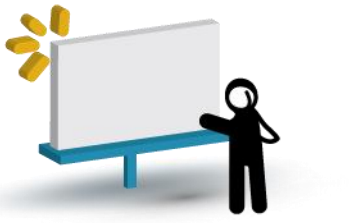
UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES
Centro de e-Learning

Programación de Video Juegos con Unity.Nivel inicial.

Centro de e-Learning SCEU UTN - BA.
Medrano 951 2do piso (1179) // Tel. +54 11 4867 7589 / Fax +54 11 4032 0148
www.sceu.frba.utn.edu.ar/e-learning

Módulo 1: Prototipado

Unidad 2: Creación y Destrucción de Objetos



Presentación:

En esta unidad aprenderemos a crear y destruir objetos, con la diferencia de que esta vez no lo haremos desde el editor, o sea, antes de ejecutar el juego, sino durante la ejecución del juego, cuando el juego está funcionando. Esto lo lograremos usando nuevas instrucciones y técnicas que iremos aprendiendo a lo largo de la unidad.

Finalmente, veremos cómo hacer para que Unity detecte el teclado y cuando el jugador presiona una tecla pueda lograr que cualquier instrucción que desee se ejecute, como moverse hacia adelante cuando presiona la tecla W, o disparar cuando presiona la tecla Espacio.

Objetivos:

Que los participantes aprendan:

- A preparar objetos para ser utilizados varias veces.
- El ciclo de vida de un objeto, o sea, creación, vida y destrucción.
- A detectar cuándo se presiona una tecla y responder ante eso.

Bloques temáticos:

- Prefabs.
- Input.
- Creación y Destrucción de Objetos

Consignas para el aprendizaje colaborativo

En esta Unidad los participantes se encontrarán con diferentes tipos de actividades que, en el marco de los fundamentos del MEC*, los referenciarán a tres comunidades de aprendizaje, que pondremos en funcionamiento en esta instancia de formación, a los efectos de aprovecharlas pedagógicamente:

Los foros proactivos asociados a cada una de las unidades.

La Web 2.0.

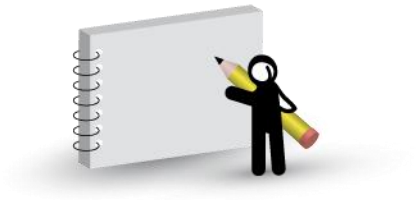
Los contextos de desempeño de los participantes.

Es importante que todos los participantes realicen algunas de las actividades sugeridas y compartan en los foros los resultados obtenidos.

Además, también se propondrán reflexiones, notas especiales y vinculaciones a bibliografía y sitios web.

El carácter constructivista y colaborativo del MEC nos exige que todas las actividades realizadas por los participantes sean compartidas en los foros.

**** El MEC es el modelo de E-learning colaborativo de nuestro Centro.***

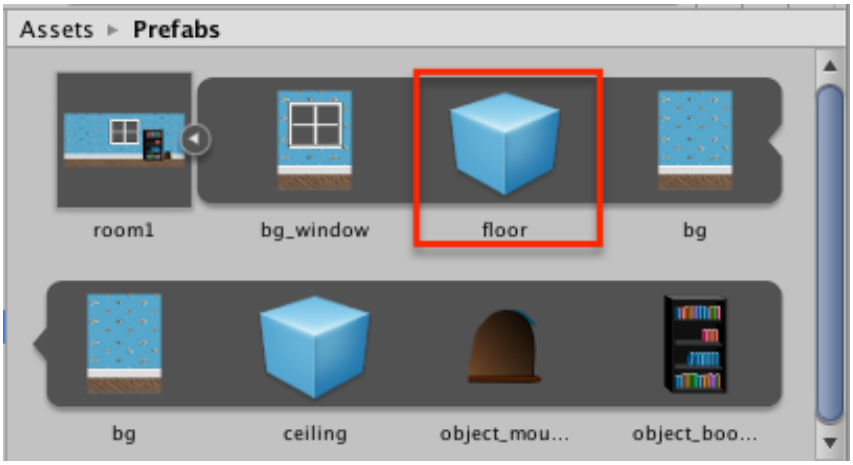


Tomen nota:

Las actividades son opcionales y pueden realizarse en forma individual, pero siempre es deseable que se las realice en equipo, con la finalidad de estimular y favorecer el trabajo colaborativo y el aprendizaje entre pares. Tenga en cuenta que, si bien las actividades son opcionales, su realización es de vital importancia para el logro de los objetivos de aprendizaje de esta instancia de formación. Si su tiempo no le permite realizar todas las actividades, por lo menos realice alguna, es fundamental que lo haga. Si cada uno de los participantes realiza alguna, el foro, que es una instancia clave en este tipo de cursos, tendrá una actividad muy enriquecedora.

Asimismo, también tengan en cuenta cuando trabajen en la Web, que en ella hay de todo, cosas excelentes, muy buenas, buenas, regulares, malas y muy malas. Por eso, es necesario aplicar filtros críticos para que las investigaciones y búsquedas se encaminen a la excelencia. Si tienen dudas con alguno de los datos recolectados, no dejen de consultar al profesor-tutor. También aprovechen en el foro proactivo las opiniones de sus compañeros de curso y colegas.

Prefabs


Assets ▸ Prefabs
room1
bg_window
floor
bg
bg
ceiling
object_mou...
object_boo...

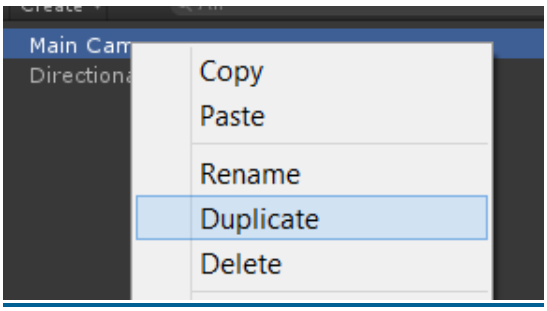
Reutilización de Objetos

Hasta ahora hemos creado diferentes objetos en Unity, cada uno independiente del otro. Recordemos que para ello creábamos un GameObject, le agregábamos o quitábamos componentes, y configurábamos los componentes que quedaban modificando sus propiedades para así poder afectar el comportamiento del componente. También podemos crear componentes propios y agregarle propiedades para así hacer que nuestro componente sirva en diferentes situaciones, como por ejemplo, configurar la velocidad para hacer dos objetos con movimiento similar pero con diferentes velocidades, o un objeto que vaya hacia atrás, configurando una velocidad negativa, utilizando un solo componente.

Luego de haber configurado el objeto para que quede como queremos tenemos el problema de que algunas veces vamos a necesitar tener un objeto exactamente igual pero con ciertos detalles diferentes, como una posición diferente, un color diferente, una velocidad diferente, etc. Nada nos impide crearlo de vuelta, pero si el objeto consta de muchos componentes con muchas configuraciones haría el trabajo muy tedioso, por lo que Unity cuenta con diferentes herramientas para ayudarnos en esta tarea.

Duplicar Objetos

La primera forma y la mas simple constaría de usar la opción de duplicación que nos permite crear una copia exacta del objeto independiente del original, por lo que una vez duplicado podemos volver a modificar las propiedades de cada uno sin afectar las del otro, para lograr esto podremos hacer Clic Derecho sobre el objeto y seleccionar la opción "Duplicate" (duplicar en ingles) o usando el atajo CTRL + D.

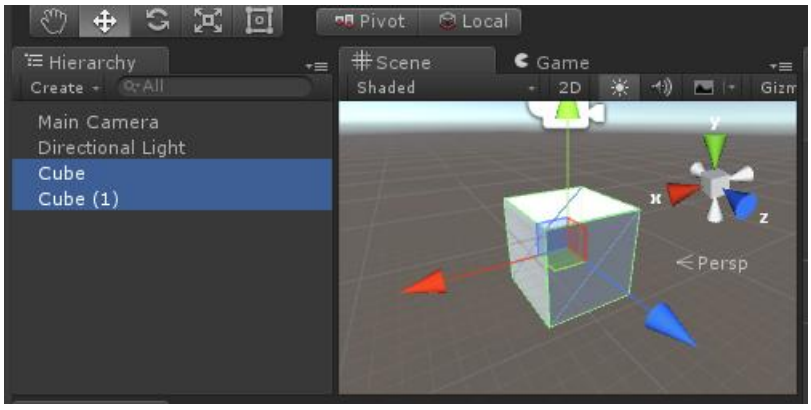


Ejercicio N°1

- *Cree un cubo.*
- *Selecciónelo haciendo clic sobre el en el panel de Scene o sobre su nombre en el panel de Hierarchy.*
- *Haga clic derecho en el y seleccione la opción "Duplicate", repita este proceso 3 veces y vea como cada vez se va creando un cubo nuevo.*
- *Seleccione alguno de los cubos y muévalos, viendo así cómo al mover se puede apreciar que todas las copias están en el mismo lugar.*
- *Repita el proceso de duplicación sobre cualquiera de las copias pero esta vez utilizando el atajo CTRL+D mientras tiene seleccionado el objeto a copiar.*

Una vez hecho esto podremos ver en la ventana de hierarchy cómo se agregó un objeto más a la lista, llamado igual al objeto duplicado pero agregándole al final del nombre entre paréntesis el número de copia, por ejemplo, si duplicásemos un objeto llamado "Cube", la copia seria "Cube (1)", si lo volviésemos a duplicar seria "Cube (2)".

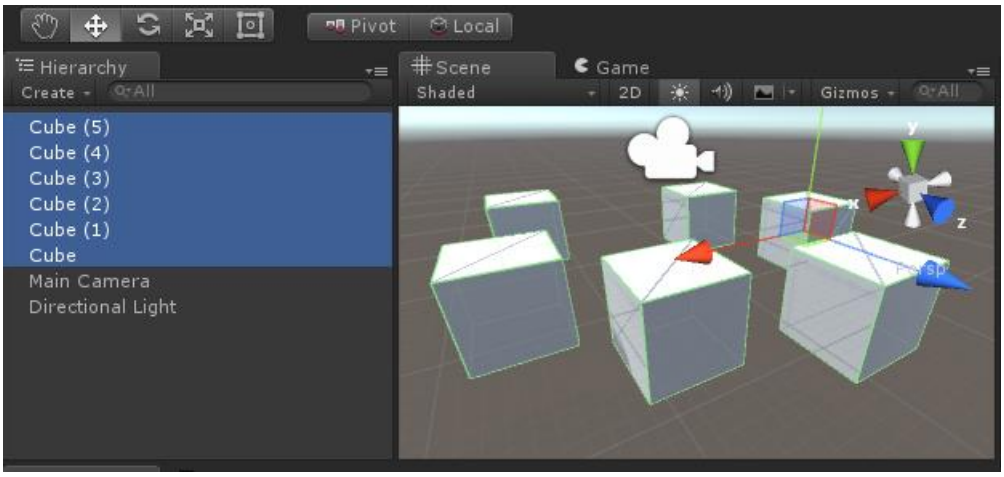
Hay que tener en cuenta que la copia se hace tal cual el objeto original, por lo que el objeto estaría ubicado en la misma posición del original, habría que moverlo para diferenciarlo del otro, sino estarían los dos en el mismo lugar y lucirían como uno (como se puede apreciar en la imagen, donde en el panel Scene se ve un solo objeto cuando en el panel de Hierarchy hay 2).



Edición de Múltiples Objetos

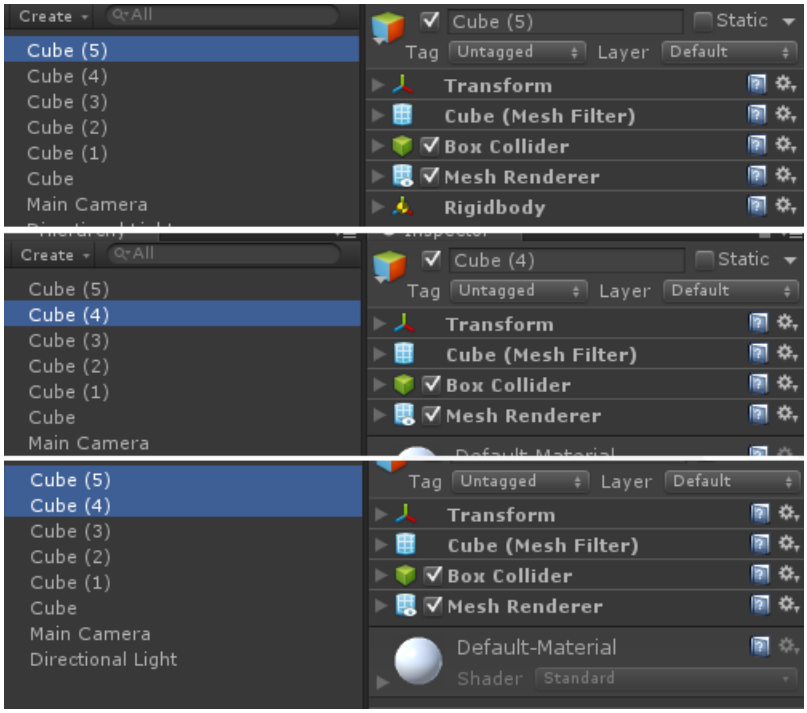
La herramienta de duplicación es muy útil pero tiene un problema. Al duplicar, ambos objetos quedan desconectados, o sea, que si cambio uno de ellos, el otro no, lo que haría que si empezamos a duplicar muchas veces el mismo objeto y luego queremos cambiar un aspecto de todas las copias y el original (que sería una copia más), tendríamos que ir uno por uno cambiando los valores. Si tenemos 100 copias y queremos cambiar 3 propiedades en cada copia, tendríamos que modificar 300 valores, un trabajo titánico considerando que probablemente cambiemos cosas frecuentemente.

Una forma que tiene Unity para resolver este problema es a través de la edición múltiple de objetos, esto implica poder cambiar una propiedad que tengan en común el conjunto de objetos seleccionados, y que el cambio aplique a todos. Para ello, primero necesitamos saber cómo seleccionar varios objetos, lo que se puede lograr manteniendo la tecla CTRL presionada y haciendo clic sobre el objeto tanto en la ventana de Hierarchy como en la de Scene, pudiéndose apreciar mientras hacemos esto que en la ventana de Hierarchy se van señalando con un fondo azul los objetos que están siendo actualmente seleccionados y en la ventana Scene se ven los gizmos (ayudas visuales) de selección sobre los objetos.

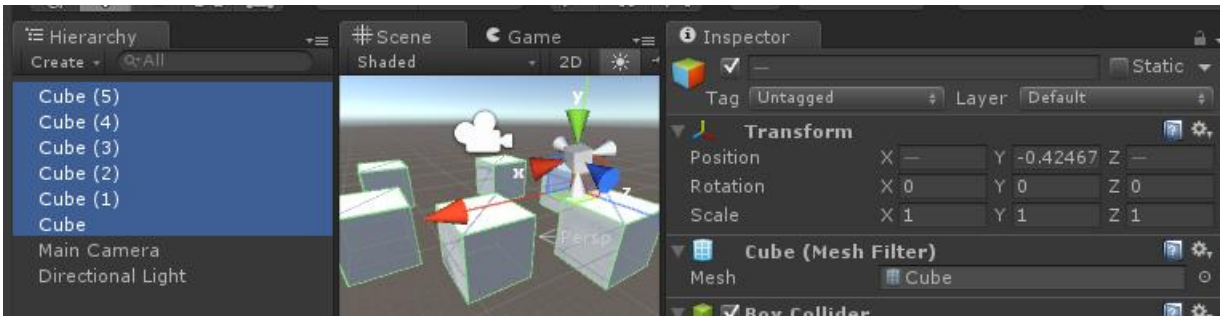


Una vez hecho esto se podrá apreciar cómo el panel de Inspector, el panel que nos muestra las propiedades del objeto seleccionado, sigue mostrando propiedades, esta vez mostrando solamente las que tienen en común los objetos, por lo que, si seleccionamos varios objetos pero con diferentes componentes, vamos a ver solamente los componentes que tienen en

común, por ejemplo, si tengo un objeto con el componente transform, rigidbody y boxcollider, y otro con transform y boxcollider solamente, seleccionando cada uno por separado el inspector nos mostrara los componentes del objeto seleccionado, pero si los seleccionamos a ambos solamente veremos los que comparten, en este caso transform y boxcollider, ya que el segundo no tiene rigidbody.



También veremos otro detalle, ciertas propiedades en vez de tener un valor, aparece en su lugar un guion ("-"), esto indica que ambos objetos tienen el mismo componente, pero que esa propiedad que tiene el guion no coincide en todos los objetos seleccionados, por ejemplo, si acabamos de duplicar un objeto y sin modificar ninguno de ellos los seleccionamos, podremos apreciar cómo ninguna propiedad tiene guiones, pero si ahora seleccionamos uno de ellos, lo movemos de lugar y volvemos a seleccionar ambos, en la propiedad posición, las coordenadas X Y Z van a tener un guion en las que son diferentes, indicando que los objetos están en posiciones diferentes.



De todas formas, con guion o sin guion podremos cambiar esa propiedad poniéndole un nuevo valor, y ese nuevo valor se aplicara a todos los objetos seleccionados, así ahorrándonos el trabajo de tener que ir uno por uno.

Ejercicio N°2

- *Cree un cubo, agréguele el componente rigidbody y dupliquelo 3 veces poniéndolo en lugares diferentes.*
- *Tome uno de ellos y destilde la opción "Use Gravity"*
- *Haga clic en el botón play viendo así como todos caen menos uno. Asegúrese de que la cámara este mirando los objetos, sino cuando haga clic en Play no se verán.*
- *Luego seleccione todos y tilde la opción "Use Gravity", vea como después de darle Play caen todos.*
- *Seleccione todos nuevamente y destilde la opción "Use Gravity", vea como después de darle Play no cae ninguno.*

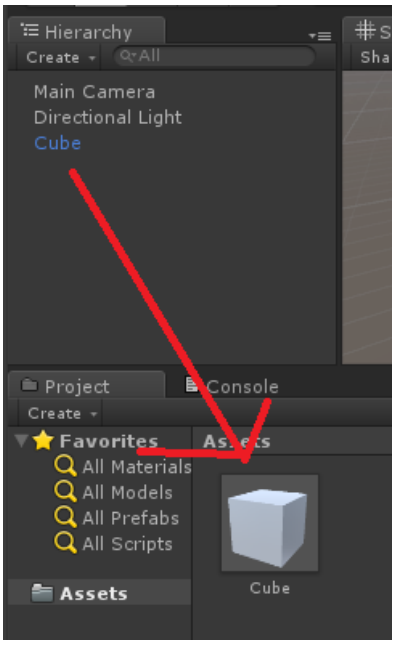
Creando Prefabs

Ahora, como vimos, cada opción sirve para diferentes situaciones, todavía nos queda contemplar una última. Supongamos que tenemos una columna, dicha columna se usa varias veces en nuestro juego como elemento estético para nuestros interiores, y claramente va a haber mas de una columna. Dicha columna, como todo objeto en nuestro juego, va a tender a cambiarse repetidas veces durante el desarrollo, en este caso por cuestiones estéticas, pero esta vez esta columna no sólo va a estar en una escena, sino que en varias, o sea, la misma columna en diferentes edificios de diferentes escenas. Hasta ahora vimos que podemos seleccionar todas las columnas y modificarlas, pero solo las de la escena que tenemos abierta, esto implica que si tenemos que modificar la columna, y que los cambios se reflejen en todas las escenas tendremos que ir escena por escena modificando las columnas, cosa que vuelve a ser un trabajo largo y tedioso, acá es donde entran los prefabs.

Los prefabs (o prefabricados en castellano), como su nombre lo indica, sirven como moldes de objetos desde el cual se pueden empezar a crear copias de ese molde, normalmente llamadas "Instancias". Uno puede crear un prefab a partir de un objeto existente y de esa manera guardar el objeto y sus configuraciones en este molde. De esta forma haríamos la columna una sola vez, crearíamos un prefab, y empezaríamos a utilizarlo en nuestras escenas así no tenemos que crearlo a mano una y otra vez.

Ahora, la principal ventaja de los prefabs es que las instancias (las copias del prefab) están "vinculadas" al prefab que los origino, eso implica, que si cambiamos el prefab, se cambian todas las instancias en todas las escenas en todo el proyecto.

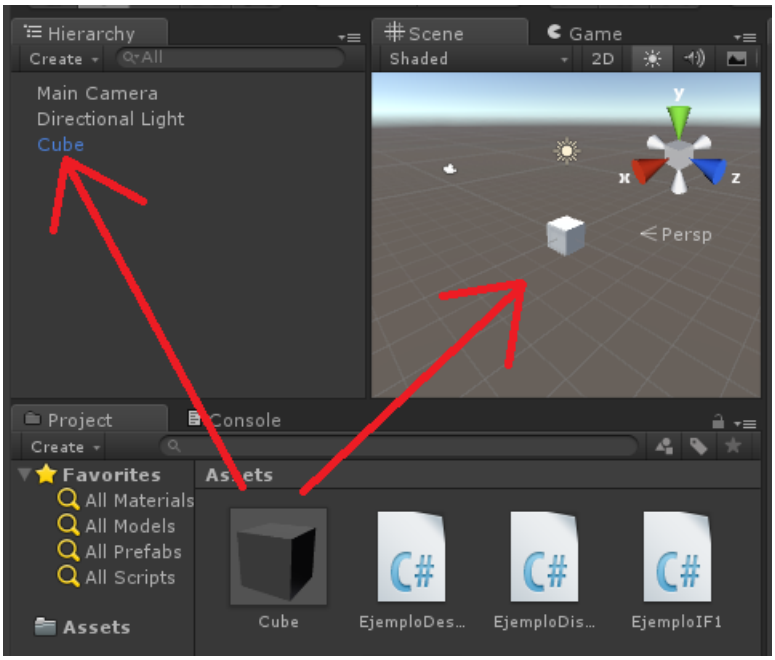
Para crear un prefab simplemente seleccionamos el objeto que deseamos convertir en prefab y lo arrastramos del panel de Hierarchy al panel Project, veremos que cuando hacemos esto suceden dos cosas. Una es que al crear el prefab se creo un asset en el panel de project (recordemos que un asset es un archivo del proyecto, algo que esta fuera de la escena, algo que compone al proyecto), y la segunda es que el nombre del objeto original se ve celeste en el panel de Hierarchy, cuando un objeto en dicho panel se ve celeste, esto indica que ese objeto es una instancia de un prefab, así identificando que si el prefab cambia, este objeto va a cambiar.



Hay que tener en cuenta que vamos a tener muchos prefabs en todo el proyecto, y que todos los objetos de una escena que estén relacionados a un prefab se van a ver de celeste, pero esto no indica que están todos relacionados al mismo prefab, se usa el color celeste solamente para indicar que esta relacionado a un prefab, no a cual.

Al usar prefabs de esta forma, garantizamos que un objeto que tiene que ser idéntico en múltiples lugares tenga un molde al cual si lo modificamos, automáticamente los cambios se vean reflejados en todos los lugares donde se use, pero este no es el único uso que le vamos a dar a los prefabs a lo largo del curso.

Ahora que creamos un prefab a partir de un objeto, podemos crear nuevas copias del mismo de dos formas. Una es simplemente duplicando un objeto que sea una instancia del prefab (que el nombre este en celeste), al hacer esto el objeto que se duplico también se va a ver de color celeste, indicando que también esta relacionado a un prefab. La segunda opción es arrastrar el prefab del panel de Project al panel de Hierarchy o al panel de Scene, de una forma u otra cuando soltemos veremos que en el panel de Hierarchy aparece un nuevo objeto, nuevamente con el nombre en color celeste, que es idéntico al prefab, este objeto también esta conectado al prefab.



Ejercicio N°3

- *Cree un cubo y agréguele el componente rigidbody.*

- *Arrastre el cubo desde el panel de Hierarchy al panel de Project, así creando un prefab.*

- *Haga dos instancias del cubo seleccionando la instancia en el panel de Hierarchy (el que esta en celeste, o sea, el que acabamos de convertir en instancia de un prefab) y haciendo clic derecho → Duplicate dos veces.*

- *Haga dos nuevas instancias del prefab pero esta vez arrastrando el prefab desde el panel de Project al panel de Hierarchy.*

Modificando Prefabs

Es importante recalcar la diferencia entre el prefab y las instancias. Recordemos que el prefab es el asset (archivo) que esta en el panel de Project, y las instancias son las copias que se ven en el panel de Hierarchy, los cuales residen en una escena, no en el proyecto en si como los prefabs, y que estos son los que forman parte del juego. Un prefab si no tiene instancias suyas en alguna escena, no se ve en ningún nivel del juego, no se esta usando, aunque de todas formas mas adelante veremos como crear copias pero cuando el juego esta funcionando por código.

Ahora, esta diferenciación también nos ayuda a entender las modificaciones sobre el prefab. Cuando seleccionamos un prefab o una instancia veremos que en el panel de Inspector (recordemos que este panel nos permite modificar los componentes del objeto seleccionado, así modificando su comportamiento y apariencia en el juego) veremos las mismas propiedades y componentes, esto demostrando que podamos modificar tanto el prefab como las instancias, pero dando resultados diferentes en cada caso.

Si nosotros seleccionamos un prefab del panel de Project y modificamos alguna de sus propiedades en el panel de Inspector veremos como todas las instancias del prefab cambian automáticamente.

Ejercicio N°4

- *Seleccione el prefab creado en el ejercicio 3 y modifique su tamaño modificando, por ejemplo, la propiedad x de la propiedad scale del componente transform a mano desde el inspector, por ejemplo poniéndole el valor 3, y vea como cambia el tamaño de todas las instancias.*

- *Desactive el componente MeshRenderer y vea como los cubos dejan de verse, reactivelo.*

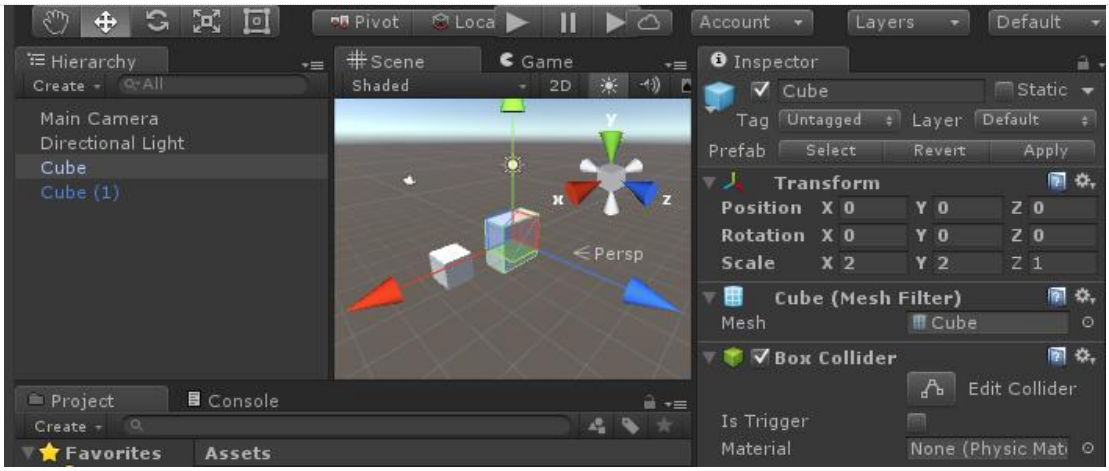
- *Por último destilde la opción Use Gravity y vea como cuando le da Play al proyecto este vez no caen, pare el proyecto, vuelva a tildarlo y pruebe nuevamente.*

Un detalle a recordar a la hora de modificar los objetos es que todo cambio que realicemos mientras el juego esta en modo Play, se va a revertir cuando lo paremos.

Volviendo a los prefabs, esta es una forma conveniente de modificar un objeto sin tener que abrir una escena que posea una instancia, simplemente buscamos el objeto a modificar en el panel de Project y lo cambiamos, pero en ciertas situaciones no alcanza con ver que cambia la propiedad en el panel del Inspector, sino que queremos ver como cambia en el panel de Scene para ver si el objeto quedo como queremos, por ejemplo, si cambiamos su tamaño o su color. En esa situación tenemos dos opciones, una es abrir una escena que posea una instancia de ese prefab y modificarla, y la otra opción es en cualquier escena crear una instancia nueva de ese prefab y modificarla, de una forma u otra podremos ver si los cambios quedaron como queremos, hasta de hecho podríamos darle Play al proyecto y ver si el objeto se comporta como lo esperado.

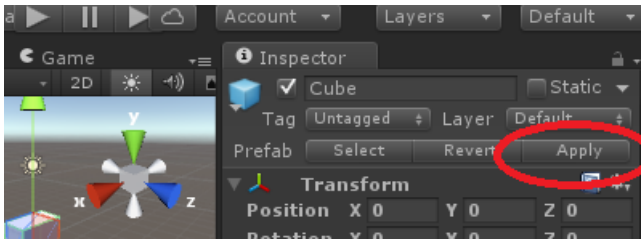
Pero hacer esto tiene un detalle. Si nos damos cuenta, al cambiar una instancia de un prefab y no el prefab en sí, veremos que las otras instancias no cambian, esto está hecho a propósito y sirve para dos cosas. Una es la capacidad de probar los cambios sobre una instancia y ver si ese cambio es el que deseamos, la otra es la capacidad de que una instancia sea parcialmente igual a su prefab, pero con ligeras diferencias, como por ejemplo, la velocidad de movimiento de esa instancia, o el color.

Otro detalle a tener en cuenta es que el nombre de las propiedades modificadas en una instancia se ven en negrita, o sea, que la letra se ve "más gruesa", esto indicando cuáles son las propiedades que difieren del prefab, cuáles modificamos. También hay que ver que la posición y la rotación de una instancia siempre va a estar en negrita, ya que no seria algo práctico compartir la misma posición y rotación de un objeto.



(en esta imagen vemos como la propiedad Z de la escala no esta en negrita, indicando que sigue estando conectada al prefab a diferencia de X e Y)

Cuando ya vimos que los cambios son los deseados, para confirmar dichos cambios podemos hacer clic en el botón Apply que figura en la parte superior del panel Inspector, manteniendo seleccionada la instancia modificada, esto hará que los cambios se apliquen al prefab y veremos cómo las propiedades de la instancia modificada dejan de estar en negrita, y como las otras instancias cambiaron.



Ejercicio N°5

- *Seleccione una de las instancias del prefab "Cube" creadas en los ejercicios anteriores y modifique su tamaño, esta vez modificándola desde el panel de Scene usando los gizmos de transformación (los 5 botones que están arriba a la izquierda en el editor), vea como se modifica esa instancia únicamente.*

- *Haga clic en el botón Apply y vea como cambian todas las instancia.*

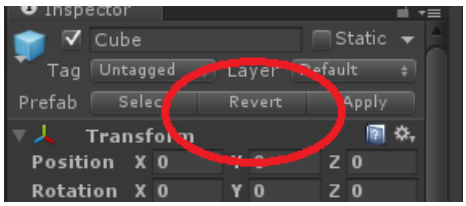
- *Guarde esta escena, cree una nueva (File → New Scene) y cree una instancia del prefab en esa escena arrastrando el prefab al panel de Scene.*

- *Vuelva a modificar el tamaño y vuelva a aplicar los cambios.*

- *Regrese a la escena anterior (haciendo doble clic en el asset de la escena en el panel de Project) y vea como los cambios hechos en la escena que teníamos cargada anteriormente se aplicaron en esta escena.*

Revirtiendo, Seleccionando y Eliminando Prefabs

Algunas veces vamos a querer probar cambios sobre una instancia de un prefab, pero no nos gusta como queda y deseamos revertir los cambios para que vuelva a ser igual al prefab, para ello hacemos clic en la opción Revert en el panel de Inspector manteniendo seleccionado en el panel de Hierarchy el objeto a revertir. Esto revierte todos los cambios a sus valores originales, los que estaban en el prefab, y veremos cómo ahora no hay ninguna propiedad en negrita.



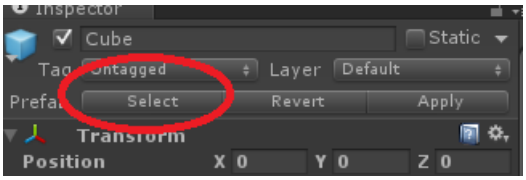
Otras veces necesitamos probar los cambios, pero solamente queremos que se apliquen algunos de esos cambios. Si bien no podemos aplicar los cambios hechos en una sola propiedad ya que el botón Apply aplica todas las modificaciones, lo que si podemos es revertir un cambio hecho sobre una instancia haciendo clic derecho en la propiedad modificada (alguna que este en negrita) y seleccionando la opción "Revert Value To Prefab".

Esto permite deshacer ciertos cambios que no deseamos y luego aplicar los cambios que si queremos al prefab. Esto lleva a un buena práctica a la hora de modificar un prefab, que es siempre crear una instancia limpia del prefab, modificarla y aplicar esos cambios en vez de seleccionar una instancia ya creada que quizás tenga alguna propiedad modificada y al no darnos cuenta que la tiene, al usar el botón Apply, se aplica esa propiedad a pesar de que no era lo que queríamos.

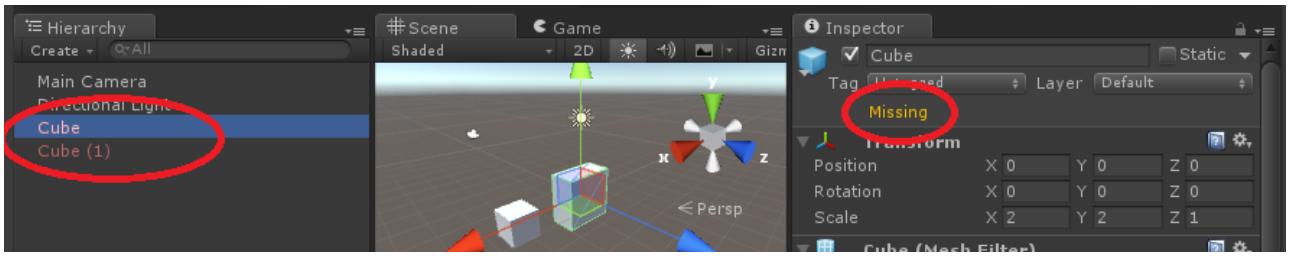
Ejercicio N°6

- *Cree una nueva instancia de un prefab, cosa de que esta instancia este limpia de cambios.*
- *Modifique su tamaño en todos los ejes.*
- *Haga clic derecho en el eje X de la escala y revierta los cambios seleccionando la opción "Revert Value to Prefab"*
- *Aplique los cambios haciendo clic en el botón Apply.*

Otra herramienta útil a la hora de manipular prefabs es saber a que prefab pertenece una instancia, simplemente seleccionando la instancia y haciendo clic en el botón Select del panel de Inspector, veremos que al hacer esto en el panel de Project se seleccionara el prefab correspondiente.



El último detalle es que al eliminar un prefab (clic derecho → delete sobre el asset), las instancias en las escenas continúan existiendo, pero se vera cómo en vez de aparecer en celeste, aparecen en rojo, indicando que el prefab al cual estaban conectadas no se puede encontrar, teniendo la desventaja de que una vez hecho esto no podremos reconectar esa instancia a un nuevo prefab. También se muestra en el panel donde estaban los botones Select, Revert y Apply una leyenda diciendo "Missing".



Ejercicio N°7

- *Elimine el prefab y vea cómo todas las instancias aparecen en rojo, esto indicando que hay que tener cuidado a la hora de eliminar prefabs.*

UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES
Centro de e-Learning

Input

Prepararse para detectar Input

Hasta ahora la mayoría de las acciones que modifican a nuestro juego las realizamos desde el editor (crear objetos, moverlos, modificar sus propiedades, etc). Como vimos en la unidad anterior, cuando editamos un nivel desde el editor estamos configurando el estado inicial del mismo (donde aparecen los objetos, donde aparece el jugador, etc), pero una vez que le damos Play al juego, los componentes entran en acción y empiezan a mover objetos, detectar colisiones y aplicar físicas (en el caso del rigidbody), entre otras cosas, así modificando el estado inicial de la escena y simulando las reglas de nuestro juego. Vimos también que podemos crear componentes propios para agregarle reglas y comportamientos nuevos a los objetos para que se ejecuten durante la reproducción del juego, como el componente que hicimos para mover al objeto constantemente, o sea, que en resumen, los componentes son el lugar donde vamos a empezar a crear la jugabilidad y reglas de nuestro juego (mecánicas de gameplay).

Como mencionamos anteriormente, la unidad pasada vimos una de las acciones más básicas, que es mover, pero hacer que el objeto se mueva automáticamente sin ningún control del usuario nos sirve solamente en casos puntuales como balas o enemigos, no para el personaje principal, el cual se tiene que mover a la orden del jugador. Dentro de la simulación que se ejecuta cuando damos play, tenemos que agregar nuevas reglas al juego que le permitan al jugador poder cambiar el desenlace del mismo, esto se hace a través de lo que se conoce como "Input"

Se le llama Input a todo aquello que le permite al jugador comunicarle sus deseos a la computadora, como lo hace un teclado, el mouse, un joystick, una pantalla táctil de un celular, un volante de un juego de carreras, una palanca de vuelo, hasta los mismos botones a los cuales les hacemos clic con el mouse en la pantalla, que a pesar de no ser físicos, se los sigue considerando input (del tipo conocido como GUI o Graphic User Interface, sobre eso hablaremos en otros capítulos). Como dato extra es conveniente recalcar que como existe el input (o entrada), también existe el output (salida), que como su nombre lo indica, al ser lo opuesto al input, le comunica cosas de la computadora al jugador, como lo hace un monitor, una impresora, etc.

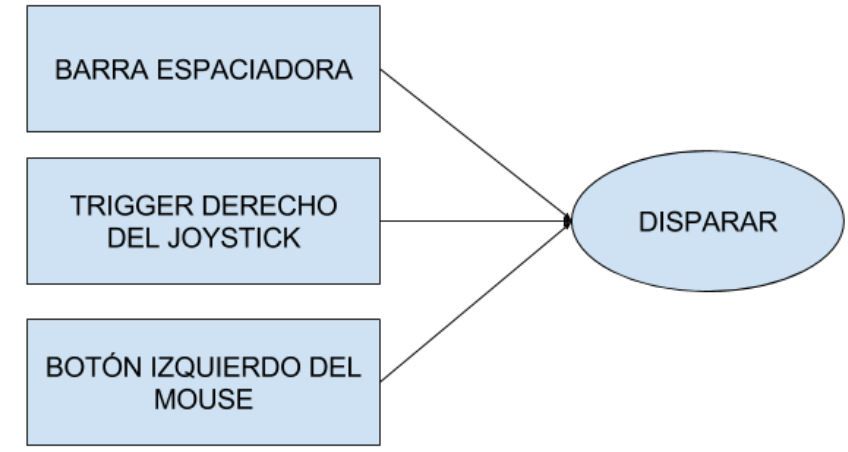
A continuación veremos cómo aprender a detectar el input que nos brinda el teclado.

Input Manager

Si bien hay varias formas de detectar input desde Unity (todo tiene varias formas de hacerse, con sus pros y contras), veamos como hacer uso de una herramienta proveída por Unity para poder tomar el input proveniente de muchas fuentes (teclado, mouse, etc…) y adaptarlo para que sea mas comprensible, a esta herramienta se la conoce como Input Manager (Administrador de Entrada).

Al tener diversas fuentes de input, es muy fácil que se empiece a complicar el código necesario para escuchar esa avalancha de información que viene de esos dispositivos, y por ello es conveniente organizarla de una forma. Por ejemplo, muchas veces queremos que el personaje dispare tanto si apretamos el clic Izquierdo, como si apretamos la tecla espacio, o también si apretamos el gatillo de un joystick, otras veces queremos que se mueva tanto con la palanca de un joystick tanto como con las teclas del teclado. Para organizar todo ello conviene dejar de pensar en botones y palancas, y empezar a pensar en acciones concretas, como disparar, saltar, pausar, etc, el sistema del Input Manager lidia con este problema realizando lo que se conoce como Mapeo de Input.

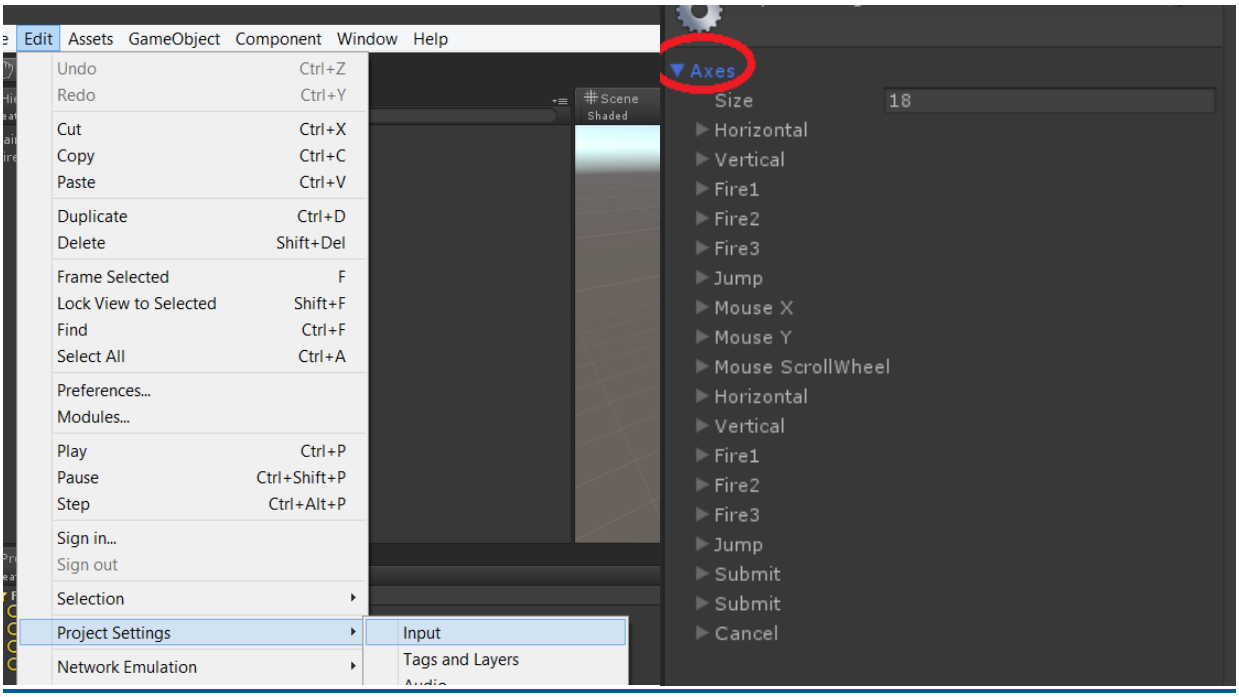
El Mapeo de Input consta de crear acciones en el Input Manager, y a esas acciones asociarle diferentes Inputs, como por ejemplo, crear la acción Disparar y asociarla al botón izquierdo del mouse, a la tecla espacio del teclado y al botón A de un joystick, esto haciendo que de ahora en mas, en vez de preguntarle a Unity si la barra espaciadora, o el botón del mouse, o el botón del joystick están presionados, preguntamos si se ejecuto la acción Disparar, olvidándonos de teclas, pensando en acciones, abstrayéndonos.



Entonces, de ahora en más, cuando pensemos en input, en vez de pensar en teclas, pensemos en acciones. Veamos cómo crear acciones en Unity.

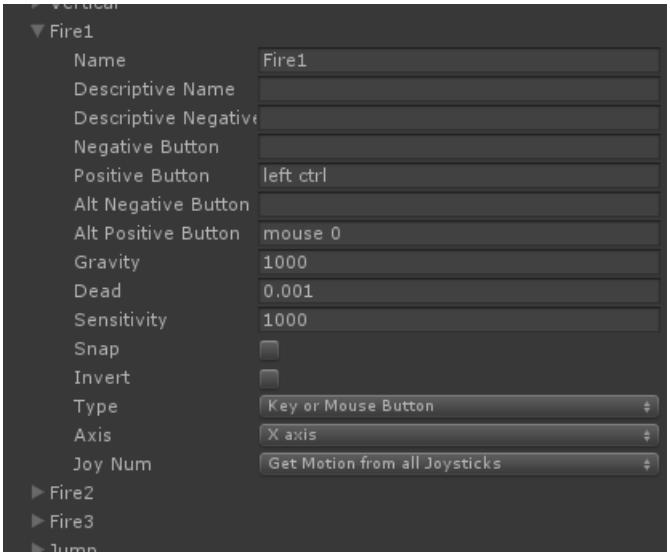
Creando y Modificando Botones

Primero debemos ir a Edit → Project Settings → Input, y una vez hecho esto veremos que en el panel de inspector aparecieron nuevas propiedades, lo que nos permite notar que el panel de inspector sirve para editar las propiedades de cualquier cosa seleccionada, más allá de los GameObjects, por lo que sirve en general, para editar propiedades de lo seleccionado. En principio veremos, en este caso, que en el inspector aparece una propiedad que dice "Axes" con una flecha a la izquierda y nada más, para ver los inputs configurados debemos hacer clic en esa flecha y se desplegaran todos los inputs que vienen configurados por defecto, en este caso, vienen 18 inputs configurados.



Ahora, si analizamos los nombres de la lista, algunos son bastante descriptivos, como Fire1 (Disparo 1), Jump (Saltar), Cancel (Cancelar, en el caso de menúes por ej), Submit (Confirmar, como cuando apretamos enter para enviar un mensaje de texto en un chat), etc., pero otros no tanto, como Horizontal, Vertical, Mouse X, ellos no suenan a acciones, suenan a ejes de coordenadas, eso se debe a que el input no sólo está compuesto de "Acciones", sino también de "Ejes", pero los veremos más adelante, por ahora concentrémonos en las Acciones. Veamos primero cómo está configurada una acción de las predefinidas, las que ya vienen en Unity.

Empecemos por Fire1, haciendo clic en la flecha que está a la izquierda de "Fire1", como lo hicimos con Axes. Allí veremos una serie de propiedades asociadas a la acción, veamos las más importantes para los botones.



- *Name: Aquí le damos nombre a la acción, hay que tener cuidado con modificar los nombres una vez que el proyecto está avanzado ya que ésto llevaría a que tengamos que modificar todos los lugares donde hemos usado esta acción.*

- *Positive Button: Aquí pondremos el nombre real de la tecla a la cual queremos que este botón responda, como "left ctrl" (la que viene por defecto, representando la tecla Control Izquierdo) o "mouse 0" (el botón izquierdo del mouse) o también podríamos poner "a" (la tecla A del teclado) como así también botones de joystick como "joystick button 0" (este botón puede ser diferente según joystick, aunque como regla general siempre se tiene en cuenta un joystick de Xbox 360). Más adelante encontraremos una lista completa con los nombres de todas las teclas tanto del teclado como de un joystick de Xbox 360.*

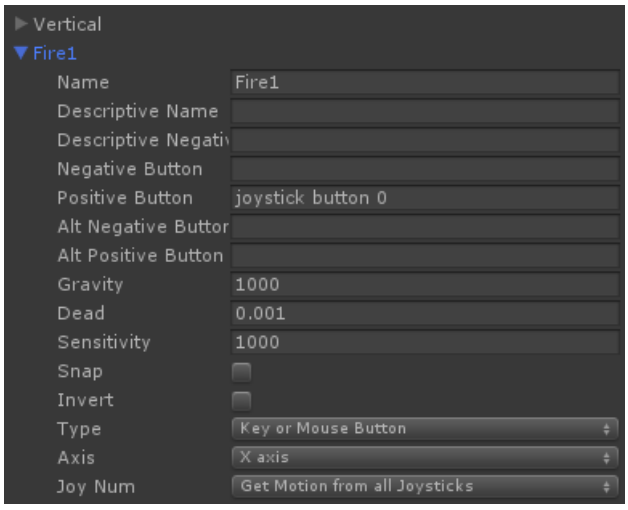
- *Alt Positive Button: Aquí podremos poner un botón alternativo, y es aquí donde empezaremos a hacer que nuestra acción responda a múltiples inputs, en este caso, vemos que el botón alternativo es "mouse 0", por lo que esta acción responde a esos dos botones.*

- *Type: En esta propiedad, mientras que sólo usemos Botones y no Ejes, hay que asegurarse de que esté seleccionada la opción "Key or Mouse Button" (haciendo clic sobre la propiedad aparecerán las opciones).*

- *Joy Num: Aquí da la opción de que en caso de querer recibir input un joystick en especifico, de cual número de joystick vamos a tomar, fíjense que viene configurado con la opción "Get Motion from all Joysticks". Si nosotros quisiéramos hacer un juego multiplayer local (pantalla dividida) podríamos tener un botón Fire para el joystick 1, y otro para el joystick 2, para así poder diferenciar quién de los dos jugadores presionó la tecla, porque si lo dejamos como viene configurado, va a tomar el botón de cualquier joystick y ambos dispararían.*

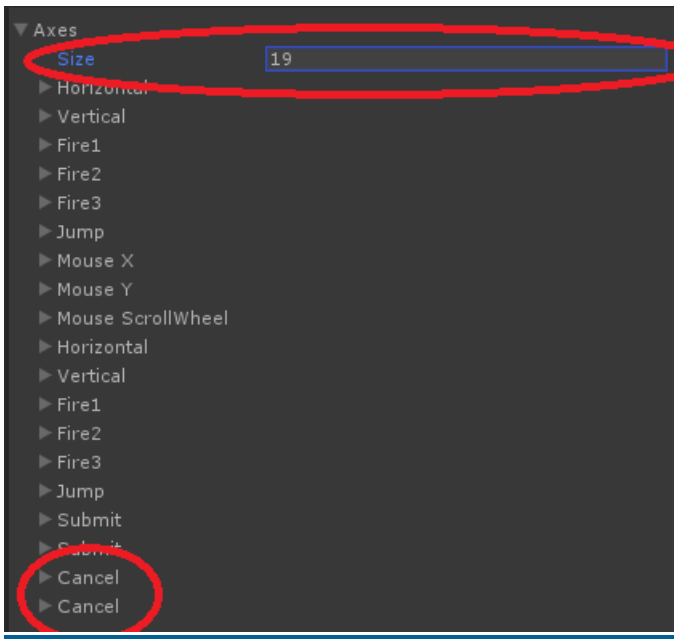
El resto de las opciones son para configurar Ejes, por lo que, por el momento, no nos vamos a preocupar.

Ahora anteriormente hablamos de que una acción puede recibir input de muchas fuentes, cuando hasta ahora solamente recibe de un máximo de 2 fuentes (Positive Button y Alt Positive Button), eso simplemente se soluciona creando otra acción que se llame igual, por ejemplo, siguiendo el caso de Fire1, veremos que hay dos Fire1 en la lista, veamos el segundo. En este caso todas las opciones menos Positive Button están configuradas igual, estando esta ultima configurada con "joystick button 0" (en el joystick de xbox 360 esta tecla seria la "A") y Alt Positive Button no tiene nada, ya que no hace falta otro botón.



Entonces, resumiendo, la acción "Fire1" responde a los inputs "left ctrl", "mouse 0" y "joystick button 0" (o sea, al control izquierdo, el clic izquierdo del mouse y al botón a de un joystick de xbox 360), por lo que la presión de cualquiera de esas teclas se traduciría a ejecutar la acción "Fire1", haciendo que se ejecuten las acciones correspondientes en todos los lugares donde nosotros programemos la detección de esa tecla.

Por último, en el caso de querer crear un nuevo botón, tenemos que cambiar la propiedad Size que vemos arriba de todo, por defecto viene en 18 (18 acciones o ejes), para agregar una nueva tendremos que cambiar esta propiedad a 19 (y luego de ello, para agregar otra más, le pondríamos 20, y así sucesivamente), veremos que se duplica la ultima tecla, por lo que tenemos que empezar a crear la nueva a partir de esta ultima.



Ejercicio N°8

- *Vaya al Input Manager (Edit → Project Settings → Input).*
- *Cree un nuevo input (cambiando el valor de size sumándole uno, o sea, de 18 a 19).*
- *Busque el último input y ábralo (usando la flecha). En caso de ser el primero que cree, esta será la copia de "Cancel", ya que "Cancel" es el último de los valores que vienen por defecto).*
- *Modifique el nombre para que se llame "Reload" (Recargar, se podría usar para recargar el arma), el Positive Button como "R" y el Alt Positive Button como "joystick button 1" (la tecla B de xbox360).*
- *Agregue una nueva acción, verá cómo se duplica la que acabamos de hacer.*
- *Modifique el Positive Button a "mouse 1" (clic derecho del mouse) y deje vació el Alt Positive Button, pero deje el nombre igual, así extendiendo la acción previamente creada.*

Nombre de los inputs

Ya vimos cómo configurar los botones y vimos los nombres de las teclas más comunes, como Control Izquierdo, Clic Izquierdo del Mouse y el botón A del Joystick. Veremos a continuación las pautas a tener en cuenta a la hora de ingresar los nombres de las teclas del teclado.

- *Teclas Alfanuméricas: Cualquier tecla que represente un carácter (a, b, c, etc.) o un numero va a tener el nombre idéntico al símbolo que lo representa, en caso de la a escribiríamos "a", en caso de la b escribiríamos "b" (y no "be"), o en el caso del uno escribiríamos "1" o en el caso de nueve escribiríamos "9". Tener en cuenta que las letras se tienen que escribir en minúscula ("a" en vez de "A").*

- *Flechas: Las flechas del teclado arriba, abajo, izquierda y derecha serian "up", "down", "left" y "right" respectivamente.*

- *Teclas de modificación: Con esto nos referimos a los shift, los control, y la tecla windows (la que tiene el logo de windows, o command en mac) y tenemos las siguientes opciones: "right shift" (shift derecho), "left shift" (shift izquierdo), "right ctrl" (control derecho), "left ctrl" (control izquierdo), "right alt" (alt derecho), "left alt" (alt izquierdo), "right cmd" (tecla windows o command derecha) y "left cmd" (tecla windows o command izquierda).*

- *Botones de mouse: Todos los botones del mouse tienen un numero, siendo el clic izquierdo el 0, el clic derecho el 1, el clic del medio (el de la rueda cuando la presionamos) el 3, y en caso de tener un mouse con mas botones esto varia según el modelo. El nombre se escribiría "mouse N" siendo N el numero de botón (por ejemplo el clic derecho seria "mouse 1").*

- *Botones de Joystick: Sería el mismo concepto del mouse, cada botón tiene un numero, se escribe "joystick button N" siendo N el numero de botón. Al final veremos una imagen los botones de un joystick de xbox. En caso de querer especificar un joystick concreto podremos poner "joystick M button N" siendo M el numero de joystick y N el numero de botón.*

- *Teclas Función: Serian los F1, F2, F3, etc., se escriben "FN" siendo N el numero de tecla de función.*

- *Teclas Especiales: Serían todo el resto de teclas que nos quedan en el teclado, como "backspace" (retroceso), "tab", "return" (enter), "escape" (esc), "space" (barra espaciadora), "delete", "enter" (enter del numpad), "insert", "home", "end", "page up", "page down".*

- *Tengan en cuenta que no hay que escribir las comillas en el input manager (cuando queremos decir "up" simplemente ponemos: up).*
- *Aquí va una imagen que pone el número correspondiente de cada botón a un joystick de xbox 360, tengna en cuenta que estas configuraciones son para windows, en mac y linux tienen otro número.*

Condicionar la ejecución de un código.

Una vez hecha la configuración inicial para poder efectivamente detectar input, que sería en este caso crear las acciones en el input manager, podemos empezar a usarlas en nuestro juego aprendiendo dos nuevas instrucciones para usar en nuestros componentes.

Instrucción IF

Antes de empezar a ver el IF vale la pena aclarar que tanto esta instrucción como cualquier otra que aprenderemos a lo largo del curso tienen muchos usos más allá de la aplicación concreta que le damos a la hora de explicarlo, parte del desafió de programar es el uso ingenioso y creativo de estas instrucciones mezclándolas para lograr diferentes resultados (como si fuesen piezas de Lego). También veremos que en diferentes estadios del curso le daremos otras utilidades que ayudarán a ejemplificar su uso.

El if es una de las instrucciones más usadas por excelencia en la programación, ya que constituye una de las bases de la lógica de la programación (la forma en al cual la programación propone la resolución de problemas, o sea, la forma de pensar a la hora de programar), sin esta instrucción, no existiría la programación como la conocemos hasta ahora. En su forma mas simple, el if permite "condicionar" la ejecución de un código., con esto nos referimos a que podemos determinar si un código. se ejecuta o no en una situación determinada, por ejemplo, si acabamos de recibir un disparo, restamos el daño recibido a nuestra vida y luego con un if podríamos determinar si la vida llego a 0, haciendo que en esa situación se destruya el objeto y se reproduzca un efecto de sonido (cosa que haremos eventualmente a lo largo del curso), pero en caso contrario, que no se haga nada, también otro ejemplo podría ser el caso en el cual el juego va contando el tiempo que pasó, y si el tiempo llego a 0, se debería perder el juego en el caso de que ello sea una condición para perder el juego (también haremos ello).

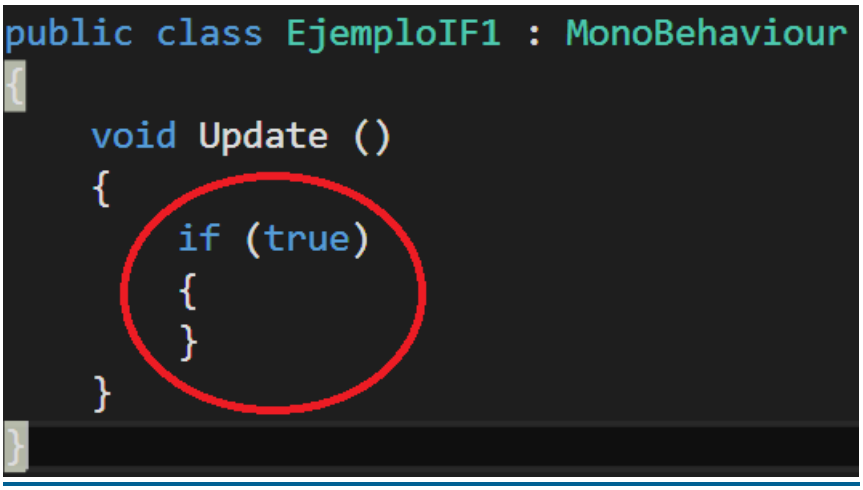
En esta unidad aprenderemos a usar el IF para hacer que el jugador se mueva si esta presionada la tecla correspondiente, por ejemplo, si apretamos arriba que se mueva para adelante, o también si presionamos abajo que se mueva para atrás.

Sintaxis del IF

Al igual que cualquier instrucción, el if tiene una sintaxis que hay que respetar, recordemos que si no escribimos la instrucción exactamente como se debe terminaríamos con un error que nos impide la ejecución de nuestro juego, a pesar de que el componente en el cual esta el error no este en uso.

La sintaxis del if en principio es similar a la de la mayoría de las instrucciones, recordemos que en la mayoría va el nombre de la instrucción seguida de paréntesis y entre los paréntesis, de ser necesario, irían los parámetros de la instrucción, o sea, los datos que necesita la instrucción para ejecutarse (por ejemplo sabemos que print("Hola"); imprime el mensaje Hola en la consola porque así lo especificamos entre paréntesis) y finalmente punto y coma (";").

En el caso del if, en vez de ";" va un conjunto de llaves al igual que en los eventos ({ }), recordemos que el conjunto de llaves sirve para especificar bloques de código., siendo estos, conjuntos de código. asociados a la instrucción que se encuentre sobre ellos. Por ejemplo, en el caso de los eventos teníamos la instrucción para especificar la creación de un evento ("void Update()" por ejemplo) y debajo inmediatamente iban un par de llaves, donde el código. que poníamos adentro, se ejecutaba en el evento, estando asociado a él.



Como vemos en la imagen, tenemos un if dentro de un Update, eso significa que todo el tiempo vamos a estar ejecutando el if. Recordemos que todo lo que pongamos en el Update

se ejecuta constantemente ya que el juego es como una película, o sea, son muchas imágenes que se muestran rápidamente, y con el update podemos modificar la siguiente imagen. También más allá del lugar concreto donde esta puesto el if, recordemos que como cualquier otra instrucción, el if puede ir dentro de cualquier evento, como el Awake o el Update y cualquier otro que usemos o creemos a lo largo de nuestro proyecto.

El tipo de datos Bool

Ahora, el siguiente detalle a tener en cuenta es el tipo de parámetro del if. Recordemos que los datos en programación se separan en tipos, hay tipos para números, para textos, para tildes (los booleanos), etc., y que tanto las propiedades como parámetros son datos, la primera siendo configuraciones del componente y la segunda siendo configuraciones de las funciones.

También recordemos que la instrucción print recibía un texto, que sería un valor del tipo llamado "string", la instrucción "transform.Translate" recibía 3 números separados por coma, o sea, tres datos de tipo "float", en el caso del if recibimos un único parámetro del tipo "bool".

Bool es un tipo de datos que a diferencia de los otros vistos hasta ahora, como los floats en donde podemos poner una cantidad amplia de valores (no infinita, ya que por cuestiones de memoria, se limita el rango de un valor, siendo este rango lo suficientemente amplio para no tener que preocuparnos por el momento), el bool solamente tiene 2 posibles valores: "true" (verdadero) y "false" (falso). Esta limitación de valores se debe a que el tipo "bool" es un dato específicamente pensado para determinar verdades, o sea, para especificar cuando algo es cierto o no (como en el caso de una tecla, puede estar presionada o no).

Ahora el if al recibir booleanos solamente puede recibir dos valores, true o false, si el if recibe el valor true, el código. que está entre sus llaves se ejecuta, si recibe un false no, lo que nos sirve para condicionar acciones.

```
public class EjemploIF1 : MonoBehaviour
{
    void Update ()
    {
        if (true)
        {
            print("Esto se va a ejecutar");
        }

        if (false)
        {
            print("Esto no");
        }
    }
}
```

En la imagen del ejemplo vemos dos ifs, el primero, como recibe el valor "true" ejecutará el código entre sus llaves (en este caso solamente tenemos una sola instrucción print, pero puede haber mas instrucciones), el segundo, al recibir un false no.

Ejercicio N°9

- *Cree un nuevo componente y haga que el mismo mueva un objeto constantemente hacia adelante (como vimos en la unidad anterior, que usamos la instrucción "transform.Translate" dentro del evento "Update").*
- *Modifique el componente para poner el "transform.Translate" adentro de un if que recibe un true por parámetro (fijarse en la imagen de la pagina anterior, donde hay un if adentro del update, y adentro del if hay un print, tendría que quedar similar pero dentro del if, en vez del print, tiene que estar el transform.Translate, el segundo if de la imagen no tiene que ir).*
- *Ponga el componente en un cubo, ponga una cámara que mire hacia el cubo y dele Play al proyecto, el cubo tiene que moverse según lo indicado.*
- *Ahora modifique el componente para que el if en vez de recibir un "true" reciba un "false", guarde los cambios y vuelva a Unity, dele Play al proyecto y vea como esta vez el cubo no se mueve, ya que el if recibió un false.*

Entonces, de esta forma, nosotros podemos condicionar el código. que esta dentro del if para que se ejecute o no dependiendo de que le pasamos por parámetros, pero hasta ahora en los ejemplos le pasamos un valor fijo, o lo que se conoce como un valor constante, o sea, al pasarle expresamente true o false, eso hace que ese valor no pueda cambiar, por lo que el uso que le dimos hasta ahora al if no tiene sentido, ya que seria lo mismo que no ponerlo si sabemos lo que va a pasar siempre, necesitamos que el if se ejecute a veces si y a veces no, dependiendo de una condición, para ello necesitamos pasarle por parámetros algo que varié, que cambie. Repasemos como pasar datos a una función.

Propiedades como parámetros

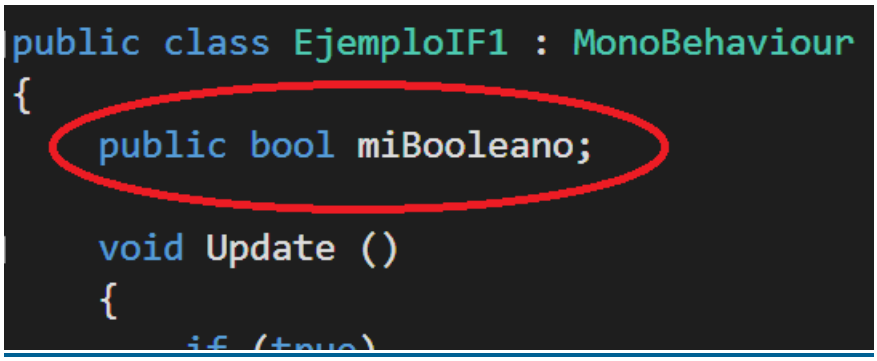
La primera forma de pasarle datos a una función, más allá de especificar el valor concreto, es a través de propiedades, como vimos en la primera unidad. Profundicemos un poco sobre esto. Primero, recordemos que las propiedades tienen tipos y son configuraciones de un componente, pero hay que recalcar la diferencia entre una propiedad y un tipo, un tipo es el formato de un dato, o sea, que representa el dato, un número, un texto, un booleano, etc., mientras que una propiedad es una configuración de un tipo específico, no es un valor, es un contenedor de un valor, como una caja, donde adentro le ponemos el valor concreto que deseamos, pero lo que tiene de bueno, es que al ser una caja, podemos cambiarle el valor cuando deseemos.

Por ahora nosotros a las propiedades las configuramos desde el editor, o sea, que le decimos qué valor contiene una propiedad en el editor antes de darle Play al juego, esto no implica que más adelante no puede cambiar, ya que en realidad, una propiedad es algo variable, algo que sabemos que tiene un valor adentro, pero que no lo sabemos con exactitud, cada vez que tenemos que leer su valor, ver qué valor tiene adentro, puede tener algo diferente, ya que algo pudo haber modificado su valor previamente. Por ejemplo, si quisiésemos podríamos comprobar todos los frames si la vida llego a 0. Y esto podríamos hacerlo con todos los frames, porque que el valor de la vida pudo haberlo cambiado otro objeto, en otro lado. Por ejemplo, una bala cuando choca un objeto (hay más sobre este tema en las siguientes unidades). Entonces, como el valor de la vida pudo haber cambiado en otro lado, podemos chequear en todos los frames el valor de la vida por si cambió, o sea, fijarnos en la caja que tiene la vida, qué valor tiene constantemente, por si cambia.

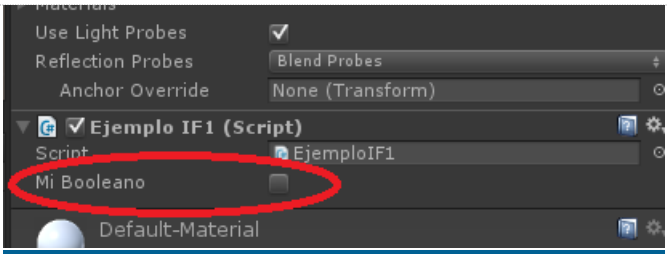
Este aspecto hace que una propiedad al poder variar su valor pueda variar el comportamiento del programa con respecto a su valor. Por ejemplo, en la unidad anterior nosotros usamos una propiedad de tipo float llamada velocidad para variar la velocidad del objeto al que movía el componente que creamos, o sea, que cada vez que Unity ejecutaba el "transform.Translate" tenia que fijarse en la propiedad velocidad qué valor tenía, ya que puede variar. Por ejemplo, si tomamos un powerup que suba la velocidad del personaje, simplemente bastaría con que al detectar que chocamos a ese powerup cambiemos el valor de velocidad para aumentarlo, así el código. del translate sería el mismo, lo que cambia es el valor de la propiedad.

Propiedades como parámetros de un if

Ahora volviendo al if, al tener que recibir un dato booleano como parámetro, además de poder especificar el valor concreto como hicimos antes, podemos pasar una propiedad de tipo booleana, ya que una propiedad de este tipo adentro tendría un booleano. Para crear una usamos la sintaxis de la imagen debajo, recordemos que por ahora la palabra public va siempre, que las propiedades se escriben afuera de las llaves de cualquier evento, pero dentro de las llaves del componente, que sí llevan punto y coma (";"), pero estando antes el nombre de la propiedad.



En el editor la propiedad se vería como la imagen de abajo, donde percibimos que el editor nos permite tildar y destildar la propiedad, o sea, que cuando esta tildada, la propiedad vale "true" y cuando esta destildada, la propiedad vale "false".



Con esto podemos hacer que nuestro componente tenga una propiedad que cuando la tilden el componente haga algo, pero cuando la destilden, deje de hacerlo, como hace el Rigidbody con la propiedad "Use Gravity" para determinar si aplica o no aplica gravedad al objeto, si bien esto no vamos a usarlo directamente para el caso del input, es algo útil en otras situaciones.

Finalmente, para poder utilizar esta propiedad en nuestro código. recordemos que basta con poner el nombre de la propiedad en el lugar donde pida un dato del mismo tipo, en el caso de translate que pedía números le pasábamos nuestra propiedad velocidad que también era un número, en este caso vamos a pasarle esta propiedad booleana a un if para que el código. del if se ejecute o no dependiendo del valor de la propiedad.

```
public class EjemploIF1 : MonoBehaviour
{
    public bool miBooleano;

    void Update ()
    {
        if (miBooleano)
        {
            print("Esto se va a ejecutar si la propiedad 'miBooleano' esta tildada");
        }
    }
}
```

En este caso, este componente va a imprimir el mensaje si tildaron la propiedad del editor, en caso contrario no.

Ejercicio N°10

- *Modifique el componente hecho en el ejercicio 7 para agregarle una propiedad booleana llamada "puedeMoverse" al componente.*
- *Luego utilice esa propiedad como parámetro del if, reemplazando el valor que hay adentro del if y poniendo el nombre de la propiedad en sí.*
- *Vaya al editor y vea cómo la propiedad viene por defecto en false, dele Play al proyecto y vea cómo el objeto no se moverá en este caso.*
- *Pare el proyecto, tilde la propiedad y vuelva a darle Play, vea cómo el objeto ahora si se mueve.*

De esta forma se puede hacer que el valor de una propiedad determine si un código. se ejecuta o no, pasándole el valor a un if. También vea que más allá del funcionamiento interno el código. queda bastante legible. En el caso del ejercicio tendría que haber quedado "if(puedeMoverse)", sabiendo que la traducción de if en ingles es "si" (en el sentido condicional, no afirmativo) se puede leer, "si puede moverse entonces traslada al objeto".

Funciones como parámetros

Otra forma de pasarle datos a una función. es con otras funciones, aquellas que se transforman en un valor, lo que también vimos en la unidad anterior con el caso de la multiplicación de la velocidad por Time.deltaTime. Aquí empezamos a ver cómo ciertos

patrones se repiten a lo largo de toda la programación. En el caso de la unidad anterior vimos que podíamos multiplicar valores con el operador "" (por ejemplo "velocidad * Time.deltaTime) y que esa instrucción tenia la particularidad de convertirse en un valor, en este caso el resultado de la multiplicación. Hay varias instrucciones que se transforman en valores, en este caso, veamos una nueva, llamada "Input.GetButton".*

La instrucción "Input.GetButton" es una instrucción que recibe por parámetro texto, o sea, un string, en este caso la instrucción recibe el nombre de una acción que esté configurada en el Input Manager (como "Fire1", "Jump" o "Reload", la que configuramos en los ejercicios anteriores). Esta, a su vez, se transforma en un booleano (true o false), indicando si en ese momento está o no presionada alguna de las teclas asociadas a la acción que se le pasó por parámetro (por ejemplo "Input.GetButton("Fire1")").

Ahora, como el operador "" se podía poner como parámetro de transform.Translate ya que la misma se transformaba en un numero, nosotros podemos poner Input.GetButton dentro de un if ya que se transforma en un booleano, y así, condicionar la ejecución del if según si la tecla está presionada o no.*

```
public class EjemploIF1 : MonoBehaviour
{
    public bool miBooleano;

    void Update ()
    {
        if (Input.GetButton("Fire1"))
        {
            print("Esto se va a ejecutar si alguna de las teclas");
            print("asociadas a la accion Fire1 esta presionada");
        }
    }
}
```

En este caso, los dos mensajes se van a imprimir en la consola mientras el input de "Fire1" esta siendo presionado. Se leería "Si el botón Fire1 esta presionado...".

Ejercicio N°11

- *Modifique el componente hecho en el ejercicio 8 para hacer que el if en vez de recibir el valor de una propiedad, reciba el valor que da la función Input.GetButton de la acción Reload hecha en los ejercicios anteriores ("if(Input.GetButton("Reload"))"). Vea el código. de ejemplo de la imagen de la pagina anterior como referencia.*

- *Dele Play al proyecto y vea como el objeto sólo se mueve cuando apretamos alguna de las teclas asociadas a "Reload", como la tecla "R", pero si no presionamos ninguna no se mueve.*

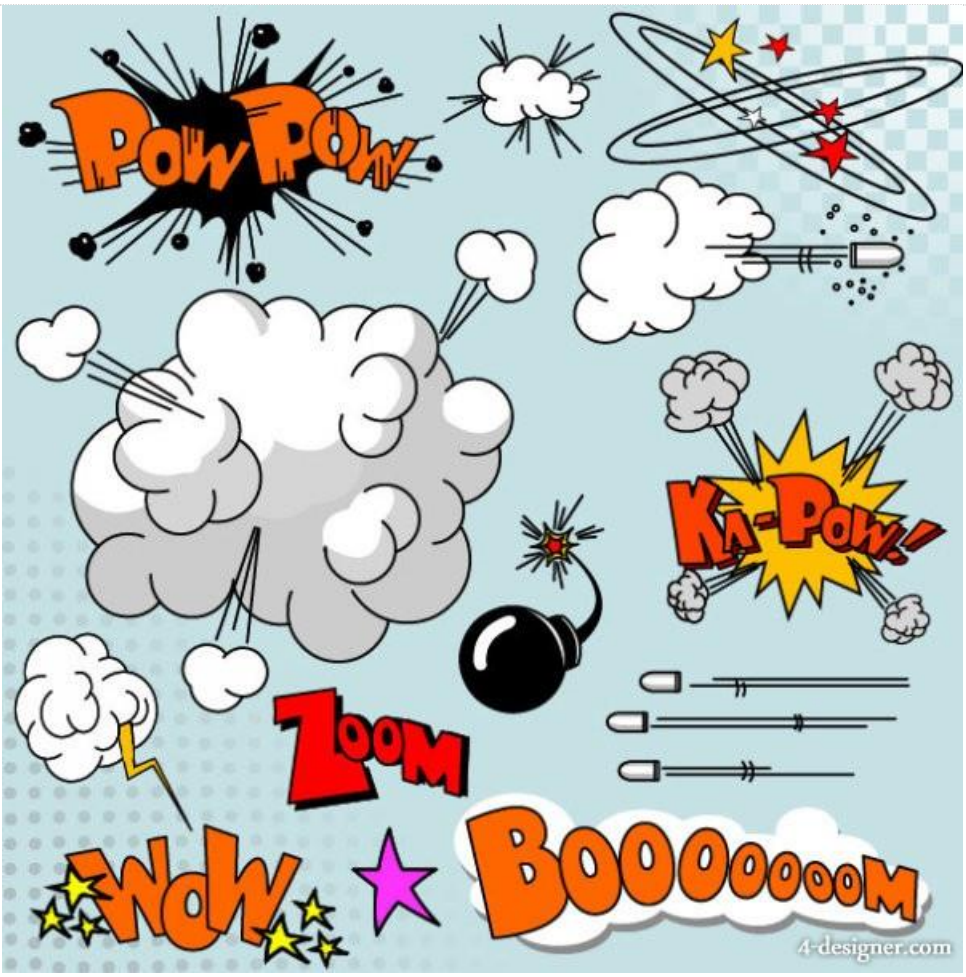
Ejercicio N°12

- *Cree una propiedad de tipo "string" llamada "actionName" ("public string actionName;") en el componente del ejercicio anterior y en vez de pasar el valor explicito "Reload" a la instrucción GetButton, pasar esta propiedad ("if(Input.GetButton(actionName))").*

- *Vaya al editor y configure la propiedad poniendo "Fire1" como valor. Vea cómo al darle play al proyecto, el objeto se mueve cuando presiona el clic izquierdo del mouse o el control izquierdo.*

- *Pare el proyecto y configure ahora en la propiedad el valor "Reload", dele play al proyecto y vea como ahora en vez de moverse con el mouse, se mueve cuando presiona la tecla "R".*

Creación y Destrucción de Objetos



Disparar creando Objetos

Hasta ahora la única forma de crear objetos es desde el editor, tanto creando objetos predeterminados (cubos, esferas, etc.) como haciendo instancias de prefabs. Ahora cuando

hagamos del juego un ejecutable y lo empecemos a distribuir, los que jueguen al juego no van a poder arrastrar un prefab para disparar, ya que ellos no verían el editor, ni tampoco seria algo tan práctico, porque disparar es algo un poco más complejo. Lo que tenemos que aprender a hacer es crear objetos con una instrucción de programación, la cual podemos ejecutar, por ejemplo, cuando se presiona la tecla espacio.

Esta instrucción se llama Instantiate, y se encarga de clonar objetos que reciba por parámetro, o sea, que si le pasamos como parámetro a Instantiate un GameObject (así se llaman los objetos que creamos desde el editor que componen a la escena) esta instrucción lo va a clonar. Antes de profundizar en el uso de esta instrucción primero necesitaremos tener una propiedad que represente un Objeto, veamos cómo.

Objetos como propiedades

Hasta ahora todas las propiedades que hemos creado fueron propiedades muy simples, aquellas cuyo valor lo podemos construir simplemente poniendo el número, el texto o tildando una opción. Ahora, a veces necesitamos que una propiedad sirva de nexo entre dos objetos, o sea, que una propiedad sirva para desde un objeto, acceder a otro, como es el caso de disparar, en el cual el objeto personaje tiene que poder clonar otro objeto, como una bala, cada vez que se presiona una tecla, aquí es donde entran los tipos de datos como GameObject.

Tanto el tipo "GameObject" (es importante escribirlo tal cual, ya que la palabra "gameObject" con g minúscula se refiere a otra cosa), como otros que vamos a ver más adelante, son propiedades, que a diferencia de las vistas hasta el momento, las cuales el valor del dato lo podemos tipear directamente (como un número o un texto), a este tipo de propiedades se les tiene que indicar a qué objeto se van a conectar, o sea, son propiedades para conectar un objeto con otro y que así puedan comunicarse.

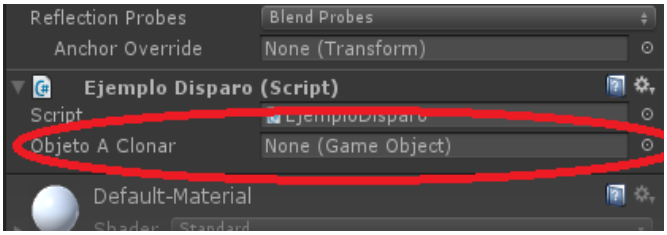
Como vimos hasta ahora, todo en Unity se forma con la unión de varios componentes, o sea, muchos objetos que tienen una conexión entre si, como los componentes vistos hasta ahora (MeshRenderer se conecta a Transform para saber en dónde tiene que dibujar la geometría). Este concepto es algo básico para programar en Unity así que lo veremos repetido en varias situaciones.

En este caso, vamos a necesitar que nuestro componente se conecte con otro objeto (la bala) para poder clonarlo, pudiendo, así, crear objetos cuando deseemos (cuando presionamos una tecla en este caso). Para ello, simplemente creamos una propiedad de

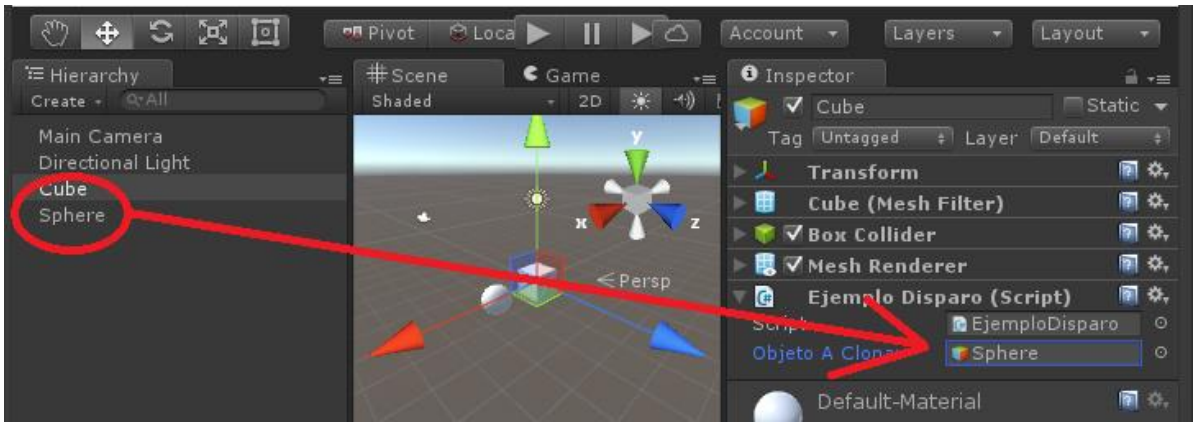
tipo “GameObject”, recalcando la necesidad de respetar las mayúsculas y minúsculas a rajatabla.

```
public class EjemploDisparo : MonoBehaviour
{
    public GameObject objetoAClonar;
}
```

Ahora que especificamos que nuestro componente necesita conectarse a otro objeto para poder acceder a él, necesitamos configurar la propiedad desde el editor, o sea, darle un valor. Si vemos en el editor, la propiedad en este caso no permite tipear valores, aunque hagamos clic y empecemos a escribir, no va a pasar nada, de hecho, vemos la leyenda “None (Game Object)”, indicando que la propiedad tiene un valor nulo, no esta conectada a nada aún.

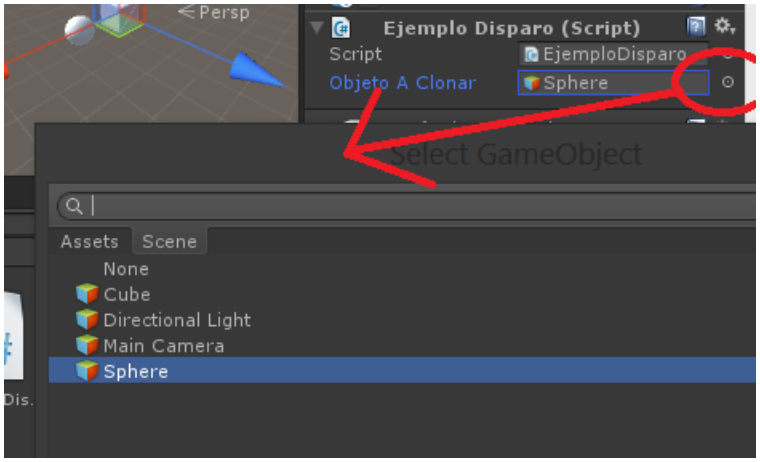


Para conectar un objeto a esta propiedad tenemos dos formas. La primera es arrastrar el objeto al cual queremos conectar la propiedad desde el panel de Hierarchy directo hacia la propiedad, cuando soltemos el objeto vamos a ver que ahora en la propiedad aparece el nombre del objeto, indicando así que la propiedad esta conectada a un objeto llamado de esa manera.

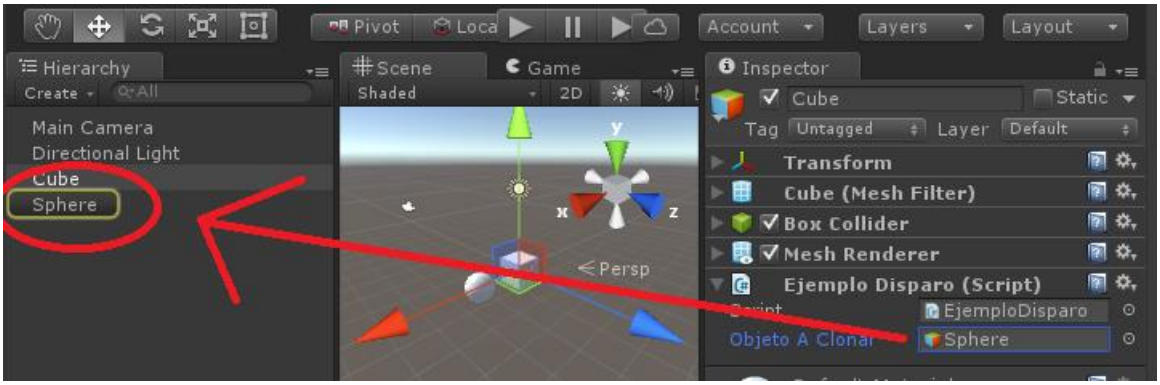


La segunda opción es haciendo clic en el botón que está a la derecha de la propiedad, aquel que tiene la forma de un circulo con un punto en el medio, y luego seleccionando un

objeto de la ventana que nos aparece haciendo doble clic al objeto, siendo esta ventana el selector de objetos, el cual nos muestra todos los objetos que hay en el proyecto. Tengamos en cuenta que esta ventana posee dos solapas en la parte superior, por lo que debemos asegurarnos para este caso, que esté seleccionada la solapa que dice Scene, así solamente aparecen los objetos que están en la escena. Veremos la otra solapa más adelante.



Hay que tener en cuenta que varios objetos pueden tener el mismo nombre, sin embargo la propiedad solamente se puede conectar a uno, en este caso estaría conectada al objeto que le arrastramos. Pero si no estamos seguros a cual esta conectada, ya que podemos haber conectado esta propiedad hace varios días y no recordemos a qué, simplemente alcanza con hacer clic en la propiedad y veremos como se resalta en el panel de Hierarchy.



La instrucción Instantiate

Ahora que nuestro componente se conectó a un objeto, pudiendo acceder a éste a través de la propiedad, podemos finalmente usar la instrucción Instantiate, pasándole por parámetro la propiedad, indicándole a Unity que deseamos clonar el objeto al cual está conectada la propiedad. En este caso lo haremos en el Start, haciendo que nuestro componente clone al objeto una sola vez cuando el juego inicie, ya que si lo hubiésemos hecho en el Update, hubiésemos visto al objeto clonarse todos los frames, empezando a llenarse el panel de Hierarchy con clones y mas clones. Un detalle a tener en cuenta es que el objeto se clona tal cual está, por lo que las copias van a aparecer en el mismo lugar y en la vista de juego no se va a poder apreciar donde están los clones: Para verlos hay que fijarse en el panel de Hierarchy cómo van apareciendo.

```
public class EjemploDisparo : MonoBehaviour
{
    public GameObject objetoAClonar;

    public void Start()
    {
        Instantiate(objetoAClonar);
    }
}
```

Ejercicio N°13

- *Cree un nuevo componente llamado "Clonador" con una propiedad de tipo GameObject y póngaselo a un objeto vació (como si fuese un spawn point).*
- *Cree una esfera y conéctela a la propiedad del componente Clonador que creamos anteriormente arrastrándola o seleccionándola en el selector de objetos.*

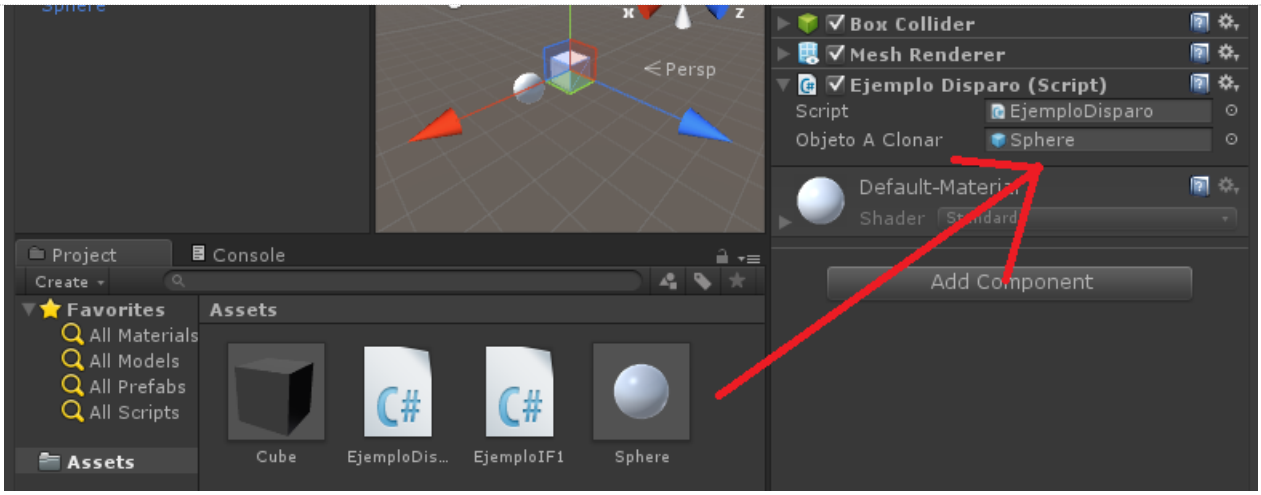
- *Haga que en el Update se instancie dicho objeto, haciendo que ese objeto se clone todo el tiempo.*

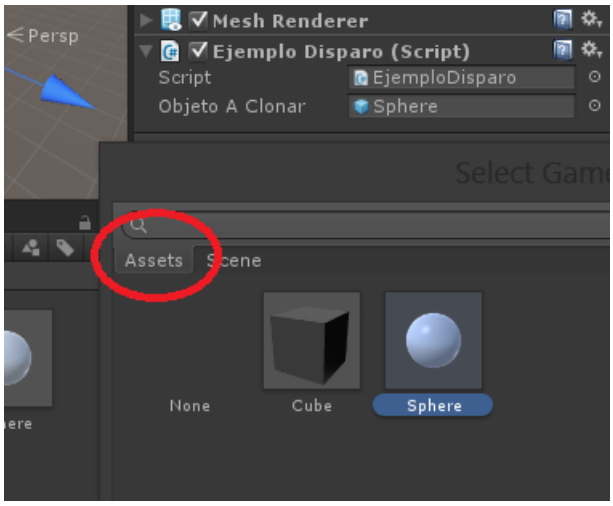
- *Dele Play al proyecto y vea cómo empiezan a hacerse copias en el panel de Hierarchy todos los frames, no deje reproduciendo este código. mucho tiempo, ya que cuantos más objetos haya, más cosas tiene que procesar Unity, y va a llegar un momento en el cual va a comenzar a trabarse.*

Clonando Prefabs

Hasta ahora vimos cómo clonar otros objetos de la escena, pero la verdad es que esto raramente es necesario, en realidad lo que nos interesa es clonar objetos que no están en la escena, ya que, por ejemplo, las balas no deberían estar en la escena a menos que presionemos la tecla espacio, o sea, que si no hay ninguna bala en la escena, no podríamos arrastrársela a la propiedad. Para solventar este problema, podemos utilizar los prefabs.

Los prefabs, al igual que los objetos de la escena, también son objetos, o sea, GameObjects, por lo que nada nos impide en vez de arrastrar un objeto de la escena a la propiedad, arrastrar un prefab, o también seleccionarlo en el selector de objetos, esta vez seleccionando la pestaña "Assets", donde ahora en vez de mostrarnos los objetos de la escena, nos mostraría todos los prefabs que tenemos en el proyecto.





Especificando la posición y rotación de las copias

Ahora nos queda un último problema. Cuando hacemos copias de objetos, estas van a aparecer en la misma posición y rotación del objeto copiado, lo cual normalmente no es lo que queremos. Lo que sí queremos es que la copia aparezca en el mismo lugar del objeto que lo copió. Por ejemplo, en el caso del personaje, la bala va a aparecer en el mismo lugar del personaje, dando a entender de que el personaje disparó esa bala, si aparece en otro lado no sabríamos quien la disparó.

Así como un componente tiene muchas instrucciones hechas (como transform.Translate), también viene con propiedades hechas (como Time.deltaTime), en este caso necesitamos acceder a la posición y rotación de nuestro propio objeto, para ello tenemos la propiedad "transform.position" y "transform.rotation", indicando respectivamente la posición y rotación en la que estamos (ampliaremos este tema más adelante).

Ahora que podemos acceder a nuestra propia posición y rotación, para decirle a Instantiate que cree la copia del objeto en esa locación, simplemente le pasamos la posición y rotación deseada como segundo y tercer parámetro, indicando que allí es dónde va a aparecer la copia, o sea, en nuestra propia posición. Más adelante veremos como ubicar la copia en otros lugares.

```
public class EjemploDisparo : MonoBehaviour
{
    public GameObject objetoAClonar;

    public void Start()
    {
        Instantiate(objetoAClonar, transform.position, transform.rotation);
    }
}
```

Ejercicio N°14

- *Modifique el componente clonador para hacer que la copia aparezca en el mismo lugar del objeto que posee el componente clonador utilizando el código. de la imagen. Pruebe el código.*

- *Ahora, en vez de ejecutar el instantiate en el Start, haga que se ejecute en el update pero dentro de un if que chequee si se presiono la tecla "Fire1", al igual que hicimos con los mensajes en los ejercicios anteriores.*
- *Cree un prefab de una esfera y arrástrelo a la propiedad del componente Clonador que está en el objeto hecho en ejercicios anteriores.*
- *Vaya al editor, dele Play al proyecto y vea cómo ahora las copias aparecen en el lugar del clonador cuando presiona la tecla espacio o el clic izquierdo del mouse.*
- *Cambie de lugar el clonador y vea cómo al hacerlo quedan las copias en el lugar donde estaban, y ahora cuando vuelva a presionar espacio, las nuevas copias aparecen en el nuevo lugar.*

Detectar si se presionó o soltó la tecla.

Un problema con el ejercicio anterior es que, si bien nosotros tenemos que preguntar constantemente si una tecla esta presionada, si lo hacemos de la forma que lo plantea el ejercicio, aunque nosotros presionemos y soltemos lo más rápido posible, va a aparecer

más de una copia, ya que en ese lapso en el cual nosotros presionamos y soltamos una tecla, sin importar que tan rápido lo hagamos, va a pasar más de un frame, ya que estos se ejecutan muy rápido. Por lo tanto, para hacer un sistema de disparo esto es muy inestable. La solución es detectar el momento exacto en el que se presionó una tecla, lo que se logra con la instrucción Input.GetButtonDown.

La instrucción Input.GetButtonDown funciona al igual que Input.GetButton. La diferencia es que GetButton devuelve true mientras la tecla esté presionada, y hasta que no soltemos la tecla no va a dejar de dar true. Recién cuando no estamos presionando ninguna de las teclas que están asociadas a la acción, va a devolver false. En cambio GetButtonDown solamente da true en el frame exacto en el cual la tecla acabo de presionarse, y hasta que no soltemos la tecla y volvamos a apretar va a dar false. Una instrucción similar es Input.GetButtonDown que, en cambio, da solamente true en el frame en el cual la tecla acabo de ser soltada.

Ejercicio N°15

- *Modifique el componente clonador para que, en vez de usar Input.GetButton, use Input.GetButtonDown*
- *Vea cómo ahora los objetos se crean cada vez que soltamos y apretamos la tecla, en vez de crearse constantemente, resolviendo así el problema que teníamos hasta el momento.*

Destruyendo Objetos

Finalmente, la última instrucción que vamos a aprender en esta unidad es la que nos permite, en vez de crear objetos, destruirlos. Esta instrucción se denomina "Destroy".

La instrucción Destroy

Esta instrucción tiene el efecto contrario a Instantiate. En vez de crear un objeto, lo destruye. Y al igual que el instantiate debemos especificarle por parámetro el objeto que queremos destruir.

```
public class EjemploDestuir : MonoBehaviour
{
    public GameObject objetoADestruir;

    void Start ()
    {
        Destroy(objetoADestruir);
    }
}
```

Ejercicio N°16

- *Cree un componente llamado "Destructor" que pregunte en el Update si se presionó una tecla (como hicimos en los ejercicios anteriores) y que dentro del If ejecute la instrucción Destroy, la cual recibe por parámetros una propiedad de tipo GameObject que crearemos en el mismo componente.*

- *En el editor ponga el componente en un cubo y arrástrele la cámara a la propiedad del destructor, vea cómo cuando presiona la tecla durante el juego desaparece la cámara.*
- *Ahora pruebe arrastrar a la propiedad el mismo objeto que tiene el componente, por lo que el componente debería destruir al mismo objeto en el cual está puesto, o sea a si mismo.*

Bibliografía utilizada y sugerida

Recursos Online

Videotutoriales de la interfaz de Unity.

http://unity3d.com/es/learn/tutorials/topics/interface-essentials

Videotutoriales de programación de Unity.

http://unity3d.com/es/learn/tutorials/topics/scripting

Libros y otros manuscritos

Creighton, Ryan Henson. Unity 3D Game Development by Example Beginner's Guide. 1° Edición. UK. Packt Publishing. 2010

Lo que vimos:

En esta unidad hemos visto:

1. *Cómo facilitar el trabajo de manipular múltiples objetos con prefabs.*
2. *Cómo configurar el input para luego utilizarlo para ejecutar acciones en el momento que dicho input fue presionado.*
3. *Cómo crear objetos y destruir por código.*

Lo que viene:

En la siguiente unidad veremos cómo hacer para que nuestro objeto empiece a interactuar con otros objetos. Como por ejemplo, chocar contra ellos, destruirlos, quitarles vida, etc.