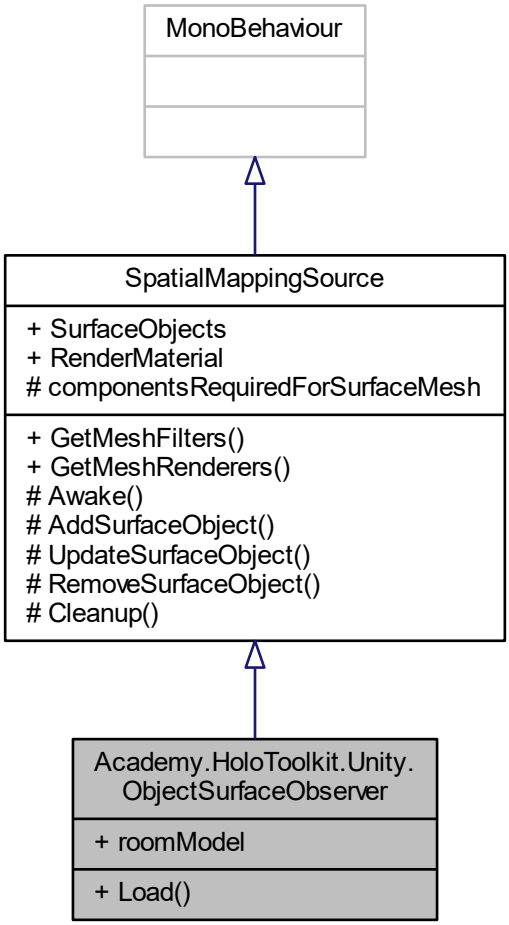
Implementa Road.

La documentación para esta clase fue generada a partir del siguiente fichero:

- Assets/Scripts/Roads/Roads/NodeVerticalButton.cs

5.92. Referencia de la Clase Academy.HoloToolkit.Unity.ObjectSurfaceObserver

Diagrama de herencias de Academy.HoloToolkit.Unity.ObjectSurfaceObserver



Métodos públicos

- void Load (GameObject roomModel)

  Loads the SpatialMapping mesh from the specified room object.

Atributos públicos

- GameObject **roomModel**

Otros miembros heredados

5.92.1. Documentación de las funciones miembro

5.92.1.1. Load() `void Academy.HoloToolkit.Unity.ObjectSurfaceObserver.Load ( GameObject roomModel )`

Loads the SpatialMapping mesh from the specified room object.

Parámetros

roomModel	The room model to load meshes from.

La documentación para esta clase fue generada a partir del siguiente fichero:

- Assets/HoloToolkit-SpatialMapping-230/SpatialMapping/Scripts/ObjectSurfaceObserver.cs

5.93. Referencia de la Estructura Academy.HoloToolkit.Unity.OrientedBoundingBox

Atributos públicos

- Vector3 **Center**
- Vector3 **Extents**
- Quaternion **Rotation**

La documentación para esta estructura fue generada a partir del siguiente fichero:

- Assets/HoloToolkit-SpatialMapping-230/SpatialMapping/Scripts/PlaneFinding.cs

5.94. Referencia de la Estructura PathContainer.Path

Define un camino.

Atributos públicos

- string pathName
  Nombre del camino.
- Transform[ ] points
  Puntos que lo forman.
- Color color
  Color del camino.
- bool drawPreview
  ¿Se dibujará el gizmo del camino?
- RoadInput ioBegin
  IO inicial.
- RoadOutput ioEnd
  IO final.

5.94.1. Descripción detallada

Define un camino.

5.94.2. Documentación de los datos miembro

5.94.2.1. color `Color PathContainer.Path.color`

Color del camino.

5.94.2.2. drawPreview `bool PathContainer.Path.drawPreview`

¿Se dibujará el gizmo del camino?

5.94.2.3. ioBegin `RoadInput PathContainer.Path.ioBegin`

IO inicial.

5.94.2.4. ioEnd `RoadOutput PathContainer.Path.ioEnd`

IO final.

5.94.2.5. pathName `string PathContainer.Path.pathName`

Nombre del camino.

5.94.2.6. points `Transform [] PathContainer.Path.points`

Puntos que lo forman.

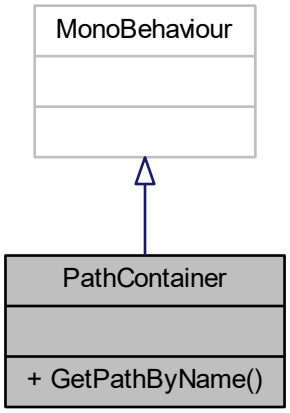
La documentación para esta estructura fue generada a partir del siguiente fichero:

- Assets/Scripts/Roads/Path/PathContainer.cs

5.95. Referencia de la Clase PathContainer

Contiene caminos por los que se moverá el robot que son listas de puntos ordenados.

Diagrama de herencias de PathContainer



Clases

- struct Path

  Define un camino.

Métodos públicos

- bool GetPathByName (in string name, out Path path)

  Retorna un camino por su nombre.

5.95.1. Descripción detallada

Contiene caminos por los que se moverá el robot que son listas de puntos ordenados.

5.95.2. Documentación de las funciones miembro

5.95.2.1. GetPathByName()

```
bool PathContainer.GetPathByName (
        in string name,
        out Path path )
```

Retorna un camino por su nombre.

Parámetros

name	El nombre del camino.
path	El camino.

Devuelve

True si se ha encontrado, false si no.

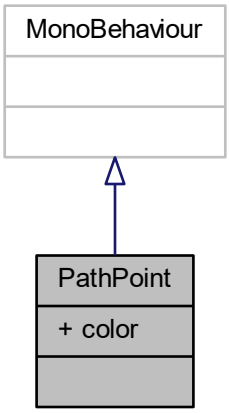
La documentación para esta clase fue generada a partir del siguiente fichero:

- Assets/Scripts/Roads/Path/PathContainer.cs

5.96. Referencia de la Clase PathPoint

Marca un punto en el camino del robot.

Diagrama de herencias de PathPoint



Atributos públicos

- Color color = Color.yellow

  Color del punto.

5.96.1. Descripción detallada

Marca un punto en el camino del robot.

5.96.2. Documentación de los datos miembro

5.96.2.1. color `Color PathPoint.color = Color.yellow`

Color del punto.

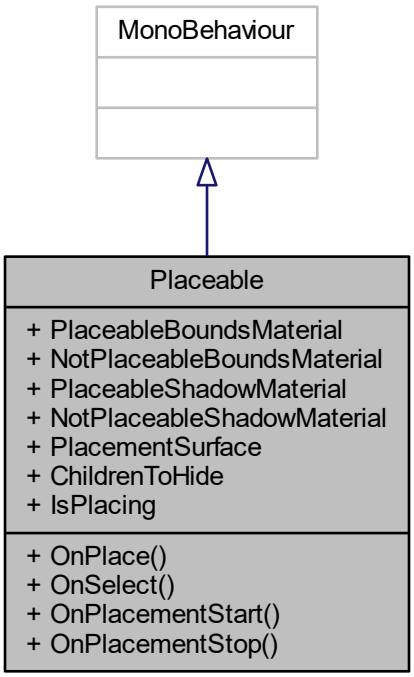
La documentación para esta clase fue generada a partir del siguiente fichero:

- Assets/Scripts/Roads/Path/PathPoint.cs

5.97. Referencia de la Clase Placeable

La clase Placeable implementa la lógica usada para determinar si un GameObject puede ser colocado en una superficie. Las restricciones que deben tenerse en cuenta a la hora de colocar un objeto son:

Diagrama de herencias de Placeable



Métodos públicos

- void **OnPlace** ()
- void **OnSelect** ()
- void OnPlacementStart ()

  Pone el objeto en modo de emplazamiento.
- void OnPlacementStop ()

  Saca al objeto del modo de emplazamiento.

Atributos públicos

- Material **PlaceableBoundsMaterial** = null
- Material **NotPlaceableBoundsMaterial** = null
- Material **PlaceableShadowMaterial** = null
- Material **NotPlaceableShadowMaterial** = null
- PlacementSurfaces **PlacementSurface** = PlacementSurfaces.Horizontal
- List< GameObject > **ChildrenToHide** = new List<GameObject>()

Propiedades

- bool IsPlacing `[get]`

  Indica si el objeto está en proceso de ser colocado.

5.97.1. Descripción detallada

La clase Placeable implementa la lógica usada para determinar si un GameObject puede ser colocado en una superficie. Las restricciones que deben tenerse en cuenta a la hora de colocar un objeto son:

- Ninguna parte de su BoxCollider se solapa con otro objeto.
- El objeto se coloca plano (dentro de las tolerancias especificadas) contra la superficie.
- No se caeria de la superficie si la gravedad estuviera activada. Esta clase tambien provee un cubo transparente que representa el BoxCollider del objeto y una sombra indicando si la posición es válida o no.

5.97.2. Documentación de las funciones miembro

5.97.2.1. OnPlacementStart() `void Placeable.OnPlacementStart ( )`

Pone el objeto en modo de emplazamiento.

5.97.2.2. OnPlacementStop() `void Placeable.OnPlacementStop ( )`

Saca al objeto del modo de emplazamiento.

5.97.3. Documentación de propiedades

5.97.3.1. IsPlacing `bool Placeable.IsPlacing [get]`

Indica si el objeto está en proceso de ser colocado.

La documentación para esta clase fue generada a partir del siguiente fichero:

- Assets/Scripts/Hololens/Placeable.cs

5.98. Referencia de la Clase Academy.HoloToolkit.Unity.PlaneFinding

Clases

- struct MeshData

  PlaneFinding is an expensive task that should not be run from Unity's main thread as it will stall the thread and cause a frame rate dip. Instead, the PlaneFinding APIs should be exclusively called from background threads. Unfortunately, Unity's built-in data types (such as MeshFilter) are not thread safe and cannot be accessed from background threads. The MeshData struct exists to work-around this limitation. When you want to find planes in a collection of MeshFilter objects, start by constructing a list of MeshData structs from those MeshFilters. You can then take the resulting list of MeshData structs, and safely pass it to the FindPlanes() API from a background thread.

Métodos públicos estáticos

- static BoundedPlane[ ] FindSubPlanes (List< MeshData > meshes, float snapToGravityThreshold=0.0f)

  Finds small planar patches that are contained within individual meshes. The output of this API can then be passed to MergeSubPlanes() in order to find larger planar surfaces that potentially span across multiple meshes.
- static BoundedPlane[ ] MergeSubPlanes (BoundedPlane[ ] subPlanes, float snapToGravityThreshold=0.0f, float minArea=0.0f)

  Takes the subplanes returned by one or more previous calls to FindSubPlanes() and merges them together into larger planes that can potentially span across multiple meshes. Overlapping subplanes that have similar plane equations will be merged together to form larger planes.
- static BoundedPlane[ ] FindPlanes (List< MeshData > meshes, float snapToGravityThreshold=0.0f, float minArea=0.0f)

  Convenience wrapper that executes FindSubPlanes followed by MergeSubPlanes via a single call into native code (which improves performance by avoiding a bunch of unnecessary data marshalling and a managed-to-native transition).

5.98.1. Documentación de las funciones miembro

5.98.1.1. FindPlanes() `static BoundedPlane [ ] Academy.HoloToolkit.Unity.PlaneFinding.FindPlanes (`

```
List< MeshData > meshes,
float snapToGravityThreshold = 0.0f,
float minArea = 0.0f )  [static]
```

Convenience wrapper that executes FindSubPlanes followed by MergeSubPlanes via a single call into native code (which improves performance by avoiding a bunch of unnecessary data marshalling and a managed-to-native transition).

Parámetros

meshes	List of meshes to run the plane finding algorithm on.
snapToGravityThreshold	Planes whose normal vectors are within this threshold (in degrees) from vertical/horizontal will be snapped to be perfectly gravity aligned. When set to something other than zero, the bounding boxes for each plane will be gravity aligned as well, rather than rotated for an optimally tight fit. Pass 0.0 for this parameter to completely disable the gravity alignment logic.
minArea	While merging subplanes together, any candidate merged plane whose constituent mesh triangles have a total area less than this threshold are ignored.

5.98.1.2. FindSubPlanes()

```
static BoundedPlane [] Academy.HoloToolkit.Unity.PlaneFinding.FindSubPlanes (
            List< MeshData > meshes,
            float snapToGravityThreshold = 0.0f )  [static]
```

Finds small planar patches that are contained within individual meshes. The output of this API can then be passed to MergeSubPlanes() in order to find larger planar surfaces that potentially span across multiple meshes.

Parámetros

meshes	List of meshes to run the plane finding algorithm on.
snapToGravityThreshold	Planes whose normal vectors are within this threshold (in degrees) from vertical/horizontal will be snapped to be perfectly gravity aligned. When set to something other than zero, the bounding boxes for each plane will be gravity aligned as well, rather than rotated for an optimally tight fit. Pass 0.0 for this parameter to completely disable the gravity alignment logic.

5.98.1.3. MergeSubPlanes()

```
static BoundedPlane [] Academy.HoloToolkit.Unity.PlaneFinding.MergeSubPlanes (
            BoundedPlane[] subPlanes,
            float snapToGravityThreshold = 0.0f,
            float minArea = 0.0f )  [static]
```

Takes the subplanes returned by one or more previous calls to FindSubPlanes() and merges them together into larger planes that can potentially span across multiple meshes. Overlapping subplanes that have similar plane equations will be merged together to form larger planes.

Parámetros

subPlanes	The output from one or more previous calls to FindSubPlanes().
snapToGravityThreshold	Planes whose normal vectors are within this threshold (in degrees) from vertical/horizontal will be snapped to be perfectly gravity aligned. When set to something other than zero, the bounding boxes for each plane will be gravity aligned as well, rather than rotated for an optimally tight fit. Pass 0.0 for this parameter to completely disable the gravity alignment logic.
minArea	While merging subplanes together, any candidate merged plane whose constituent mesh triangles have a total area less than this threshold are ignored.

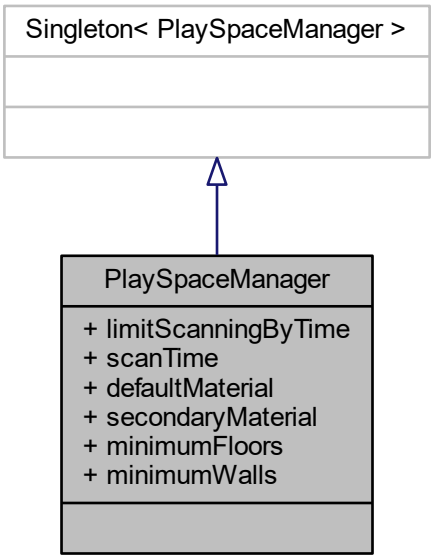
La documentación para esta clase fue generada a partir del siguiente fichero:

- Assets/HoloToolkit-SpatialMapping-230/SpatialMapping/Scripts/PlaneFinding.cs

5.99. Referencia de la Clase PlaySpaceManager

La clase SurfaceManager permite a los programas escanear el espacio que rodea al usuario por un tiempo especificado y después procesará el Spatial Mapping Mesh (encontrar planos, eliminar vértices) una vez que el tiempo acabó.

Diagrama de herencias de PlaySpaceManager



Atributos públicos

- bool **limitScanningByTime** = true
- float **scanTime** = 30.0f
- Material **defaultMaterial**
- Material **secondaryMaterial**
- uint **minimumFloors** = 1
- uint **minimumWalls** = 1

5.99.1. Descripción detallada

La clase SurfaceManager permite a los programas escanear el espacio que rodea al usuario por un tiempo especificado y después procesará el Spatial Mapping Mesh (encontrar planos, eliminar vértices) una vez que el tiempo acabó.

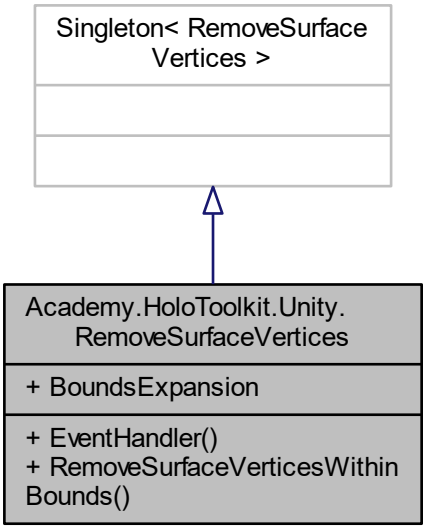
La documentación para esta clase fue generada a partir del siguiente fichero:

- Assets/Scripts/Hololens/PlaySpaceManager.cs

5.100. Referencia de la Clase Academy.HoloToolkit.Unity.RemoveSurfaceVertices

RemoveSurfaceVertices will remove any vertices from the Spatial Mapping Mesh that fall within the bounding volume. This can be used to create holes in the environment, or to help reduce triangle count after finding planes.

Diagrama de herencias de Academy.HoloToolkit.Unity.RemoveSurfaceVertices



Métodos públicos

- delegate void EventHandler (object source, EventArgs args)

  Delegate which is called when the RemoveVerticesComplete event is triggered.
- void RemoveSurfaceVerticesWithinBounds (IEnumerable< GameObject > boundingObjects)

  Removes portions of the surface mesh that exist within the bounds of the boundingObjects.

Atributos públicos

- float **BoundsExpansion** = 0.0f

Eventos

- EventHandler RemoveVerticesComplete

  EventHandler which is triggered when the RemoveSurfaceVertices is finished.

5.100.1. Descripción detallada

RemoveSurfaceVertices will remove any vertices from the Spatial Mapping Mesh that fall within the bounding volume. This can be used to create holes in the environment, or to help reduce triangle count after finding planes.

5.100.2. Documentación de las funciones miembro

5.100.2.1. EventHandler() `delegate void Academy.HoloToolkit.Unity.RemoveSurfaceVertices.EventHandler (`

```
object source,
EventArgs args )
```

Delegate which is called when the RemoveVerticesComplete event is triggered.

Parámetros

source	
args	

5.100.2.2. RemoveSurfaceVerticesWithinBounds() `void Academy.HoloToolkit.Unity.RemoveSurfaceVertices.RemoveSurfaceVerticesWithinBounds (`

```
IEnumerable< GameObject > boundingObjects )
```

Removes portions of the surface mesh that exist within the bounds of the boundingObjects.

Parámetros

boundingObjects	Collection of GameObjects that define the bounds where spatial mesh vertices should be removed.

5.100.3. Documentación de los eventos

5.100.3.1. RemoveVerticesComplete `EventHandler Academy.HoloToolkit.Unity.RemoveSurfaceVertices.RemoveVerticesComplete`

EventHandler which is triggered when the RemoveSurfaceVertices is finished.

La documentación para esta clase fue generada a partir del siguiente fichero:

- Assets/HoloToolkit-SpatialMapping-230/SpatialMapping/Scripts/RemoveSurfaceVertices.cs

5.101. Referencia de la plantilla de la Clase ResponseWrapper< TPetition, TResponse >

Clase para generar respuesta a un mensaje.

Métodos públicos

- ResponseWrapper (TPetition petition, TResponse response)

  Inicializa una instancia de la clase ResponseWrapper<TPetition, TResponse>.

Propiedades

- TPetition Petition `[get]`

  Retorna la petición.
- TResponse Response `[get]`

  Retorna la respuesta.

5.101.1. Descripción detallada

Clase para generar respuesta a un mensaje.

Parámetros del template

TPetition	.
TResponse	.

5.101.2. Documentación del constructor y destructor

5.101.2.1. ResponseWrapper() `ResponseWrapper< TPetition, TResponse >.ResponseWrapper ( TPetition petition, TResponse response )`

Inicializa una instancia de la clase ResponseWrapper<TPetition, TResponse>.

Parámetros

petition	Mensaje al que se responde.
response	La respuesta.

5.101.3. Documentación de propiedades

5.101.3.1. Petition `TPetition ResponseWrapper< TPetition, TResponse >.Petition [get]`

Retorna la petición.

5.101.3.2. Response `TResponse ResponseWrapper< TPetition, TResponse >.Response [get]`

Retorna la respuesta.

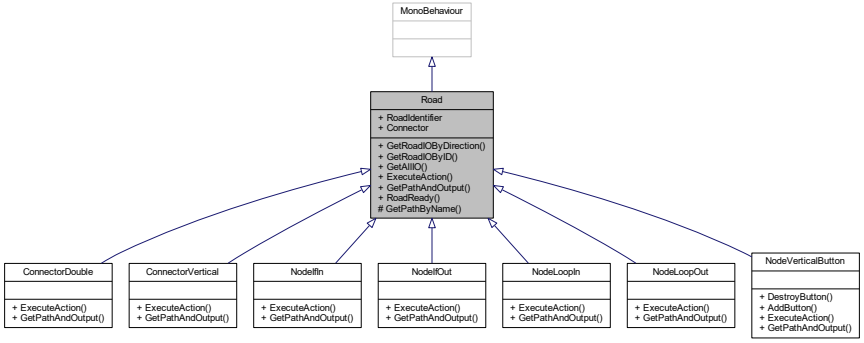
La documentación para esta clase fue generada a partir del siguiente fichero:

- Assets/Scripts/MessageHub/ResponseWrapper.cs

5.102. Referencia de la Clase Road

Clase padre de las carreteras.

Diagrama de herencias de Road



Métodos públicos

- List< RoadIO > GetRoadIOByDirection (IODirection direction)

  Retorna la IO por su dirección.
- RoadIO GetRoadIOByID (string ioID)

  Retorna IO por su ID.
- RoadIO[ ] GetAllIO ()

  Retorna toda la IO.
- abstract void ExecuteAction (in string[ ] args)

  Ejecuta una acción en base a una lista de argumentos.
- abstract bool GetPathAndOutput (in RoadInput input, out Path path, out RoadOutput output)

  Dado un input retorna el camino que inicia en él y el output en el que termina.
- bool RoadReady ()

  ¿Está la carretera preparada?

Métodos protegidos

- bool GetPathByName (in string name, out Path path)

  Retorna un camino por su nombre.

Propiedades

- string RoadIdentifier `[get]`

  En este caso el identificador es el nombre del objeto.
- bool Connector `[get]`

  Devuelve si es conector o no.

5.102.1. Descripción detallada

Clase padre de las carreteras.

5.102.2. Documentación de las funciones miembro

5.102.2.1. ExecuteAction() `abstract void Road.ExecuteAction ( in string[] args ) [pure virtual]`

Ejecuta una acción en base a una lista de argumentos.

Parámetros

args	Los argumentos.

Implementado en NodeVerticalButton, NodeLoopIn, ConnectorDouble, NodeIfIn, NodeIfOut, NodeLoopOut y ConnectorVertical.

5.102.2.2. GetAllIO() `RoadIO [] Road.GetAllIO ( )`

Retorna toda la IO.

Devuelve

Array de IO.

5.102.2.3. GetPathAndOutput() `abstract bool Road.GetPathAndOutput ( in RoadInput input, out Path path, out RoadOutput output ) [pure virtual]`

Dado un input retorna el camino que inicia en él y el output en el que termina.

Parámetros

input	El input RoadInput.
path	El camino Path.
output	El output RoadOutput.

Devuelve

True si hay camino, false si no.

Implementado en NodeVerticalButton, NodeLoopIn, NodeIfIn, ConnectorDouble, NodeIfOut, NodeLoopOut y ConnectorVertical.

5.102.2.4. GetPathByName()

```
bool Road.GetPathByName (
    in string name,
    out Path path )  [protected]
```

Retorna un camino por su nombre.

Parámetros

name	El nombre del camino.
path	El camino en si.

Devuelve

True si hay camino, false si no.

5.102.2.5. GetRoadIOByDirection()

```
List<RoadIO> Road.GetRoadIOByDirection (
    IODirection direction )
```

Retorna la IO por su dirección.

Parámetros

direction	La dirección IODirection.

Devuelve

La lista de IO.

5.102.2.6. GetRoadIOByID() RoadIO Road.GetRoadIOByID (string *ioID*)

Retorna IO por su ID.

Parámetros

ioID	El ID.

Devuelve

La IO encontrada.

5.102.2.7. RoadReady() `bool Road.RoadReady ( )`

¿Está la carretera preparada?

Devuelve

True si lo está, false si no.

5.102.3. Documentación de propiedades

5.102.3.1. Connector `bool Road.Connector [get]`

Devuelve si es conector o no.

5.102.3.2. RoadIdentifier `string Road.RoadIdentifier [get]`

En este caso el identificador es el nombre del objeto.

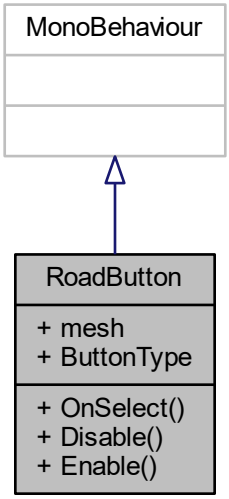
La documentación para esta clase fue generada a partir del siguiente fichero:

- Assets/Scripts/Roads/Road.cs

5.103. Referencia de la Clase RoadButton

Botón para colocar una carretera.

Diagrama de herencias de RoadButton



Métodos públicos

- void OnSelect ()
  OnSelect.
- void Disable ()
  Desactiva el botón.
- void Enable ()
  Activa el botón.

Atributos públicos

- GameObject mesh
  Mesh del botón.

Propiedades

- Buttons ButtonType `[get]`
  Retorna el tipo del botón.

5.103.1. Descripción detallada

Botón para colocar una carretera.

5.103.2. Documentación de las funciones miembro

5.103.2.1. Disable() `void RoadButton.Disable ( )`

Desactiva el botón.

5.103.2.2. Enable() `void RoadButton.Enable ( )`

Activa el botón.

5.103.2.3. OnSelect() `void RoadButton.OnSelect ( )`

OnSelect.

5.103.3. Documentación de los datos miembro

5.103.3.1. mesh `GameObject RoadButton.mesh`

Mesh del botón.

5.103.4. Documentación de propiedades

5.103.4.1. ButtonType `Buttons RoadButton.ButtonType [get]`

Retorna el tipo del botón.

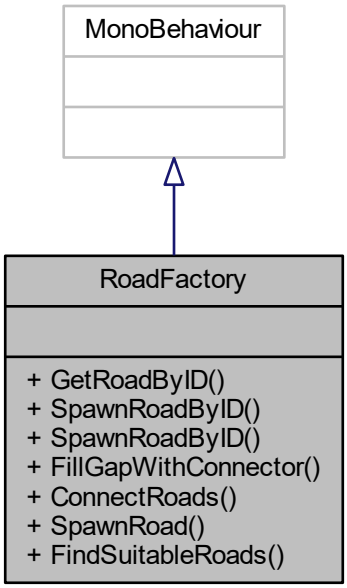
La documentación para esta clase fue generada a partir del siguiente fichero:

- Assets/Scripts/Roads/RoadButton.cs

5.104. Referencia de la Clase RoadFactory

Define la clase RoadFactory la cual genera carreteras de acuerdo a una serie de condiciones.

Diagrama de herencias de RoadFactory



Métodos públicos

- bool GetRoadByID (in string id, out Road road)

  Devuelve una carretera usando su identificador.
- bool SpawnRoadByID (in string id, in List< RoadIO > ioToMatch, out Road road, out Dictionary< string, string > connectionsR_C)

  Genera una carretera por su ID si encaja con la IO que se le pasa.
- bool SpawnRoadByID (in string id, in List< RoadIO > ioToMatch, in List< RoadIO > ioToMatch2, out Road road, out Dictionary< string, string > connectionsR1_Connector, out Dictionary< string, string > connectionsR2_Connector)

  Genera una carretera por su id si encaja en el hueco entre dos carreteras.
- bool FillGapWithConnector (in List< RoadIO > ioToMatch, in List< RoadIO > ioToMatch2, out Road road, out Dictionary< string, string > connectionsR1_Connector, out Dictionary< string, string > connections↩ R2_Connector)

  Dado un hueco entre dos carreteras lo rellena con el conector que encaje.
- Road ConnectRoads (in Road[ ] connectors, in List< RoadIO > ioRoad1, in List< RoadIO > ioRoad2, float errorMargin, out Dictionary< string, string > connectionsR1_Connector, out Dictionary< string, string > connectionsR2_Connector)

  Dadas una lista de carreteras, una lista de io de la carretera a un lado de un hueco y una lista de io de la carretera al otro lado intenta conectarlas mediante una carretera de la lista anterior.
- bool SpawnRoad (in Road roadToSpawn, in List< RoadIO > ioToMatch, in float errorMargin, out Road spawnedRoad, out Dictionary< string, string > connections)

Intenta spawnear la carretera que se le pide en el conjunto de io que se le pasa.

- bool FindSuitableRoads (in Road[ ] roadList, in List< RoadIO > ioToMatch, in float errorMargin, out List< Road > validRoads, out List< Dictionary< string, string >> connectionsDictionary)

  Dada una lista de carreteras y una serie de IO, devuelve una lista de carreteras que encajan ahi y un dictionario con las conexiones (id de ioToMatch, id de conector)

5.104.1. Descripción detallada

Define la clase RoadFactory la cual genera carreteras de acuerdo a una serie de condiciones.

5.104.2. Documentación de las funciones miembro

5.104.2.1. ConnectRoads()

```
Road RoadFactory.ConnectRoads (
    in Road[] connectors,
    in List< RoadIO > ioRoad1,
    in List< RoadIO > ioRoad2,
    float errorMargin,
    out Dictionary< string, string > connectionsR1_Connector,
    out Dictionary< string, string > connectionsR2_Connector )
```

Dadas una lista de carreteras, una lista de io de la carretera a un lado de un hueco y una lista de io de la carretera al otro lado intenta conectarlas mediante una carretera de la lista anterior.

Parámetros

connectors	Lista de carreteras que se pueden usarRoad[ ].
ioRoad1	Lista de IO que tiene que satisfacer la carreteraList<RoadIO>.
ioRoad2	Lista de IO que tiene que satisfacer la carreteraList<RoadIO>.
errorMargin	Margen de error al conectar las carreterasfloat.
connectionsR1_Connector	Las conexiones que se han hechoDictionary<string, string>.
connectionsR2_Connector	Las conexiones que se han hechoDictionary<string, string>.

Devuelve

La carretera que ha satisfecho las condiciones Road.

5.104.2.2. FillGapWithConnector()

```
bool RoadFactory.FillGapWithConnector (
    in List< RoadIO > ioToMatch,
    in List< RoadIO > ioToMatch2,
    out Road road,
    out Dictionary< string, string > connectionsR1_Connector,
    out Dictionary< string, string > connectionsR2_Connector )
```

Dado un hueco entre dos carreteras lo rellena con el conector que encaje.

Parámetros

ioToMatch	Lista de IO que tiene que satisfacer la carreteraList<RoadIO>.
ioToMatch2	Lista de IO que tiene que satisfacer la carreteraList<RoadIO>.
road	La carretera generadaRoad.
connectionsR1_Connector	Las conexiones que se han hechoDictionary<string, string>.
connectionsR2_Connector	Las conexiones que se han hechoDictionary<string, string>.

Devuelve

True si ha funcionado, false si no bool.

5.104.2.3. FindSuitableRoads() `bool RoadFactory.FindSuitableRoads (`

```
in Road[] roadList,
in List< RoadIO > ioToMatch,
in float errorMargin,
out List< Road > validRoads,
out List< Dictionary< string, string >> connectionsDictionary )
```

Dada una lista de carreteras y una serie de IO, devuelve una lista de carreteras que encajan ahi y un dictionario con las conexiones (id de ioToMatch, id de conector)

Parámetros

roadList	Lista de carreterasRoad[ ].
ioToMatch	La lista de IO que satisfacerList<RoadIO>.
errorMargin	El margen de errorfloat.
validRoads	Lista de carreteras validasList<Road>.
connectionsDictionary	Las conexiones que se han hechoList<Dictionary<string, string>>.

Devuelve

True si ha funcionado, false si no bool.

5.104.2.4. GetRoadByID() `bool RoadFactory.GetRoadByID (`

```
in string id,
out Road road )
```

Devuelve una carretera usando su identificador.

Parámetros

id	El idstring.
road	La carreteraRoad.

Devuelve

True si se ha encontrado, false si no.

5.104.2.5. SpawnRoad()

```
bool RoadFactory.SpawnRoad (
    in Road roadToSpawn,
    in List< RoadIO > ioToMatch,
    in float errorMargin,
    out Road spawnedRoad,
    out Dictionary< string, string > connections )
```

Intenta spawnear la carretera que se le pide en el conjunto de io que se le pasa.

Parámetros

roadToSpawn	La carretera a generarRoad.
ioToMatch	La lista de IO que satisfacerList<RoadIO>.
errorMargin	El margen de errorfloat.
spawnedRoad	La carretera que se ha generadoRoad.
connections	Lista de conexiones que se han llevado a caboDictionary<string, string>.

Devuelve

True si ha funcionado, false si no bool.

5.104.2.6. SpawnRoadByID() [1/2]

```
bool RoadFactory.SpawnRoadByID (
    in string id,
    in List< RoadIO > ioToMatch,
    in List< RoadIO > ioToMatch2,
    out Road road,
    out Dictionary< string, string > connectionsR1_Connector,
    out Dictionary< string, string > connectionsR2_Connector )
```

Genera una carretera por su id si encaja en el hueco entre dos carreteras.

Parámetros

id	El id de la carretera a generar.
ioToMatch	Lista de IO que tiene que satisfacer la carretera.
ioToMatch2	Lista de IO que tiene que satisfacer la carretera.
road	La carretera generada.
connectionsR1_Connector	Las conexiones que se han hecho.
connectionsR2_Connector	Las conexiones que se han hecho.

Devuelve

True si ha funcionado, false si no.

5.104.2.7. SpawnRoadByID() [2/2]

```
bool RoadFactory.SpawnRoadByID (
    in string id,
    in List< RoadIO > ioToMatch,
    out Road road,
    out Dictionary< string, string > connectionsR_C )
```

Genera una carretera por su ID si encaja con la IO que se le pasa.

Parámetros

id	El idstring de la carretera a generar.
ioToMatch	Lista de IO que tiene que satisfacer la carreteraList<RoadIO>.
road	La carretera generada Road.
connectionsR_C	Las conexiones que se han hechoDictionary<string, string>.

Devuelve

True si ha funcionado, false si no bool.

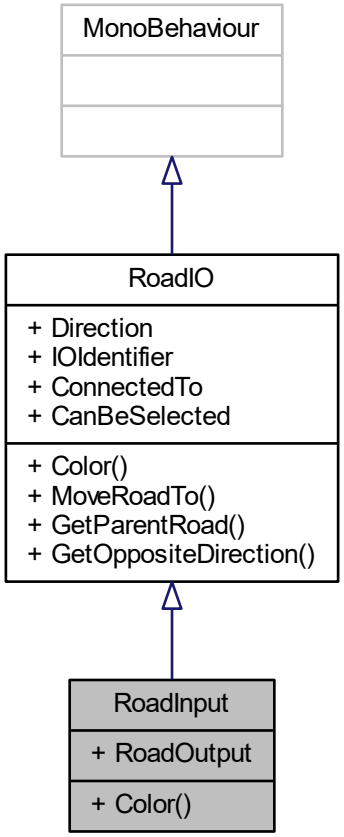
La documentación para esta clase fue generada a partir del siguiente fichero:

- Assets/Scripts/Roads/RoadFactory/RoadFactory.cs

5.105. Referencia de la Clase RoadInput

Marca un objeto como input de una carretera.

Diagrama de herencias de RoadInput



Métodos públicos

- override Color Color ()

  Retorna el color del gizmo.

Propiedades

- RoadOutput RoadOutput `[get, set]`

  Retorna el output asociado, si tiene.

Otros miembros heredados

5.105.1. Descripción detallada

Marca un objeto como input de una carretera.

5.105.2. Documentación de las funciones miembro

5.105.2.1. Color() override Color RoadInput.Color ( ) [virtual]

Retorna el color del gizmo.

Devuelve

El color.

Implementa RoadIO.

5.105.3. Documentación de propiedades

5.105.3.1. RoadOutput RoadOutput RoadInput.RoadOutput [get], [set]

Retorna el output asociado, si tiene.

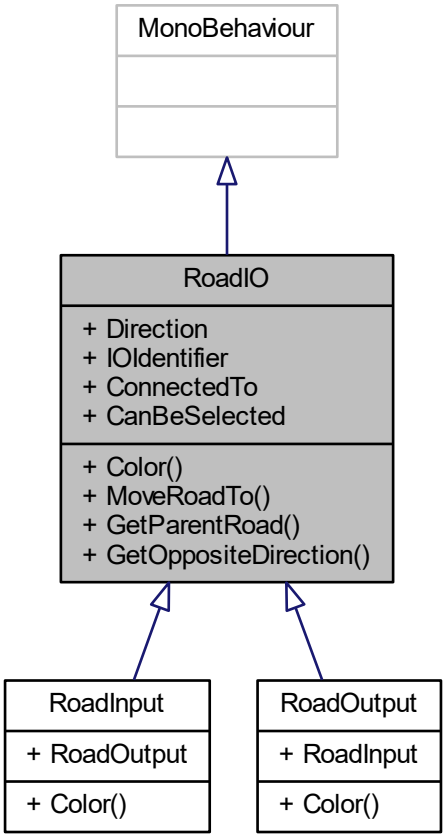
La documentación para esta clase fue generada a partir del siguiente fichero:

- Assets/Scripts/Roads/RoadIO/RoadInput.cs

5.106. Referencia de la Clase RoadIO

Entradas y salidas de una carretera.

Diagrama de herencias de RoadIO



Tipos públicos

- enum IODirection {
  Forward = 0, **Back** = 1, **Left** = 2, **Right** = 3,
  Undefined }

  Dirección a la que apunta la instancia.

Métodos públicos

- abstract Color Color ()

  Retorna el color del gizmo.
- void MoveRoadTo (in Vector3 newPos)

  Mueve la carretera a un punto usando esta IO como centro.
- Road GetParentRoad ()

  Retorna la carretera padre.

Métodos públicos estáticos

- static IODirection GetOppositeDirection (IODirection direction)

  Calcula la dirección opuesta a otra.

Propiedades

- IODirection Direction `[get]`

  Retorna la dirección a la que apunta la instancia.
- string IOIdentifier `[get]`

  Retorna el ID.
- RoadIO ConnectedTo `[get, set]`

  Retorna a qué está conectada esta IO o la conecta con otra.
- bool CanBeSelected `[get]`

  Retorna si puede ser seleccionada.

5.106.1. Descripción detallada

Entradas y salidas de una carretera.

5.106.2. Documentación de las enumeraciones miembro de la clase

5.106.2.1. IODirection `enum RoadIO.IODirection [strong]`

Dirección a la que apunta la instancia.

5.106.3. Documentación de las funciones miembro

5.106.3.1. Color() `abstract Color RoadIO.Color ( ) [pure virtual]`

Retorna el color del gizmo.

Devuelve

Color.

Implementado en RoadInput y RoadOutput.

5.106.3.2. GetOppositeDirection() `static IODirection RoadIO.GetOppositeDirection ( IODirection direction ) [static]`

Calcula la dirección opuesta a otra.

Parámetros

direction	La dirección de entrada.

Devuelve

La IODirection de salida.

5.106.3.3. GetParentRoad() `Road RoadIO.GetParentRoad ( )`

Retorna la carretera padre.

Devuelve

La carretera padre.

5.106.3.4. MoveRoadTo() `void RoadIO.MoveRoadTo ( in Vector3 newPos )`

Mueve la carretera a un punto usando esta IO como centro.

Parámetros

newPos	La nueva posición.

5.106.4. Documentación de propiedades

5.106.4.1. CanBeSelected `bool RoadIO.CanBeSelected [get]`

Retorna si puede ser seleccionada.

5.106.4.2. ConnectedTo `RoadIO RoadIO.ConnectedTo [get], [set]`

Retorna a qué está conectada esta IO o la conecta con otra.

5.106.4.3. Direction `IODirection RoadIO.Direction [get]`

Retorna la dirección a la que apunta la instancia.

5.106.4.4. IOIdentifier `string RoadIO.IOIdentifier [get]`

Retorna el ID.

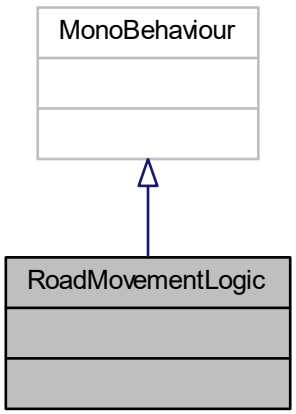
La documentación para esta clase fue generada a partir del siguiente fichero:

- Assets/Scripts/Roads/RoadIO/RoadIO.cs

5.107. Referencia de la Clase RoadMovementLogic

Esta clase RoadMovementLogic contiene la lógica para que el robot pequeño se mueva por la carretera.

Diagrama de herencias de RoadMovementLogic



5.107.1. Descripción detallada

Esta clase RoadMovementLogic contiene la lógica para que el robot pequeño se mueva por la carretera.

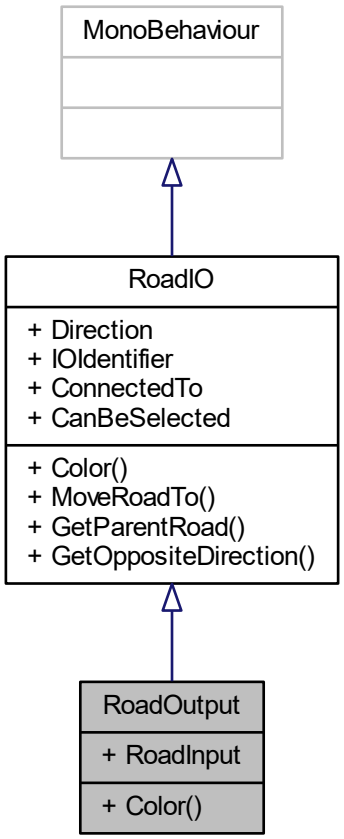
La documentación para esta clase fue generada a partir del siguiente fichero:

- Assets/Scripts/GameLogic/RoadMovementLogic.cs

5.108. Referencia de la Clase RoadOutput

Marca un objeto como output de una carretera.

Diagrama de herencias de RoadOutput



Métodos públicos

- override Color Color ()

  Retorna el color del gizmo.

Propiedades

- RoadInput RoadInput `[get, set]`

  Retorna el input asociado, si tiene.

Otros miembros heredados

5.108.1. Descripción detallada

Marca un objeto como output de una carretera.

5.108.2. Documentación de las funciones miembro

5.108.2.1. Color() `override Color RoadOutput.Color ( ) [virtual]`

Retorna el color del gizmo.

Devuelve

El color.

Implementa RoadIO.

5.108.3. Documentación de propiedades

5.108.3.1. RoadInput `RoadInput RoadOutput.RoadInput [get], [set]`

Retorna el input asociado, si tiene.

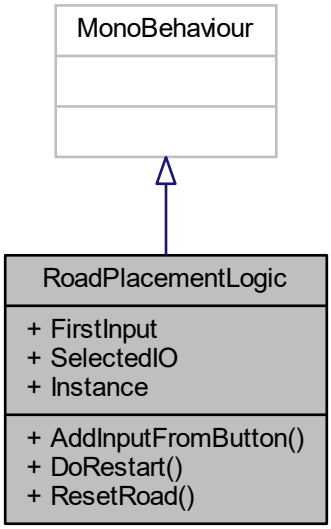
La documentación para esta clase fue generada a partir del siguiente fichero:

- Assets/Scripts/Roads/RoadIO/RoadOutput.cs

5.109. Referencia de la Clase RoadPlacementLogic

La clase RoadPlacementLogic se encarga de formar la carretera apropiada de acuerdo a las ordenes del usuario.

Diagrama de herencias de RoadPlacementLogic



Métodos públicos

- void AddInputFromButton (Buttons buttonIndex)

  Ejecuta la acción correspondiente a un botón.
- void DoRestart ()

  Lógica del botón restart.
- void ResetRoad ()

  Resetea la carretera a su estado original.

Propiedades

- RoadIO FirstInput `[get]`

  Primer input de la carretera.
- RoadIO SelectedIO `[get, set]`

  IO seleccionada.
- static RoadPlacementLogic Instance `[get]`

  Retorna la instancia de la clase.

5.109.1. Descripción detallada

La clase RoadPlacementLogic se encarga de formar la carretera apropiada de acuerdo a las ordenes del usuario.

5.109.2. Documentación de las funciones miembro

5.109.2.1. AddInputFromButton() `void RoadPlacementLogic.AddInputFromButton ( Buttons buttonIndex )`

Ejecuta la acción correspondiente a un botón.

Parámetros

buttonIndex	El botón pulsado Buttons.

5.109.2.2. DoRestart() `void RoadPlacementLogic.DoRestart ( )`

Lógica del botón restart.

5.109.2.3. ResetRoad() `void RoadPlacementLogic.ResetRoad ( )`

Resetea la carretera a su estado original.

5.109.3. Documentación de propiedades

5.109.3.1. FirstInput `RoadIO RoadPlacementLogic.FirstInput [get]`

Primer input de la carretera.

5.109.3.2. Instance `RoadPlacementLogic RoadPlacementLogic.Instance [static], [get]`

Retorna la instancia de la clase.

5.109.3.3. SelectedIO `RoadIO RoadPlacementLogic.SelectedIO [get], [set]`

IO seleccionada.

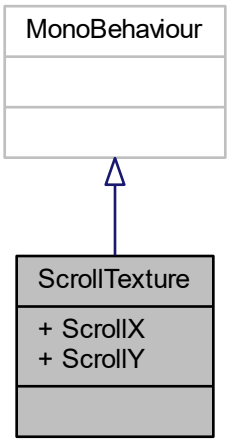
La documentación para esta clase fue generada a partir del siguiente fichero:

- Assets/Scripts/GameLogic/RoadPlacementLogic.cs

5.110. Referencia de la Clase ScrollTexture

Hace scroll continuo a una textura.

Diagrama de herencias de ScrollTexture



Atributos públicos

- float ScrollX = 0.5f
  Scroll en X.
- float ScrollY = 0.5f
  Scroll en Y.

5.110.1. Descripción detallada

Hace scroll continuo a una textura.

5.110.2. Documentación de los datos miembro

5.110.2.1. ScrollX `float ScrollTexture.ScrollX = 0.5f`

Scroll en X.

5.110.2.2. ScrollY `float ScrollTexture.ScrollY = 0.5f`

Scroll en Y.

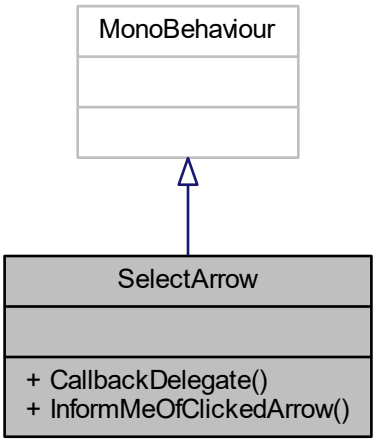
La documentación para esta clase fue generada a partir del siguiente fichero:

- Assets/Scripts/Visual/ScrollTexture.cs

5.111. Referencia de la Clase SelectArrow

Clase de las flechas de selección de nivel.

Diagrama de herencias de SelectArrow



Métodos públicos

- delegate void CallbackDelegate ()

  Definición del delegado de métodos a los que llamar cuando se hace click.
- void InformMeOfClickedArrow (CallbackDelegate action)

  Permite a otros objetos suscribirse a este para ejecutar el callback cuando se pulse.

5.111.1. Descripción detallada

Clase de las flechas de selección de nivel.

5.111.2. Documentación de las funciones miembro

5.111.2.1. CallbackDelegate() `delegate void SelectArrow.CallbackDelegate ( )`

Definición del delegado de métodos a los que llamar cuando se hace click.

5.111.2.2. InformMeOfClickedArrow() `void SelectArrow.InformMeOfClickedArrow ( CallbackDelegate action )`

Permite a otros objetos suscribirse a este para ejecutar el callback cuando se pulse.

Parámetros

action	El método al que llamar cuando se pulse en esta flecha.

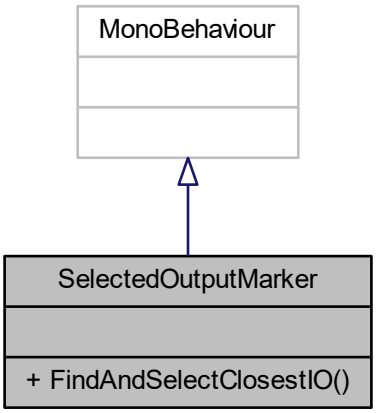
La documentación para esta clase fue generada a partir del siguiente fichero:

- Assets/Scripts/MapMenu/SelectArrow.cs

5.112. Referencia de la Clase SelectedOutputMarker

Clase de la flecha para seleccionar carretera.

Diagrama de herencias de SelectedOutputMarker



Métodos públicos

- void FindAndSelectClosestIO ()

  Busca y selecciona la IO más cercana.

5.112.1. Descripción detallada

Clase de la flecha para seleccionar carretera.

5.112.2. Documentación de las funciones miembro

5.112.2.1. FindAndSelectClosestIO() `void SelectedOutputMarker.FindAndSelectClosestIO ( )`

Busca y selecciona la IO más cercana.

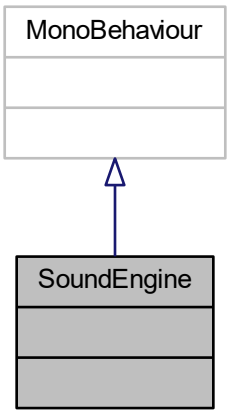
La documentación para esta clase fue generada a partir del siguiente fichero:

- Assets/Scripts/Roads/SelectedOutputMarker.cs

5.113. Referencia de la Clase SoundEngine

Pequeña clase con métodos relativos al sonido.

Diagrama de herencias de SoundEngine



5.113.1. Descripción detallada

Pequeña clase con métodos relativos al sonido.

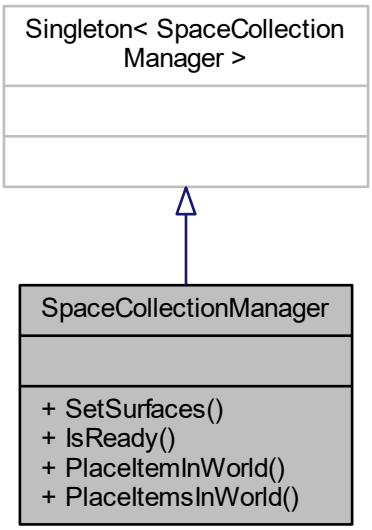
La documentación para esta clase fue generada a partir del siguiente fichero:

- Assets/Scripts/Sound/SoundEngine.cs

5.114. Referencia de la Clase SpaceCollectionManager

Llamado por PlaySpaceManager después de que los planos hayan sido generados a partir de un Spatial Mapping Mesh. Esta clase puede colocar objetos con el componente Placeable de forma que queden cercanos al usuario.

Diagrama de herencias de SpaceCollectionManager



Métodos públicos

- void SetSurfaces (List< GameObject > horizontalSurfaces, List< GameObject > verticalSurfaces)

  Genera una colección de Placeables en el mundo y los coloca en planos adecuados.
- bool **IsReady** ()
- bool PlaceItemInWorld (GameObject spaceObjectPrefabs)

  Intenta colocar un objeto en el escenario.
- bool **PlaceItemsInWorld** (List< GameObject > spaceObjectPrefabs)

5.114.1. Descripción detallada

Llamado por PlaySpaceManager después de que los planos hayan sido generados a partir de un Spatial Mapping Mesh. Esta clase puede colocar objetos con el componente Placeable de forma que queden cercanos al usuario.

5.114.2. Documentación de las funciones miembro

5.114.2.1. PlaceItemInWorld() `bool SpaceCollectionManager.PlaceItemInWorld ( GameObject spaceObjectPrefabs )`

Intenta colocar un objeto en el escenario.

Parámetros

spaceObjectPrefabs	Objeto a colocar.

Devuelve

True si ha conseguido colocarlo, false si no.

5.114.2.2. SetSurfaces()

```
void SpaceCollectionManager.SetSurfaces (
    List< GameObject > horizontalSurfaces,
    List< GameObject > verticalSurfaces )
```

Genera una colección de Placeables en el mundo y los coloca en planos adecuados.

Parámetros

horizontalSurfaces	Planos horizontales.
verticalSurfaces	Planos verticales.

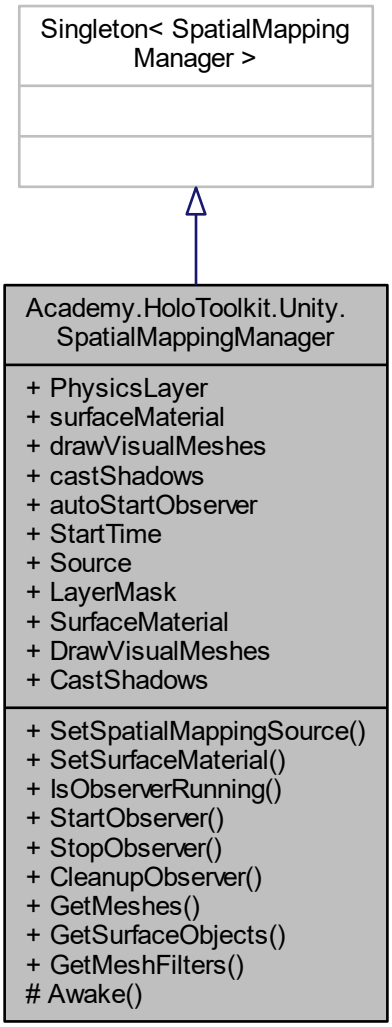
La documentación para esta clase fue generada a partir del siguiente fichero:

- Assets/Scripts/Hololens/SpaceCollectionManager.cs

5.115. Referencia de la Clase Academy.HoloToolkit.Unity.SpatialMappingManager

The SpatialMappingManager class allows applications to use a SurfaceObserver or a stored Spatial Mapping mesh (loaded from a file). When an application loads a mesh file, the SurfaceObserver is stopped. Calling StartObserver() clears the stored mesh and enables real-time SpatialMapping updates.

Diagrama de herencias de Academy.HoloToolkit.Unity.SpatialMappingManager



Métodos públicos

- void SetSpatialMappingSource (SpatialMappingSource mappingSource)

  Sets the source of surface information.
- void SetSurfaceMaterial (Material surfaceMaterial)

  Sets the material used by all Spatial Mapping meshes.
- bool IsObserverRunning ()

  Checks to see if the SurfaceObserver is currently running.
- void StartObserver ()

  Instructs the SurfaceObserver to start updating the SpatialMapping mesh.
- void StopObserver ()

  Instructs the SurfaceObserver to stop updating the SpatialMapping mesh.
- void CleanupObserver ()

Instructs the SurfaceObserver to stop and cleanup all meshes.

- List< Mesh > GetMeshes ()

  Gets all meshes that are associated with the SpatialMapping mesh.

- List< SpatialMappingSource.SurfaceObject > GetSurfaceObjects ()

  Gets all the surface objects associated with the Spatial Mapping mesh.

- List< MeshFilter > GetMeshFilters ()

  Gets all Mesh Filter objects associated with the Spatial Mapping mesh.

Atributos públicos

- int **PhysicsLayer** = 31
- Material **surfaceMaterial**
- bool **drawVisualMeshes** = false
- bool **castShadows** = false
- bool **autoStartObserver** = true

Métodos protegidos

- void **Awake** ()

Propiedades

- float StartTime `[get]`

  Time when StartObserver() was called.

- SpatialMappingSource Source `[get]`

  The current source of spatial mapping data.

- int LayerMask `[get]`

  Returns the layer as a bit mask.

- Material SurfaceMaterial `[get, set]`

  The material to use when rendering surfaces.

- bool DrawVisualMeshes `[get, set]`

  Specifies whether or not the SpatialMapping meshes are to be rendered.

- bool CastShadows `[get, set]`

  Specifies whether or not the SpatialMapping meshes can cast shadows.

5.115.1. Descripción detallada

The SpatialMappingManager class allows applications to use a SurfaceObserver or a stored Spatial Mapping mesh (loaded from a file). When an application loads a mesh file, the SurfaceObserver is stopped. Calling StartObserver() clears the stored mesh and enables real-time SpatialMapping updates.

5.115.2. Documentación de las funciones miembro

5.115.2.1. CleanupObserver() `void Academy.HoloToolkit.Unity.SpatialMappingManager.Cleanup↩ Observer ( )`

Instructs the SurfaceObserver to stop and cleanup all meshes.

5.115.2.2. GetMeshes() `List<Mesh> Academy.HoloToolkit.Unity.SpatialMappingManager.GetMeshes ( )`

Gets all meshes that are associated with the SpatialMapping mesh.

Devuelve

Collection of Mesh objects representing the SpatialMapping mesh.

5.115.2.3. GetMeshFilters() `List<MeshFilter> Academy.HoloToolkit.Unity.SpatialMappingManager.↩ GetMeshFilters ( )`

Gets all Mesh Filter objects associated with the Spatial Mapping mesh.

Devuelve

Collection of Mesh Filter objects.

5.115.2.4. GetSurfaceObjects() `List<SpatialMappingSource.SurfaceObject> Academy.HoloToolkit.↩ Unity.SpatialMappingManager.GetSurfaceObjects ( )`

Gets all the surface objects associated with the Spatial Mapping mesh.

Devuelve

Collection of SurfaceObjects.

5.115.2.5. IsObserverRunning() `bool Academy.HoloToolkit.Unity.SpatialMappingManager.IsObserver↩ Running ( )`

Checks to see if the SurfaceObserver is currently running.

Devuelve

True, if the observer state is running.

5.115.2.6. SetSpatialMappingSource() `void Academy.HoloToolkit.Unity.SpatialMappingManager.Set↩ SpatialMappingSource (`

```
            SpatialMappingSource mappingSource )
```

Sets the source of surface information.

Parámetros

mappingSource	The source to switch to. Null means return to the live stream if possible.

5.115.2.7. SetSurfaceMaterial()

`void Academy.HoloToolkit.Unity.SpatialMappingManager.SetSurface↩ Material (`
`Material surfaceMaterial )`

Sets the material used by all Spatial Mapping meshes.

Parámetros

surfaceMaterial	New material to apply.

5.115.2.8. StartObserver()

`void Academy.HoloToolkit.Unity.SpatialMappingManager.StartObserver ( )`

Instructs the SurfaceObserver to start updating the SpatialMapping mesh.

5.115.2.9. StopObserver()

`void Academy.HoloToolkit.Unity.SpatialMappingManager.StopObserver ( )`

Instructs the SurfaceObserver to stop updating the SpatialMapping mesh.

5.115.3. Documentación de propiedades

5.115.3.1. CastShadows

`bool Academy.HoloToolkit.Unity.SpatialMappingManager.CastShadows [get], [set]`

Specifies whether or not the SpatialMapping meshes can cast shadows.

5.115.3.2. DrawVisualMeshes

`bool Academy.HoloToolkit.Unity.SpatialMappingManager.DrawVisual↩ Meshes [get], [set]`

Specifies whether or not the SpatialMapping meshes are to be rendered.

5.115.3.3. LayerMask `int Academy.HoloToolkit.Unity.SpatialMappingManager.LayerMask [get]`

Returns the layer as a bit mask.

5.115.3.4. Source `SpatialMappingSource Academy.HoloToolkit.Unity.SpatialMappingManager.Source [get]`

The current source of spatial mapping data.

5.115.3.5. StartTime `float Academy.HoloToolkit.Unity.SpatialMappingManager.StartTime [get]`

Time when StartObserver() was called.

5.115.3.6. SurfaceMaterial `Material Academy.HoloToolkit.Unity.SpatialMappingManager.Surface↩ Material [get], [set]`

The material to use when rendering surfaces.

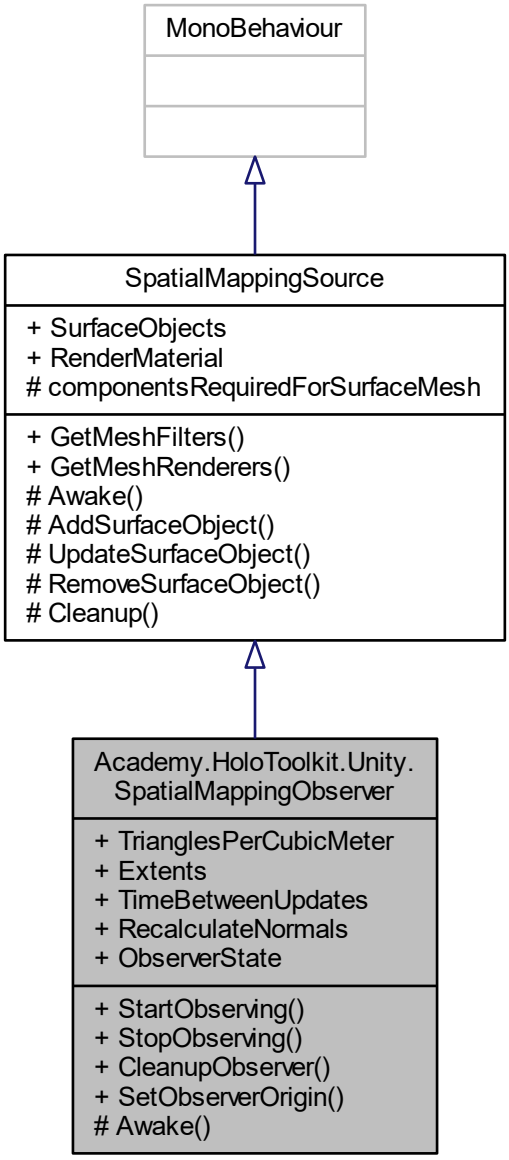
La documentación para esta clase fue generada a partir del siguiente fichero:

- Assets/HoloToolkit-SpatialMapping-230/SpatialMapping/Scripts/SpatialMappingManager.cs

5.116. Referencia de la Clase Academy.HoloToolkit.Unity.SpatialMappingObserver

The SpatialMappingObserver class encapsulates the SurfaceObserver into an easy to use object that handles managing the observed surfaces and the rendering of surface geometry.

Diagrama de herencias de Academy.HoloToolkit.Unity.SpatialMappingObserver



Métodos públicos

- void StartObserving ()

  Starts the Surface Observer.
- void StopObserving ()

  Stops the Surface Observer.
- void CleanupObserver ()

  Cleans up all memory and objects associated with the observer.
- bool SetObserverOrigin (Vector3 origin)

  Can be called to override the default origin for the observed volume. Can only be called while observer has been started.

Atributos públicos

- float **TrianglesPerCubicMeter** = 500f
- Vector3 **Extents** = Vector3.one ∗ 10.0f
- float **TimeBetweenUpdates** = 3.5f
- bool **RecalculateNormals** = false

Métodos protegidos

- override void **Awake** ()

Propiedades

- ObserverStates ObserverState `[get]`

  Indicates the current state of the Surface Observer.

Eventos

- SurfaceObserver.SurfaceChangedDelegate SurfaceChanged

  Event for hooking when surfaces are changed.
- SurfaceObserver.SurfaceDataReadyDelegate DataReady

  Event for hooking when the data for a surface is ready.

Otros miembros heredados

5.116.1. Descripción detallada

The SpatialMappingObserver class encapsulates the SurfaceObserver into an easy to use object that handles managing the observed surfaces and the rendering of surface geometry.

5.116.2. Documentación de las funciones miembro

5.116.2.1. CleanupObserver() `void Academy.HoloToolkit.Unity.SpatialMappingObserver.Cleanup↩ Observer ( )`

Cleans up all memory and objects associated with the observer.

5.116.2.2. SetObserverOrigin() `bool Academy.HoloToolkit.Unity.SpatialMappingObserver.SetObserver↩ Origin (`

```
Vector3 origin )
```

Can be called to override the default origin for the observed volume. Can only be called while observer has been started.

5.116.2.3. StartObserving() void Academy.HoloToolkit.Unity.SpatialMappingObserver.StartObserving ( )

Starts the Surface Observer.

5.116.2.4. StopObserving() void Academy.HoloToolkit.Unity.SpatialMappingObserver.StopObserving ( )

Stops the Surface Observer.

Sets the Surface Observer state to ObserverStates.Stopped.

5.116.3. Documentación de propiedades

5.116.3.1. ObserverState ObserverStates Academy.HoloToolkit.Unity.SpatialMappingObserver.↩ ObserverState [get]

Indicates the current state of the Surface Observer.

5.116.4. Documentación de los eventos

5.116.4.1. DataReady SurfaceObserver.SurfaceDataReadyDelegate Academy.HoloToolkit.Unity.↩ SpatialMappingObserver.DataReady

Event for hooking when the data for a surface is ready.

5.116.4.2. SurfaceChanged SurfaceObserver.SurfaceChangedDelegate Academy.HoloToolkit.Unity.↩ SpatialMappingObserver.SurfaceChanged

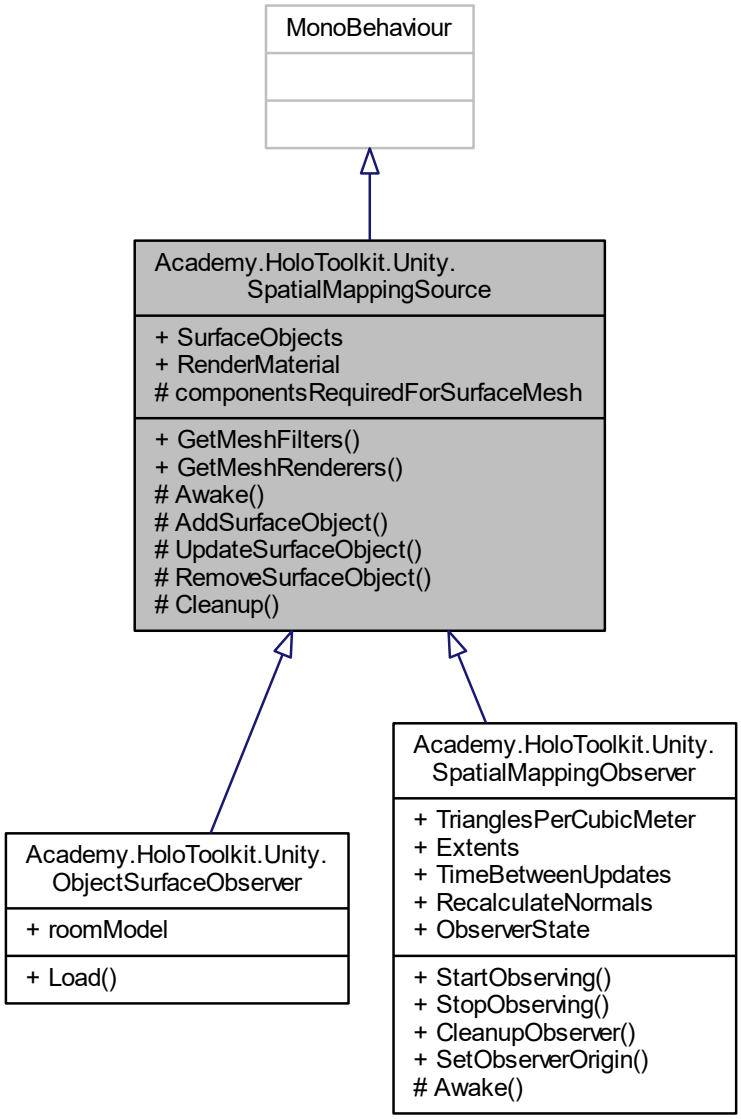
Event for hooking when surfaces are changed.

La documentación para esta clase fue generada a partir del siguiente fichero:

- Assets/HoloToolkit-SpatialMapping-230/SpatialMapping/Scripts/SpatialMappingObserver.cs

5.117. Referencia de la Clase Academy.HoloToolkit.Unity.SpatialMappingSource

Diagrama de herencias de Academy.HoloToolkit.Unity.SpatialMappingSource



Clases

- struct SurfaceObject

  Surface object

Métodos públicos

- virtual List< MeshFilter > GetMeshFilters ()

  Gets all mesh filters that have a valid mesh.
- virtual List< MeshRenderer > GetMeshRenderers ()

  Gets all mesh renderers that have been created.

Métodos protegidos

- virtual void **Awake** ()
- GameObject AddSurfaceObject (Mesh mesh, string objectName, Transform parentObject, int meshID=0)

  Creates a new surface game object.
- void UpdateSurfaceObject (GameObject gameObject, int meshID)

  Updates an existing surface object.
- void RemoveSurfaceObject (GameObject surface, bool removeAndDestroy=true)

  Removes and optionally destroys the specified surfaceObject that we've created
- void Cleanup ()

  Cleans up references to objects that we have created.

Atributos protegidos

- Type[ ] componentsRequiredForSurfaceMesh

  When a mesh is created we will need to create a game object with a minimum set of components to contain the mesh. These are the required component types.

Propiedades

- List< SurfaceObject > SurfaceObjects `[get]`

  Collection of surface objects that have been created for this spatial mapping source.
- virtual Material RenderMaterial `[get]`

  Material to use for rendering the mesh

5.117.1. Documentación de las funciones miembro

5.117.1.1. AddSurfaceObject() `GameObject Academy.HoloToolkit.Unity.SpatialMappingSource.AddSurfaceObject (`

```
            Mesh mesh,
            string objectName,
            Transform parentObject,
            int meshID = 0 )  [protected]
```

Creates a new surface game object.

Parámetros

mesh	The mesh to attach. Can be null.
objectName	What to name this object.
parentObject	What to parent this object to.
meshID	Optional user specified ID for the mesh.

Devuelve

The newly created game object.

5.117.1.2. Cleanup() `void Academy.HoloToolkit.Unity.SpatialMappingSource.Cleanup ( ) [protected]`

Cleans up references to objects that we have created.

5.117.1.3. GetMeshFilters() `virtual List<MeshFilter> Academy.HoloToolkit.Unity.SpatialMapping↩ Source.GetMeshFilters ( ) [virtual]`

Gets all mesh filters that have a valid mesh.

Devuelve

A list of filters, each with a mesh containing at least one triangle.

5.117.1.4. GetMeshRenderers() `virtual List<MeshRenderer> Academy.HoloToolkit.Unity.Spatial↩ MappingSource.GetMeshRenderers ( ) [virtual]`

Gets all mesh renderers that have been created.

Devuelve

5.117.1.5. RemoveSurfaceObject() `void Academy.HoloToolkit.Unity.SpatialMappingSource.Remove↩ SurfaceObject ( GameObject surface, bool removeAndDestroy = true ) [protected]`

Removes and optionally destroys the specified surfaceObject that we've created

Parámetros

surface	The surface game object
removeAndDestroy	If true, the surface will be removed and destroyed. If false, it will only be removed.

5.117.1.6. UpdateSurfaceObject() `void Academy.HoloToolkit.Unity.SpatialMappingSource.Update↩ SurfaceObject ( GameObject gameObject, int meshID ) [protected]`

Updates an existing surface object.

Parámetros

gameObject	Game object reference to the surfaceObject.
meshID	User specified ID for the mesh.

Devuelve

True if successful

5.117.2. Documentación de los datos miembro

5.117.2.1. componentsRequiredForSurfaceMesh `Type [] Academy.HoloToolkit.Unity.SpatialMappingSource.componentsRequiredForSurfaceMesh [protected]`

Valor inicial:

```
=
        {
            typeof(MeshFilter),
            typeof(MeshRenderer),
            typeof(MeshCollider)
        }
```

When a mesh is created we will need to create a game object with a minimum set of components to contain the mesh. These are the required component types.

5.117.3. Documentación de propiedades

5.117.3.1. RenderMaterial `virtual Material Academy.HoloToolkit.Unity.SpatialMappingSource.RenderMaterial [get], [protected]`

Material to use for rendering the mesh

5.117.3.2. SurfaceObjects `List<SurfaceObject> Academy.HoloToolkit.Unity.SpatialMappingSource.SurfaceObjects [get]`

Collection of surface objects that have been created for this spatial mapping source.

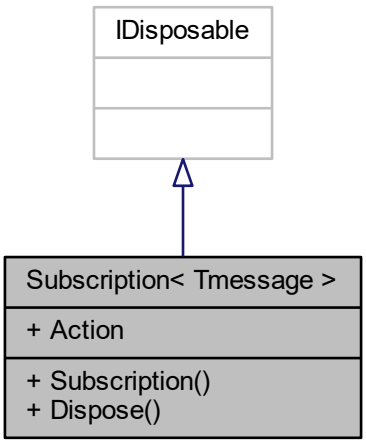
La documentación para esta clase fue generada a partir del siguiente fichero:

- Assets/HoloToolkit-SpatialMapping-230/SpatialMapping/Scripts/SpatialMappingSource.cs

5.118. Referencia de la plantilla de la Clase Subscription< Tmessage >

Clase Subscription<Tmessage> que simboliza la subscripción de una acción a un EventAggregator. Tomado y adaptado de https://www.c-sharpcorner.com/UploadFile/pranayamr/publisher-or-subscriber-pat

Diagrama de herencias de Subscription< Tmessage >



Métodos públicos

- **Subscription** (Action< Tmessage > action, EventAggregator eventAggregator)
- void **Dispose** ()

Propiedades

- Action< Tmessage > **Action** `[get]`

5.118.1. Descripción detallada

Clase Subscription<Tmessage> que simboliza la subscripción de una acción a un EventAggregator. Tomado y adaptado de https://www.c-sharpcorner.com/UploadFile/pranayamr/publisher-or-subscriber-pat

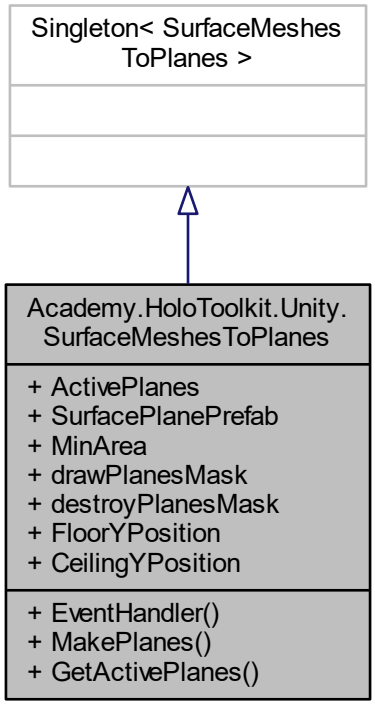
La documentación para esta clase fue generada a partir del siguiente fichero:

- Assets/Scripts/MessageHub/Subscription.cs

5.119. Referencia de la Clase Academy.HoloToolkit.Unity.SurfaceMeshesToPlanes

SurfaceMeshesToPlanes will find and create planes based on the meshes returned by the SpatialMappingManager's Observer.

Diagrama de herencias de Academy.HoloToolkit.Unity.SurfaceMeshesToPlanes



Métodos públicos

- delegate void EventHandler (object source, EventArgs args)

  Delegate which is called when the MakePlanesCompleted event is triggered.
- void MakePlanes ()

  Creates planes based on meshes gathered by the SpatialMappingManager's SurfaceObserver.
- List< GameObject > GetActivePlanes (PlaneTypes planeTypes)

  Gets all active planes of the specified type(s).

Atributos públicos

- List< GameObject > **ActivePlanes**
- GameObject **SurfacePlanePrefab**
- float **MinArea** = 0.025f
- PlaneTypes drawPlanesMask

  Determines which plane types should be rendered.
- PlaneTypes destroyPlanesMask = PlaneTypes.Unknown

  Determines which plane types should be discarded. Use this when the spatial mapping mesh is a better fit for the surface (ex: round tables).

Propiedades

- float FloorYPosition `[get]`

  Floor y value, which corresponds to the maximum horizontal area found below the user's head position. This value is reset by SurfaceMeshesToPlanes when the max floor plane has been found.
- float CeilingYPosition `[get]`

  Ceiling y value, which corresponds to the maximum horizontal area found above the user's head position. This value is reset by SurfaceMeshesToPlanes when the max ceiling plane has been found.

Eventos

- EventHandler MakePlanesComplete

  EventHandler which is triggered when the MakePlanesRoutine is finished.

5.119.1. Descripción detallada

SurfaceMeshesToPlanes will find and create planes based on the meshes returned by the SpatialMappingManager's Observer.

5.119.2. Documentación de las funciones miembro

5.119.2.1. EventHandler()

```
delegate void Academy.HoloToolkit.Unity.SurfaceMeshesToPlanes.EventHandler (
            object source,
            EventArgs args )
```

Delegate which is called when the MakePlanesCompleted event is triggered.

Parámetros

source	
args	

5.119.2.2. GetActivePlanes()

```
List<GameObject> Academy.HoloToolkit.Unity.SurfaceMeshesToPlanes.GetActivePlanes (
            PlaneTypes planeTypes )
```

Gets all active planes of the specified type(s).

Parámetros

planeTypes	A flag which includes all plane type(s) that should be returned.

Devuelve

A collection of planes that match the expected type(s).

5.119.2.3. MakePlanes() `void Academy.HoloToolkit.Unity.SurfaceMeshesToPlanes.MakePlanes ( )`

Creates planes based on meshes gathered by the SpatialMappingManager's SurfaceObserver.

5.119.3. Documentación de los datos miembro

5.119.3.1. destroyPlanesMask `PlaneTypes Academy.HoloToolkit.Unity.SurfaceMeshesToPlanes.↩ destroyPlanesMask = PlaneTypes.Unknown`

Determines which plane types should be discarded. Use this when the spatial mapping mesh is a better fit for the surface (ex: round tables).

5.119.3.2. drawPlanesMask `PlaneTypes Academy.HoloToolkit.Unity.SurfaceMeshesToPlanes.draw↩ PlanesMask`

Valor inicial:

```
=
            (PlaneTypes.Wall | PlaneTypes.Floor | PlaneTypes.Ceiling | PlaneTypes.Table)
```

Determines which plane types should be rendered.

5.119.4. Documentación de propiedades

5.119.4.1. CeilingYPosition `float Academy.HoloToolkit.Unity.SurfaceMeshesToPlanes.CeilingY↩ Position [get]`

Ceiling y value, which corresponds to the maximum horizontal area found above the user's head position. This value is reset by SurfaceMeshesToPlanes when the max ceiling plane has been found.

5.119.4.2. FloorYPosition `float Academy.HoloToolkit.Unity.SurfaceMeshesToPlanes.FloorYPosition [get]`

Floor y value, which corresponds to the maximum horizontal area found below the user's head position. This value is reset by SurfaceMeshesToPlanes when the max floor plane has been found.

5.119.5. Documentación de los eventos

5.119.5.1. MakePlanesComplete `EventHandler Academy.HoloToolkit.Unity.SurfaceMeshesToPlanes.↩ MakePlanesComplete`

EventHandler which is triggered when the MakePlanesRoutine is finished.

La documentación para esta clase fue generada a partir del siguiente fichero:

- Assets/HoloToolkit-SpatialMapping-230/SpatialMapping/Scripts/SurfaceMeshesToPlanes.cs

5.120. Referencia de la Estructura Academy.HoloToolkit.Unity.SpatialMappingSource.SurfaceObject

Surface object

Atributos públicos

- int **ID**
- int **UpdateID**
- GameObject **Object**
- MeshRenderer **Renderer**
- MeshFilter **Filter**

5.120.1. Descripción detallada

Surface object

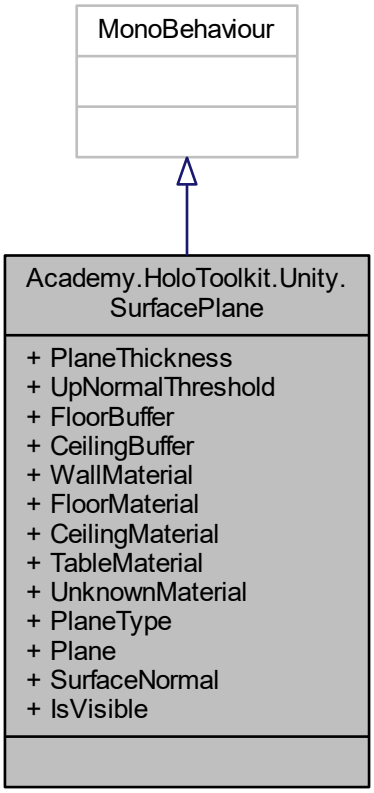
La documentación para esta estructura fue generada a partir del siguiente fichero:

- Assets/HoloToolkit-SpatialMapping-230/SpatialMapping/Scripts/SpatialMappingSource.cs

5.121. Referencia de la Clase Academy.HoloToolkit.Unity.SurfacePlane

The SurfacePlane class is used by SurfaceMeshesToPlanes to create different types of planes (walls, floors, tables, etc.) based on the Spatial Mapping data returned by the SpatialMappingManager's source. This script should be a component on the SufacePlane prefab, which is used by SurfaceMeshesToPlanes.

Diagrama de herencias de Academy.HoloToolkit.Unity.SurfacePlane



Atributos públicos

- float **PlaneThickness** = 0.01f
- float **UpNormalThreshold** = 0.9f
- float **FloorBuffer** = 0.1f
- float **CeilingBuffer** = 0.1f
- Material **WallMaterial**
- Material **FloorMaterial**
- Material **CeilingMaterial**
- Material **TableMaterial**
- Material **UnknownMaterial**
- PlaneTypes **PlaneType** = PlaneTypes.Unknown

Propiedades

- BoundedPlane Plane `[get, set]`

  Gets or Sets the BoundedPlane, which determines the orientation/size/position of the gameObject.
- Vector3 SurfaceNormal `[get]`

  Gets the normal of the plane that was determined by the BoundedPlane object.
- bool IsVisible `[get, set]`

  Gets or sets the visibility of the current gameObject.

5.121.1. Descripción detallada

The SurfacePlane class is used by SurfaceMeshesToPlanes to create different types of planes (walls, floors, tables, etc.) based on the Spatial Mapping data returned by the SpatialMappingManager's source. This script should be a component on the SufacePlane prefab, which is used by SurfaceMeshesToPlanes.

5.121.2. Documentación de propiedades

5.121.2.1. IsVisible `bool Academy.HoloToolkit.Unity.SurfacePlane.IsVisible [get], [set]`

Gets or sets the visibility of the current gameObject.

5.121.2.2. Plane `BoundedPlane Academy.HoloToolkit.Unity.SurfacePlane.Plane [get], [set]`

Gets or Sets the BoundedPlane, which determines the orientation/size/position of the gameObject.

5.121.2.3. SurfaceNormal `Vector3 Academy.HoloToolkit.Unity.SurfacePlane.SurfaceNormal [get]`

Gets the normal of the plane that was determined by the BoundedPlane object.

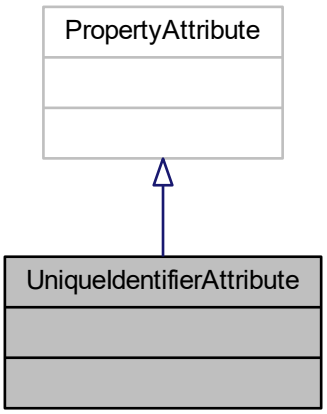
La documentación para esta clase fue generada a partir del siguiente fichero:

- Assets/HoloToolkit-SpatialMapping-230/SpatialMapping/Scripts/SurfacePlane.cs

5.122. Referencia de la Clase UniqueIdentifierAttribute

Clase para el editor que genera automáticamente un identificador para un objeto. Extraído de https↩
://answers.unity.com/questions/487121/automatically-assigning-gameobjects-a-unique-and-c
html

Diagrama de herencias de UniqueIdentifierAttribute



5.122.1. Descripción detallada

Clase para el editor que genera automáticamente un identificador para un objeto. Extraído de https↩
://answers.unity.com/questions/487121/automatically-assigning-gameobjects-a-unique-and-c
html

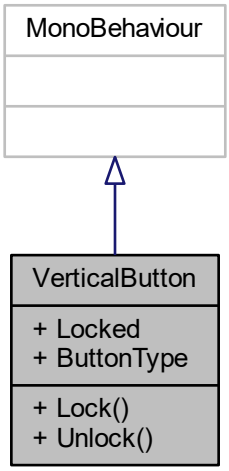
La documentación para esta clase fue generada a partir del siguiente fichero:

- Assets/Scripts/UniqueIdentifier/UniqueIdentifierAttribute.cs

5.123. Referencia de la Clase VerticalButton

Clase de los botones que activa el robot al moverse por las carreteras.

Diagrama de herencias de VerticalButton



Métodos públicos

- void Lock ()

  Bloquea el botón.
- void Unlock ()

  Desbloquea el botón.

Propiedades

- bool Locked `[get]`

  False si no está bloqueado, true si lo está.
- string ButtonType `[get]`

  Retorna el tipo del botón.

5.123.1. Descripción detallada

Clase de los botones que activa el robot al moverse por las carreteras.

5.123.2. Documentación de las funciones miembro

5.123.2.1. Lock() `void VerticalButton.Lock ( )`

Bloquea el botón.

5.123.2.2. Unlock() `void VerticalButton.Unlock ( )`

Desbloquea el botón.

5.123.3. Documentación de propiedades

5.123.3.1. ButtonType `string VerticalButton.ButtonType [get]`

Retorna el tipo del botón.

5.123.3.2. Locked `bool VerticalButton.Locked [get]`

False si no está bloqueado, true si lo está.

La documentación para esta clase fue generada a partir del siguiente fichero:

- Assets/Scripts/Roads/VerticalButton.cs

Índice alfabético

BIBLIOGRAFÍA

[1] *About Snap!* URL: https://snap.berkeley.edu/about (visitado 09-06-2020).

[2] M. A. S. Araújo. «Ensino de programação para crianças: uma análise da situação atual. Monografia (Licenciatura em computação)». En: *Instituto Federal de Educação, Ciência e Tecnologia do Sertão Pernambucano, Campus Petrolina, Petrolina - PE* (2014).

[3] *Blockly, A JavaScript library for building visual programming editors.* URL: https://developers.google.com/blockly (visitado 11-06-2020).

[4] Y. Chou. *Octalysis – the complete Gamification framework.* URL: https://yukaichou.com/gamification-examples/octalysis-complete-gamification-framework/ (visitado 21-03-2020).

[5] S. Cooper, W. Dann y R. Pausch. «Alice: A 3-D Tool for Introductory Programming Concepts». En: *J. Comput. Sci. Coll.* 15.5 (abr. de 2000), págs. 107-116. ISSN: 1937-4771.

[6] C. Duncan y T. Bell. «A Pilot Computer Science and Programming Course for Primary School Students». En: *Proceedings of the Workshop in Primary and Secondary Computing Education.* WiPSCE '15. London, United Kingdom: Association for Computing Machinery, 2015, págs. 39-48. ISBN: 9781450337533. DOI: 10.1145/2818314.2818328. URL: https://doi.org/10.1145/2818314.2818328.

[7] Logo Foundation. *Logo and Learning.* 2015. URL: https://el.media.mit.edu/logo-foundation/what_is_logo/logo_and_learning.html (visitado 10-06-2020).

[8] A Fowler, T. Fristce y M. MacLauren. «Kodu Game Lab: a programming environment». En: *The Computer Games Journal* 1.1 (mayo de 2012), págs. 17-28. DOI: 10.1007/bf03392325. URL: https://doi.org/10.1007/bf03392325.

[9] T. Karsenti. *12 reasons to learn coding at school.* 2019. URL: https://www.edcan.ca/articles/coding-in-schools/ (visitado 21-03-2020).

[10] A. Kay. «Squeak etoys, children & learning». En: *online article* 2006 (2005).

[11] D. Malan y H. Leitner. «Scratch for Budding Computer Scientists». En: *SIGCSE 2007: 38th SIGCSE Technical Symposium on Computer Science Education* 39 (mar. de 2007). DOI: 10.1145/1227310.1227388.

[12] A. Repenning y A. Ioannidou. «AgentCubes: Raising the Ceiling of End-User Development in Education through Incremental 3D». En: *Visual Languages and Human-Centric Computing (VL/HCC'06).* 2006, págs. 27-34.

[13] H. U. Roland. «Teaching Programming Using the Karel the Robot Paradigm Realized with a Conventional Language». En: 1990.

[14] S. Sánchez-Sobrino y col. *RoboTIC: a serious game based on augmented reality for learning programming.* Multimedia Tools and Applications. 2019.

[15] K. Schwab. *The Fourth Industrial Revolution*. 2010. URL: https://www.britannica.com/topic/The-Fourth-Industrial-Revolution-2119734 (visitado 20-03-2020).