UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN

DISEÑO DE UN ENTORNO DE PROGRAMACIÓN INTERACTIVA CONTROLADO POR JOY-CONS

MEMORIA PARA OPTAR AL TÍTULO DE
INGENIERO CIVIL EN COMPUTACIÓN

EMILIO VICENTE LIZAMA CASTRO

PROFESOR GUÍA:
FRANCISCO GUTIÉRREZ FIGUEROA

MIEMBROS DE LA COMISIÓN:
FEDERICO OLMEDO BERÓN
AIDAN HOGAN

Este trabajo ha sido parcialmente financiado por Proyecto FONDECYT Nº 11190248

SANTIAGO DE CHILE
2021

Resumen

El pensamiento computacional es una habilidad que se ha vuelto útil en una gran cantidad de disciplinas y en la vida diaria. Por esta misma razón, en estos últimos años se ha querido masificar esta habilidad en la población, especialmente en el nivel escolar. Actualmente, existen herramientas que permiten abordar la enseñanza de esta destreza a niños, siendo una de las más predominantes los lenguajes de programación basados en bloques, como *Scratch*. Todos los lenguajes de programación de este tipo han utilizado como mecanismo de entrada mouse y teclado, o entradas táctiles. Sin embargo, en ninguna de estas se ha utilizado como entrada la tecnología de *motion-tracking* junto a metáforas de interacción natural.

En este trabajo trabajo de título se busca explorar el uso de *motion-tracking* con interacciones naturales dentro de herramientas para la enseñanza del pensamiento computacional a niños de edades entre 10 y 12 años. Para esto, se diseñaron y desarrollaron una biblioteca de metáforas de interacción natural (i.e., gestos) junto a un entorno de desarrollo integrado (IDE). Estos dos utilizan como dispositivo de entrada los mandos de videojuegos Joy-Con, que permite el motion-tracking del usuario.

Para el IDE y la biblioteca, se realizó primero un prototipo de fidelidad media para probar una primera aproximación a la solución. Este fue probado con un grupo de usuarios expertos en el área, donde se utilizó su retroalimentación para generar una versión ajustada del prototipo. Al finalizar esta etapa, se procedió a implementar este último prototipo.

Para validar la usabilidad y funcionalidad del IDE desarrollado, se ejecutó una prueba de concepto con un grupo de estudiantes y egresados de la Universidad de Chile, en donde tuvieron que realizar una cierta cantidad de tareas sobre este. Los resultados de esta prueba mostraron que este IDE era una alternativa novedosa, la cual era funcional, usable y divertida. Sin embargo, esta prueba no fue realizada sobre los usuarios objetivo del IDE, por lo que estos resultados no permiten demostrar del todo la usabilidad del IDE con el grupo etario objetivo.

Este trabajo presenta un primer acercamiento a diseñar un mecanismo que utilice metáforas de interacción natural para la enseñanza del pensamiento computacional en niños. Así pues, como trabajo futuro se propone primero realizar una segunda prueba de concepto para evaluar la usabilidad del IDE sobre los usuarios objetivo. Después de esto, es importante realizar un nuevo experimento para probar el impacto en el aprendizaje del pensamiento computacional, en este mismo grupo de población, que tiene este IDE.

Agradecimientos

Muchas gracias a mi profesor guía por su apoyo a lo largo de la realización de esta memoria. Agradezco también a mi familia, por el apoyo y motivación que me han entregado en este mismo proceso, y durante toda mi carrera universitaria. Por otra parte, doy las gracias a la página Kenney.nl por el uso de algunos assets dentro de esta memoria. Este trabajo de memoria ha sido parcialmente financiado por el proyecto Fondecyt N° 11190248.

Tabla de Contenido

Índice de Tablas

Índice de Ilustraciones

Capítulo 1

Introducción

En esta última década, el pensamiento computacional ha sido un tema de gran importancia a nivel mundial. Al respecto, múltiples países y organizaciones internacionales han realizado esfuerzos para motivar el desarrollo de esta habilidad, especialmente a nivel escolar [2]. A modo de ejemplo, en Inglaterra en el año 2013 se instauró la asignatura de "Ciencias de la computación" para K-12 (desde kínder a cuarto medio en Chile) como un mecanismo para desarrollar esta habilidad en la población escolar [7]. Por otro lado, existe la ONG CODE, la cual es respaldada por empresas como Microsoft, Facebook y Google, entre otros, que se encarga de promover cursos de ciencias de la computación de forma gratuita para que estén disponibles para escolares de todo el mundo [5]. Chile también está trabajando en promover esta habilidad a través del Plan Nacional de Lenguajes Digitales del Ministerio de Educación, proporcionando cursos en línea, convocatorias y formación de docentes en el área [11].

Pero, ¿qué se entiende por pensamiento computacional? Según Jeannette Wing, investigadora y pionera de esta área, el pensamiento computacional engloba "los procesos del pensamiento humano implicados en la formulación de un problema y la expresión de su solución o soluciones de forma que un computador (humano o máquina) pueda efectivamente llevarla a cabo" [17]. Cabe recalcar que esta es una habilidad que todas las personas que hayan estudiado en el área de ciencias de la computación deberían haber desarrollado. Incluso Wing define, de manera alternativa, que el pensamiento computacional es "pensar como un científico de la computación". Sin embargo, esta capacidad no es exclusiva de los profesionales de esta área, sino que es una habilidad útil para todas las personas [17].

El pensamiento computacional puede caracterizarse por un proceso iterativo de tres pasos [13]. A continuación, se presentan estos con las habilidades que se utilizan en cada uno:

1. Definición del problema:
   (a) Formular el problema
   (b) Abstracción, donde se tiene que identificar y extraer la información relevante del problema.
   (c) Reformulación del problema
   (d) Descomposición, que se encarga de dividir el problema en otros problemas más pequeños y manejables.

2. Resolución del problema:
    (a) Recolección de datos y análisis
    (b) Diseño de algoritmos
    (c) Paralelización e iteración
    (d) Automatizar la solución
3. Análisis de la solución:
    (a) Generalización, que es transferir este proceso de resolución de problemas a un rango más amplio de problemas.
    (b) Testing y evaluación, que tiene que ver con analizar la solución en términos de recursos y eficiencia. Además, incluye la detección de errores en la solución propuesta (debugging).

El pensamiento computacional ha sido ampliamente ocupado en las áreas de las ciencias e ingeniería [17]. Sin embargo, esta habilidad no solo está restringida a estas. Incluso se puede aplicar el pensamiento computacional a actividades de la vida diaria. Como ejemplo, se puede considerar el cocinar una comida para muchas personas. Para realizar este trabajo de manera eficiente, se deberían repartir las actividades de forma que se hagan en paralelo, como dejar cocinando carne mientras que se hace una salsa. Otro ejemplo es el proceso de lavandería. Aquí se puede ocupar *pipelining* entre los procesos de lavado, secado y planchado. También se puede buscar algún nombre en una lista, ordenada alfabéticamente. Aquí se puede ocupar búsqueda binaria para encontrarlo de forma más eficiente. Estos y muchos otros ejemplos existen, donde el pensamiento computacional puede ayudar.

Entonces, se tiene que la importancia del pensamiento computacional se debe, especialmente, por su valor multidisciplinario. Debido a su importancia, sería ideal que todas las personas pudieran desarrollar esta habilidad. Sin embargo, la comunidad de investigadores en educación en ciencias de la educación aún no tiene claro cómo se puede enseñar el pensamiento computacional de manera eficiente y efectiva. En especial, un desafío particular aún abierto en la literatura es el de desarrollar el pensamiento computacional en niños y programadores inexpertos.

A diferencia de los niños, la literatura da cuenta de formas para enseñar el pensamiento computacional en adultos a través de las ciencias de la computación, debido a que ya desde hace décadas que se han enseñado tales contenidos en ciertas carreras universitarias. Por otro lado, conceptos abstractos como el pensamiento matemático se han desarrollado, desde hace siglos y ya hay suficiente exploración de cómo enseñar esta habilidad. Esta falta de experiencia en el área hace que surjan distintas preguntas: ¿qué alternativas existen para enseñar pensamiento computacional en niños?, y por sobre todo, ¿cuál es forma más efectiva de enseñarlo?

Existen diversas alternativas en dominios similares, que podrían resultar útiles para abordar concretamente la enseñanza del pensamiento computacional. Por un lado, es posible utilizar la robótica [6] como medio para enseñar dichas habilidades. Por otro lado, se tienen los lenguajes basados en bloques [16], que también son una forma bastante efectiva de enseñar. También hay otras metodologías utilizando herramientas "desenchufadas" [3], es decir, sin utilizar algún dispositivo electrónico. En resumen, existen múltiples alternativas para enseñar

pensamiento computacional a niñas y niños. Sin embargo, no hay respuestas concluyentes de cuál de estas o combinación de estas, es la más efectiva.

Entre las herramientas utilizadas para enseñar pensamiento computacional, existen muchas que necesitan de la interacción con un computador. De estas se mencionaron los lenguajes de programación basados en bloques y la utilización de la robótica. Para que un usuario pueda interactuar con la primera herramienta, se utiliza una interfaz gráfica y como dispositivos de entrada un mouse y teclado. Además de este tipo de interacciones, existen muchas más. Al respecto, el área de "interacción humano-computador" (HCI por sus siglas en inglés), se define como "la disciplina dedicada a diseñar, evaluar e implementar sistemas informáticos interactivos para el uso humano, y a estudiar los fenómenos relacionados más significativos" [9].

En el área de HCI, existen un tipo de interacciones que son hechas por medio de *Interfaces de Usuario Naturales*. Dentro de esta se tienen, por ejemplo, las que son basadas en "motion tracking" (captura de movimiento). Aquí se interpretan los movimientos del usuario, utilizando sensores o cámaras, y se traducen a comandos dentro del software. En el ámbito de los videojuegos, un ejemplo de este mecanismo de navegación y control son los "motion-based controllers", es decir, controles basados en movimiento. En particular, en el estado del arte de estos dispositivos de uso masivo, se tienen los Joy-Con, que son el mando de la consola de videojuegos Nintendo Switch. Estos se pueden ver en la figura 1.1.

Figura 1.1: Mandos Joy-Con de la consola de videojuegos Nintendo Switch.

Como se puede apreciar en la figura 1.1, son dos unidades individuales, en general de distinto color, el cual uno está diseñado para utilizar con la mano izquierda (azul) y el otro con la mano derecha (rojo). Estos pueden ser ocupados de distintas formas. Algunos ejemplos de su uso, se pueden ver en la figura 1.2. En la primera imagen, de izquierda a derecha, de la figura, se tiene la forma de ocupar 1 control en cada mano. La segunda imagen, hace uso del "Grip" donde se pueden "juntar" las dos unidades, tomando la apariencia de un control más clásico. En la tercera imagen se tiene ambas unidades conectadas a la consola. Finalmente, en la última imagen, se ocupa solamente una unidad con las dos manos.

Para poder detectar el movimiento del usuario, cada unidad tiene un giroscopio y acelerómetro. Con esto en mente, la forma de tomar los mandos que permite la mayor libertad de movimiento es la primera (figura 1.2-a), debido a que al tener un Joy-Con en cada mano,

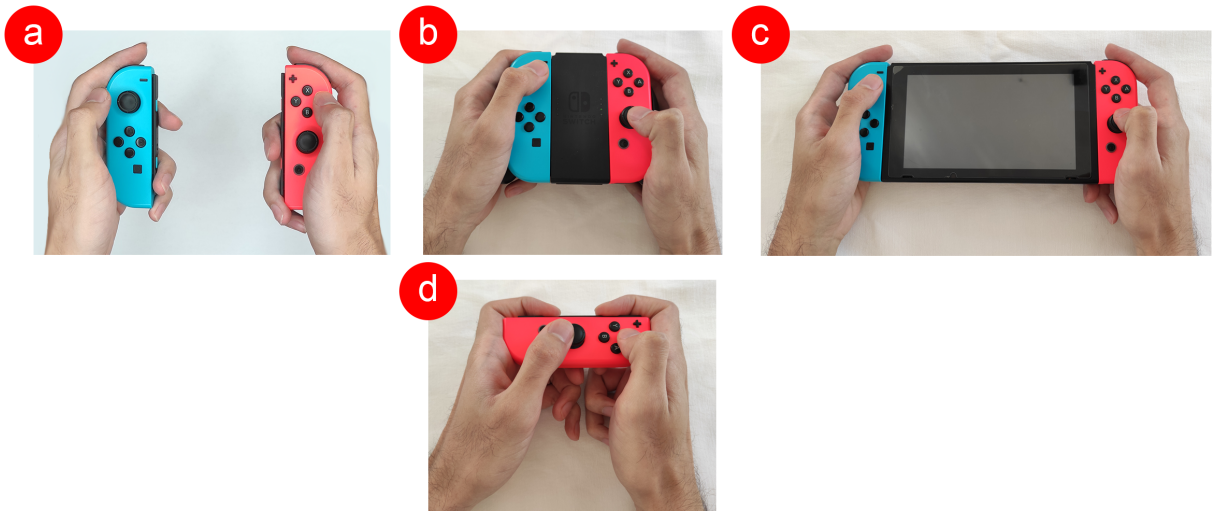


Figura 1.2: Modos de uso de los controles Joy-Con.

permite que se muevan de forma independiente.

1.1. Problema abordado

El uso de "motion based-controllers" ha tenido aplicaciones más allá que de los videojuegos. Por ejemplo, se han visto posibles usos de los Joy-Con para el entrenamiento de médicos en el uso de cirugías con asistencia robótica [1]. Para esto se realizó una investigación, donde su uso entregó resultados favorables en un ambiente controlado. Sin embargo, aún faltan más pruebas prácticas para asegurar la efectividad del uso de Joy-Cons para estos entrenamientos.

Tal como se mencionó anteriormente, la literatura aún no reporta una forma definitiva de cómo enseñar pensamiento computacional en niños. Debido a esto, se podría pensar en otras alternativas como medio para introducir estos conceptos y habilidades en este grupo de población. Así pues, una de las alternativas que actualmente no se ha aprovechado es la de explorar el uso de mecanismos alternativos de interacción (como por ejemplo, a través de interfaces naturales controladas por gestos y controles de videojuegos). Dado esto, en el presente trabajo de título se exploró cómo utilizar los mandos Joy-Con, mediados por una interfaz de software, para desarrollar habilidades básicas de pensamiento computacional. Luego, ¿qué ventajas tiene utilizar este medio a diferencia del resto? Primero, existe familiaridad de los niños con este instrumento. La Nintendo Switch es hoy en día una consola muy popular en este grupo etario, lo que hace que muchos ya tengan acceso a estos dispositivos. Que la consola sea popular se evidencia en que esta ha llegado a un total de 79.87 millones de unidades vendidas a nivel mundial[1] y en que tiene un catálogo de videojuegos en los que el público objetivo son niños como es el caso del juego *Mario Kart Live: Home Circuit*. Volviendo a lo anterior, otra ventaja de los Joy-Con es que los niños lo asocian al juego, por lo que su acercamiento sería más lúdico y amigable. En efecto, de acuerdo a Edwards [8], los niños al creer que están jugando, logran indirectamente que su aprendizaje sea percibido como más efectivo y más fácil de adquirir.

[1]Este valor es del 31 de diciembre del 2020 y fue obtenido de la página oficial de Nintendo, en el siguiente enlace: `https://www.nintendo.co.jp/ir/en/finance/hard_soft/index.html`

Si bien aún no se ha estudiado empíricamente, existen actualmente juegos de Nintendo Switch que podrían servir como vehículo para desarrollar el pensamiento computacional. Entre ellos se puede mencionar "Super Mario Maker 2"[2], que es un juego donde uno programa niveles de la saga "Mario Bros" para después compartirlos o jugarlos uno mismo. Este juego implica el uso indirecto de varias habilidades de pensamiento computacional, tales como la "formulación de problemas" al diseñar el nivel o el de "testing y evaluación", al tener que probar que su nivel pueda ser completado.

Otros ejemplos de juegos actualmente disponibles para la plataforma son "Overcooked"[3] y "Human Resource Machine"[4]. El primero es un juego multijugador donde se tienen que cocinar distintos platos que se van pidiendo a medida que progresa la partida. Esto tiene la dificultad de coordinar la preparación de distintos platos entre muchos jugadores. Aquí la habilidad de pensamiento computacional que se desarrolla es el de "paralelización", al tener que coordinar ir generar los platos al mejor ritmo posible. El segundo, es un juego de puzzles, donde se toma el rol de un oficinista. En cada nivel, se tiene que resolver un puzzle que involucra dar instrucciones al oficinista para que las lleve a cabo. Para esto se ocupan bloques de instrucciones, y uno "programa" lo que va a hacer este oficinista. Estos bloques de código tienen instrucciones similares a las de un lenguaje de assembly. Los bloques se van dejando utilizando una interacción de "drag and drop", utilizando la pantalla táctil de la consola o con el sensor infrarrojo del Joy-Con, donde se mueve como un puntero. Cada vez que se tiene listo el código se puede "correr" para que el oficinista realice las tareas dichas, mostrando de forma visual los pasos del oficinista. En este juego se utilizan muchas habilidades de pensamiento computacional, como lo es la "abstracción del problema", "diseño de algoritmos " y "testing y evaluación". El último, debido a la reproducción visual de lo que hace el oficinista, uno puede probar si el algoritmo diseñado es correcto o no.

Lo que tienen en común estos juegos, es que no ocupan todas las facultades del control para hacer estas interacciones. En general, solo ocupan los botones, palancas analógicas y la pantalla táctil de la Nintendo Switch como inputs para cada juego. Si bien en "Human Resource Machine" se han ocupado los sensores de movimiento para utilizar los Joy-Con como un puntero, no se ha intentado utilizar controles gestuales. Ejemplos de estos últimos, en el caso de interacción mediada por un Joy-Con, sería agitarlo y que el gesto sea detectado por el dispositivo e interpretado apropiadamente por una interfaz de software.

Dado todo lo anterior, en este trabajo de título se trabajó en la creación de una herramienta que sirva como una nueva alternativa para enseñar pensamiento computacional en niños. Esta alternativa consiste en un prototipo de lenguaje de programación, el cual es controlado por los mandos Joy-Con utilizando interacciones gestuales y táctiles. Para lograr esto, se diseñó y desarrolló una biblioteca de programación con estas interacciones y un entorno de desarrollo integrado (IDE). Esto es una contribución al área de educación en ciencias de computación debido a que, por un lado, busca proponer una nueva alternativa de enseñar pensamiento computacional en niños y, por el otro, aporta en el estudio de diseño de me-

[2]Para más información acerca del videojuego Super Mario Maker 2: `https://store.nintendo.cl/super-mario-maker-2`

[3]Para más información acerca de la versión de PC del videojuego Overcooked: `https://store.steampowered.com/app/448510/Overcooked/`

[4]Para más información acerca de la versión de PC del videojuego Human Resource Machine: `https://store.steampowered.com/app/375820/Human_Resource_Machine/`

canismos de interacción natural en HCI, al proponer nuevas metáforas de interacción para promover creatividad y desarrollo de tecnología por niños.

1.2. Objetivos

Objetivo General

El objetivo general del presente trabajo de título es diseñar y desarrollar una biblioteca de metáforas de interacción natural y un prototipo de entorno de desarrollo integrado que permitan a niños de entre 10-12 años desarrollar pensamiento computacional. Esta biblioteca se caracteriza por utilizar como dispositivo de entrada los mandos Joy-Con, aprovechando sus mecanismos para generar interacciones gestuales y táctiles.

Objetivos Específicos

Para abordar el objetivo general propuesto en este trabajo de título, se plantearon los siguientes objetivos específicos:

1. Diseñar, siguiendo una estrategia de diseño participativo, una biblioteca de metáforas de interacción natural que sean consideradas aceptables por niños de entre 10 y 12 años.
2. Diseñar un prototipo de entorno de desarrollo integrado que pueda ser usado por niños de entre 10 y 12 años para crear aplicaciones de software utilizando la biblioteca de metáforas de interacción natural propuesta.
3. Desarrollar una interfaz de software que permita mapear metáforas de interacción natural con un prototipo de entorno de desarrollo integrado de aplicaciones, de tal manera que la interacción sea considerada usable por niños de entre 10 y 12 años.

1.3. Solución desarrollada

La solución que se desarrolló para resolver el problema planteado consistió en diseñar y desarrollar una biblioteca de programación de interacciones gestuales y táctiles, generadas por Joy-Cons. Esta biblioteca fue diseñada para que, por medio de estas interacciones, se puedan desarrollar programas sencillos que cuenten con las siguientes características:

- Instrucciones de movimiento en un plano 2D de alguna imagen
- Instrucciones de bifurcación (if-else)
- Instrucciones de iteración (while)

El alcance de este trabajo se limitó a la definición de los elementos mínimos que permiten la enseñanza de conceptos importantes y básicos de la programación, tales como la idea de secuencialidad, lógica condicional e iteración [15].

Junto a la biblioteca, se desarrolló también un entorno de desarrollo integrado (IDE) para poder desarrollar estos programas. Esta es una interfaz gráfica, la cuál cuenta con un área de trabajo, donde el usuario genera el código y un área de producción, donde se entrega el resultado de correr dicho código. Además, este entorno se desarrolló para que funcione

solamente con computadores con el sistema operativo Windows, pero podría ser expandido en el futuro a otras plataformas.

Como primer acercamiento, el área de trabajo se basó en los lenguajes de programación por bloques, es decir, también se utilizan bloques que representen código, y que se van conectando como piezas. Sin embargo, hay cambios al orientarlo al uso de interacciones gestuales. Por ejemplo, en lugar de ocupar interacciones "drag and drop" para generar bloques, se ocupan gestos que generen este mismo bloque. Por otro lado, el área de producción muestra un plano 2D, donde se reproducen las instrucciones generadas por el código.

Se mencionó que para las interacciones se utilizaron los Joy-Con. De estos, se ocupó su giroscopio para detectar gestos que se generen al mover los Joy-Con de cierta forma. En conjunto a esto, se usaron todos los botones y ambos sticks analógicos para crear una mayor cantidad de inputs.

La biblioteca fue diseñada para que los Joy-Con sean usados como en la figura 1.2-a. Al utilizarlos así, se pudo usar de forma más variada los controles de movimiento, al generar inputs de forma dividida entre ambas unidades.

A continuación, se presenta un pequeño caso de uso de la biblioteca en conjunto con su IDE. Cabe destacar que este no es el diseño final, sino un primer acercamiento de una interacción entre biblioteca y IDE:

Al iniciar el IDE, el área de producción se vería como en la figura 1.3. Aquí se presenta un bloque inicial donde empieza el código, que se llama "Aquí empieza tu código" y un botón "Start" para empezar a correr el código en pantalla. Este botón se activa al pulsar el botón "+" de los Joy-Con.

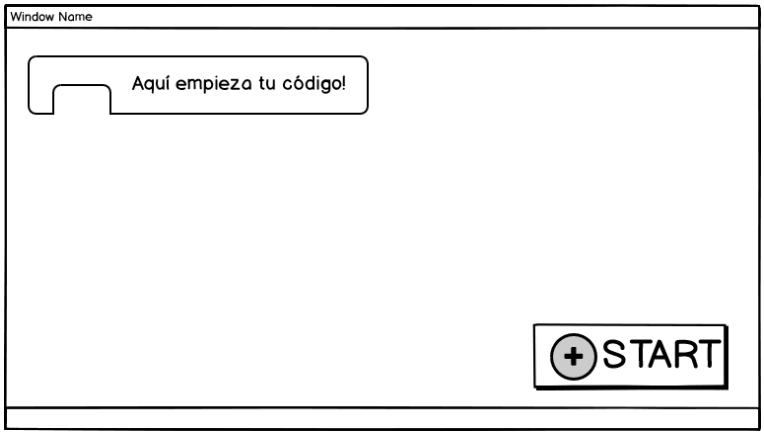


Figura 1.3: IDE al ser iniciado por primera vez, primer acercamiento.

Para generar un bloque de repetición "while", se mueve el Joy-Con derecho en forma circular como se muestra en la imagen izquierda de la figura 1.4. Dentro de la misma figura, en la imagen derecha se encuentra el resultado de realizar ese movimiento, que es el bloque while creado.

El bloque se mueve con el análogo del Joy-Con izquierdo y se suelta con el botón "A",

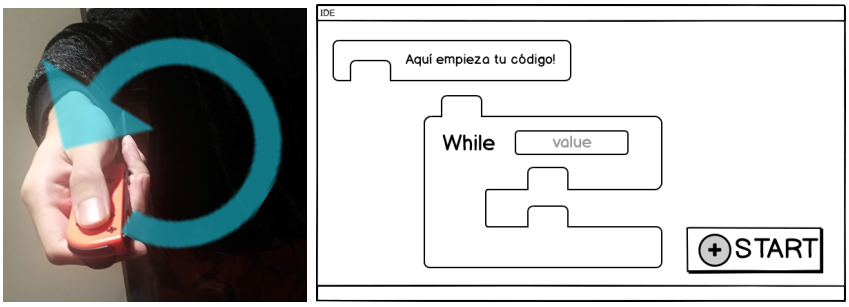


Figura 1.4: Creación de bloque de ciclo while, primer acercamiento.

conectándolo con el inicio del código. El entorno queda como la figura 1.5.

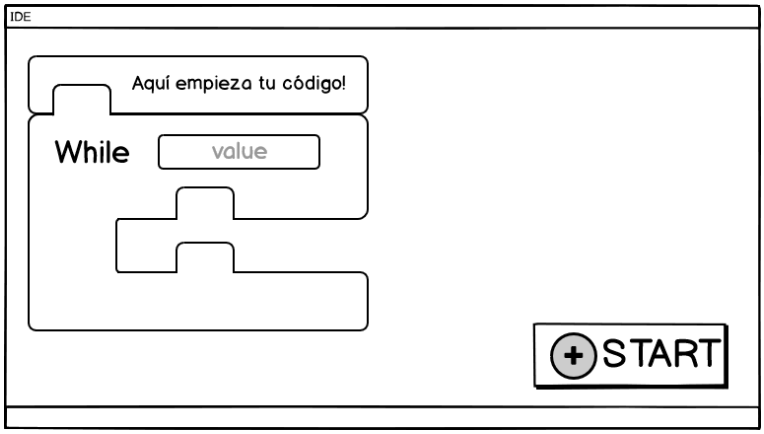


Figura 1.5: Ciclo while conectado, primer acercamiento.

Para colocar un valor en el while, se selecciona el campo "value" al mover el análogo izquierdo y pulsando el botón "A". Después, si se quiere generar un valor booleano, se agita arriba y abajo, el Joy-Con derecho para un valor "False" y el izquierdo para un valor "True". El gesto de agitar el control se puede ver en la primera imagen de la izquierda de la figura 1.6. Dentro de la misma figura, la imagen de al medio muestra el resultado en el IDE cuando se agita el Joy-Con izquierdo, y la imagen de la derecha, el Joy-Con derecho.

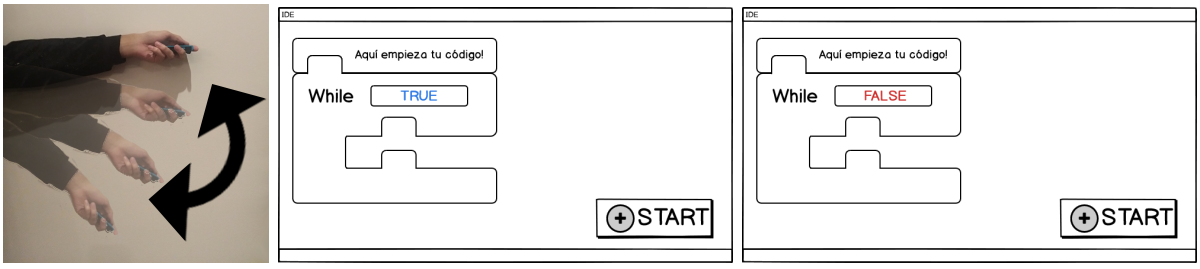


Figura 1.6: Asignación de valores booleanos, primer acercamiento

Continuando de esta forma se puede hacer un código por bloques que quede como en la figura 1.7.

Este código resultante, lo que haría sería mover el sprite del gato, 10 pasos hacia la derecha en cada frame. El resultado de correr este código se puede ver en la figura 1.8, donde las

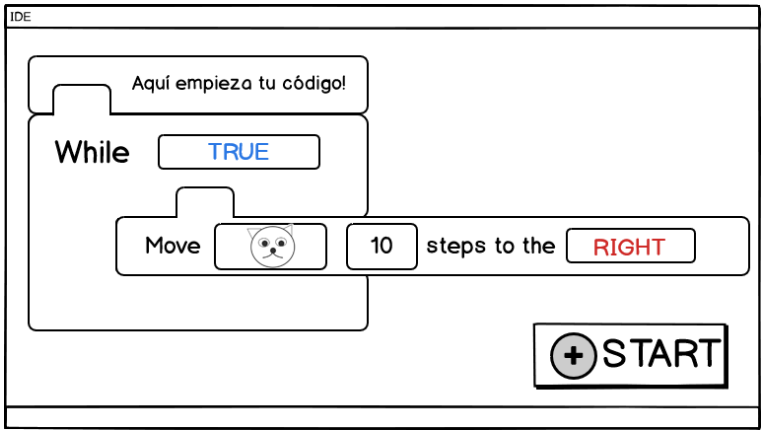


Figura 1.7: Ejemplo de código realizado en el IDE, primer acercamiento.

imágenes, de izquierda a derecha, muestran que el sprite del gato se mueve indefinidamente por lo que al final desaparece de la pantalla

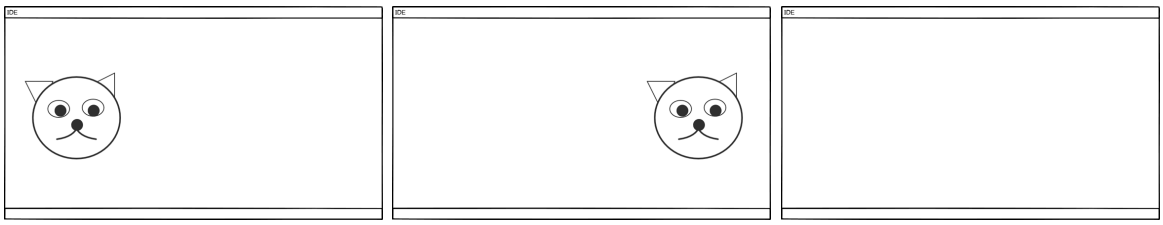


Figura 1.8: Resultado de correr el código de la figura 1.7, primer acercamiento.

Para la implementación de la solución se utilizó el motor de juego "Unity". En este se desarrolló el entorno de desarrollo además de la biblioteca de interacciones. Su elección se debe al extenso soporte que hay de su comunidad, la familiaridad con la herramienta y porque permite el desarrollo a múltiples plataformas. Esta también permite el uso de Joy-Cons con la librería mencionada en el capítulo 2.

A medida que se fue desarrollando la solución, esta se fue validando con distintos grupos de usuarios. Para realizar la validación, se utilizaron distintas técnicas para el desarrollo de sistemas interactivos. Entre estas se encuentran el diseño de distintos prototipos, variando de baja hasta alta fidelidad, la realización de demos con tests de usabilidad, entre otros.

1.4. Organización del documento

El resto de este documento se estructura como sigue. En el capítulo 2 se presenta y discute la literatura relacionada que sustenta el trabajo ejecutado en esta memoria de título. El capítulo 3 expone el proceso de prototipado del diseño del entorno de desarrollo junto al de la biblioteca de metáforas de interacción. En este proceso, primero se diseñó una versión inicial del entorno y la biblioteca, que después fue evaluada su usabilidad con un grupo de usuarios experto por medio de una prueba. Luego, se actualizó el prototipo en función de los resultados obtenidos en esta prueba. A continuación, en el capítulo 4, se detalla la implementación los distintos elementos que componen del entorno de desarrollo. Posteriormente, en el

capítulo 5, se presenta del diseño y los resultados de una prueba de concepto para verificar el correcto funcionamiento y usabilidad del entorno de desarrollo implementado. Finalmente, en el capítulo 6 se presentan las conclusiones y se ofrecen perspectivas de trabajo futuro.

Capítulo 2

Trabajo Relacionado

Tal como fue mencionado anteriormente, una de las soluciones existentes al problema que se abordó en este trabajo de memoria son los lenguajes de programación por bloques. A diferencia de lenguajes más tradicionales, en los que se tiene que escribir código con el teclado cumpliendo con la sintaxis pedida por un lenguaje típico, se utilizan bloques con código pre-hecho, que se van conectando uno a uno. Al conectar todos los bloques deseados, se puede correr el código de la misma forma que se haría con un lenguaje de programación tradicional. Debido a la facilidad de uso que tienen en comparación a lenguajes tradicionales, se utilizan para enseñar pensamiento computacional en niños [14]. Además, son especialmente útiles porque las habilidades aprendidas en estos pueden ser traspasadas a otros lenguajes de programación de más bajo nivel.

Uno de los primeros lenguajes de este tipo es Scratch [14]. Scratch utiliza este sistema de bloques además de un mecanismo basado en eventos para que los usuarios puedan programar juegos, música, historias, tutoriales y muchos más, todos estos de forma sencilla. El público objetivo de este lenguaje son niños de entre 8-18 años e incluso, hay una versión para niños aún más pequeños llamado Scratch Jr., para edades entre 5-7 años. Esta última se diferencia de Scratch, al utilizar bloques que son más simples y que describen su función solamente con un ícono.

Otra herramienta importante para el estudio del problema abordado en este trabajo de título es Blockly diseñada por Google. Esta herramienta es una biblioteca de Javascript que permite desarrollar lenguajes basados en bloques, que son similares a Scratch. Para realizar esto, Blockly cuenta con un editor que puede ser modificado. El editor, tiene una interfaz gráfica similar a la de Scratch, esto es, "drag and drop" de bloques. Su modificación se puede realizar a través de código en Javascript o utilizando los mismos bloques que vienen predefinidos en la biblioteca.

Una de las características de Scratch y Blockly es que ambas se ocupan desde un computador personal (PC), por lo que para interactuar con este se utilizan un mouse y teclado. Por otra parte, Scratch Jr. se ocupa desde dispositivos móviles como smartphone o tablets, por lo que para interactuar con ella se utiliza la pantalla táctil del mismo dispositivo. En general, los lenguajes de programación basados en bloques utilizan mouse y teclado o una pantalla

táctil como dispositivos de entrada.

Por otro lado, en las tecnologías ocupadas en la solución del problema planteado en este trabajo de título, se tienen los Joy-Con. Entre las características que tiene este tipo de control, se pueden mencionar dos palancas analógicas, 22 botones digitales, giroscopio, acelerómetro, vibración HD (HD rumble), sensor infrarrojo (solamente en el Joy-Con derecho) y antena NFC (solamente en el Joy-Con derecho). Este acelerómetro y giroscopio permiten que se pueda detectar movimiento, por lo que se pueden ocupar para detectar gestos. Por ejemplo, si uno agita el mando de la derecha, este puede ser procesado e interpretado por la consola. Asimismo se tiene la vibración HD, que es un sistema de vibraciones que permite reproducir ciertos sonidos a través de estas. La conexión del mando se hace por medio de bluetooth.

Todas las funcionalidades que fueron mencionadas anteriormente se pueden ocupar si los mandos están conectados a la consola Nintendo Switch. Sin embargo, la conexión de estos controles a un computador no es tan sencilla. Los Joy-Con, como no fueron diseñados para ser conectados a un computador y aún no existe soporte oficial por parte de la empresa para hacer esto, se tienen que utilizar herramientas que hayan sido diseñadas por terceros. Entre las herramientas existentes para poder soportar los Joy-Con dentro de un computador se tiene la biblioteca "JoyconLib" [1]. Esta biblioteca sirve para que el mando pueda ser utilizado dentro de algún motor de desarrollo de videojuegos. Al respecto, las funcionalidades que entrega la biblioteca son las siguientes:

- Admite la conexión de más de 1 Joy-Con al mismo tiempo.
- Permite que se realice un sondeo (*polling*) de los botones y palancas analógicas de los mandos conectados. Con esto, se puede detectar si un botón fue pulsado o la dirección en que se encuentran las palancas.
- Permite la utilización de la vibración HD, para generar vibraciones con distinta amplitud, frecuencia y duración.
- Entrega los datos en tiempo real del giroscopio y acelerómetro. Con estos datos se pueden hacer controles de movimiento y gestuales.

Con todas estas características, las únicas componentes del mando a las que no tiene acceso la librería es al sensor infrarrojo y la antena NFC.

Otro elemento importante a considerar en el desarrollo de la solución para el problema planteado en este trabajo de título, es la elección de un motor de desarrollo de videojuegos. Al respecto, Unity es una plataforma ampliamente utilizada en la industria, enfocada en el diseño y creación de software interactivo de estándar profesional. Este motor permite la creación de juegos 3D, 2D, 2.5D y de realidad virtual, así como simulaciones. Esto hace que su uso no se limite solamente a la industria de videojuegos, sino también a otras, como la industria automotriz, la ingeniería y el cine. Los videojuegos creados con Unity pueden ser lanzados a múltiples plataformas (25 en total), tales como PC con sistemas operativos (OS) como Windows y macOS, y consolas como Xbox One, PlayStation4 y Nintendo Switch. Para la creación de estos videojuegos, Unity incluye una plataforma de desarrollo, soportada por Windows, macOS y Linux.

[1] `https://github.com/Looking-Glass/JoyconLib`

Unity cuenta con un sistema de físicas tanto 2D como 3D. Estos permiten que los objetos dentro de un proyecto tengan velocidad, fuerza y fricción. Incluso permite la generación y detección de colisiones entre objetos.

Por el lado gráfico, se cuentan con sistemas para facilitar la animación 2D y 3D, adecuándose al tipo de proyecto a realizar. Asimismo, se incluye un sistema de renderizado e iluminación. Para la lógica dentro de un proyecto se crean archivos de código, escritos en el lenguaje de programación C#.

Los proyectos en Unity se componen principalmente de objetos de juego (*GameObjects*), componentes (*components*) y escenas (*scenes*). Cada objeto dentro de un juego es un GameObject, desde personajes hasta luces y cámaras. Estos objetos, por sí solos no hacen nada. Aquí es donde entran las componentes, que son quienes les entregan la lógica y propiedades. Unity viene con una cierta cantidad de componentes predeterminadas, que entregan, por ejemplo, física a los objetos. Sin embargo, las componentes también pueden ser archivos escritos en el lenguaje C#. Entonces, se tiene que cada objeto de juego, puede tener muchas componentes. Finalmente se presentan las escenas. Aquí es donde se tienen distribuidos los objetos, formando así el entorno y ambiente del juego o simulación. Generalmente, dentro de una escena se tiene un objeto especial de cámara, que es el área que vería el usuario final del proyecto.

En la figura 2.1 se tiene la disposición de un proyecto en Unity. En la siguiente lista, se tienen las descripciones de cada área:

- Project Window(ventana del Proyecto): Se muestran los archivos del proyecto.
- Scene View (vista de escena): Visualiza la escena actualmente abierta. Se puede navegar y editar los objetos que encuentran en esta. Además, existe la Game View, que se visualiza al hacer click en la pestaña "Game". En esta se muestra la reproducción de la escena en tiempo real.
- Hierarchy Window(ventana de jerarquía): Esta área es una representación de texto jerárquico de cada objeto en la escena.
- Inspector Window(ventana del inspector): Muestran las componentes atadas a un objeto de juego seleccionado. Se pueden editar en esta misma área.
- Toolbar (barra de herramientas): En esta zona, se encuentran las herramientas esenciales para trabajar. Entre estas, se encuentra los controles de reproducción del proyecto (empezar, pausa, parar y pasos) y de movimiento de los objetos dentro de la escena.

Con lo anterior, se tiene entonces un resumen de las características que tiene el motor de videojuegos Unity. Por otro lado, en los siguientes párrafos se analiza una aplicación interesante, para el presente trabajo de título, en la que se utilizan los dispositivos Joy-Con.

Desde que fue lanzada la Nintendo Switch, Nintendo ha buscado implementar formas creativas de utilizar los Joy-Con en la consola. Una de las propuestas de la empresa nipona, ha sido "Nintendo Labo". Esta aplicación consiste en un videojuego y un set de piezas de cartón con los que se extiende la experiencia interactiva proporcionada por el software corriendo en la consola. Con este set, uno modifica la consola y los controles, y al utilizar esto en conjunto con el videojuego, permite crear nuevas formas de interacción con la consola, como simular

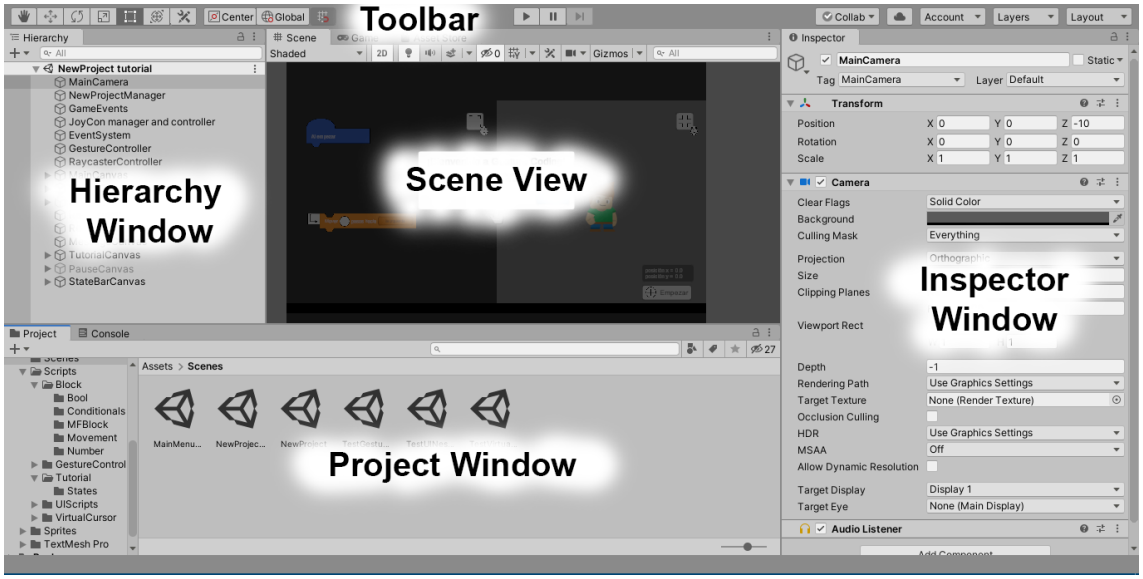


Figura 2.1: Layout de un proyecto en Unity.

un teclado o una caña de pescar con ella.

En la aplicación de Nintendo Labo corriendo en la consola se incluyen instrucciones de cómo armar las piezas de este, así como pequeños videojuegos con los que se puede interactuar. En general, estos videojuegos son bastante estrictos: uno tiene que construir un objeto específico utilizando las piezas de cartón y hay un juego con el que se puede utilizar este objeto. La parte creativa de este software está en la sección llamada "Toy-Con Garage". Esta es una aplicación que permite programar utilizando nodos. En general, permite detectar como inputs las interacciones con los botones y movimientos generados con los Joy-Cons, así como la presión sobre la pantalla táctil de la consola. Como salidas, se pueden desplegar luces en la pantalla, vibraciones en los controles e incluso crear interacciones con un videojuego.

En el entorno de programación, se manejan dos tipos de nodos: de entrada (*input*) y de salida (*output*), los que se unen entre sí con acciones, que corresponden a nodos de cálculo (operaciones matemáticas). La navegación de la aplicación se puede realizar empleando la pantalla táctil de la consola o utilizando el Joy-Con derecho como puntero.

Se tiene entonces que el Toy-Con Garage es una herramienta que permite escribir programas de forma distinta al típico lenguaje de programación, al utilizar un formato por nodos. Se podría pensar que este software es un buena introducción a conceptos de programación y a la programación. Sin embargo, este acercamiento cuenta con ciertas limitaciones. Primero, se tiene que la curva de aprendizaje del software es un poco alta. Esto se debe en parte a los nodos que posee. Por ejemplo, para hacer un ciclo dentro de Toy-Con Garage, la forma más simple de hacerlo es a través de un nodo de tipo condicional más un contador ya que no existen nodos que generen ciclos. Resulta que llegar a esta solución no es evidente para una persona que esta aprendiendo programación por primera vez. Es por esto que faltan tipos de nodos que faciliten el aprendizaje de conceptos como estos.

Otra limitación que tiene es el espacio para trabajar. Debido a que se ocupa el formato

de nodos, y todos estos se encuentran en una misma grilla, resulta en que el espacio se llene fácilmente y, a simple vista, no se entienda el flujo del código realizado. Una desventaja que tiene es el uso de colores. En general todo es blanco y negro con excepción de los puertos en los nodos. Esto causa que no haya referencias visuales que puedan facilitar el aprendizaje. En resumen, Toy-Con Garage, si bien es bastante poderoso y tiene una gran flexibilidad, resulta difícil e intimidante para una persona que esta recién aprendiendo a programar.

La solución que se desarrolló en este trabajo de título se distingue del Toy-Con Garage al tener un enfoque en la navegación completamente distinto. En Toy-Con Garage, se utiliza la pantalla táctil de la consola o el Joy-Con derecho como puntero, para moverse en la aplicación. En la solución se enfocó en el control de movimiento (de los Joy-Con) como el principal medio de navegación en la aplicación, de tal manera que permita una facilidad en el aprendizaje de conceptos computacionales. Esto también la diferencia de lenguajes por bloques como Scratch, donde se utiliza un mouse y teclado. La idea fundamental de este trabajo es el de explorar una forma distinta de las que ya existen, de introducir conceptos de pensamiento computacional a niñas y niños.

Capítulo 3

Diseño del Prototipo

Para poder desarrollar el entorno de desarrollo integrado (IDE) propuesto como solución al problema planteado en este trabajo de título, se necesita un prototipo de las interfaces de software que sirva como guía para su implementación. Una característica de este prototipo es que tiene que ser validado con usuarios o usuarios expertos en el área, con la finalidad de poder desarrollar un producto que tenga más posibilidades de ser considerado usable y útil por la población objetivo.

Para llegar a un prototipo de estas características se realizó lo siguiente:

1. Se desarrolló un primer prototipo con las interfaces de software del IDE.
2. Luego, se planificó una prueba de usabilidad para la validación del prototipo.
3. Se ejecutó esta evaluación con una muestra de usuarios expertos del dominio.
4. Se actualizó el prototipo, tomando en cuenta los resultados de la evaluación ejecutada.

En este capítulo, se van a describir en detalle los prototipos y los pasos que se llevaron a cabo para desarrollarlos.

3.1. Prototipo I

En esta sección se describen las vistas y características del primer prototipo desarrollado del IDE.

3.1.1. Vista inicial

La vista inicial que se muestra al usuario al lanzar el IDE es la que se presenta en la figura 3.1. Esta corresponde a un menú de bienvenida, en donde se encuentran los botones *Empezar* y *Salir*. El primer botón sirve para comenzar un nuevo proyecto y el segundo para cerrar el IDE.

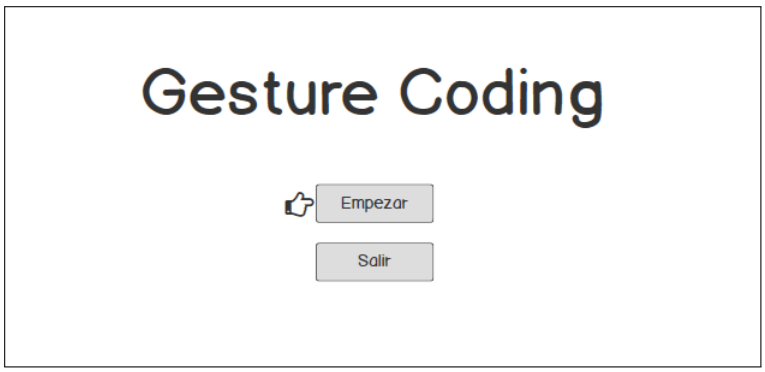


Figura 3.1: Vista inicial, prototipo I.

3.1.2. Vista *nuevo proyecto*

La vista de un nuevo proyecto se abre al seleccionar en el menú inicial la opción de *Empezar* y se puede ver en la figura 3.2. Esta es la vista principal del IDE, donde los usuarios podrán programar acciones con bloques para que las realice el *personaje*.

Esta vista se divide en dos zonas: la izquierda corresponde al *área de trabajo* y la derecha al *área del personaje*, tal como se muestra en la figura 3.3. Por un lado, el *área de trabajo* es donde el usuario va a ir creando y conectando los bloques de programación. Por otro lado, el *área del personaje*, tal como su nombre lo dice, es donde se encuentra el personaje.

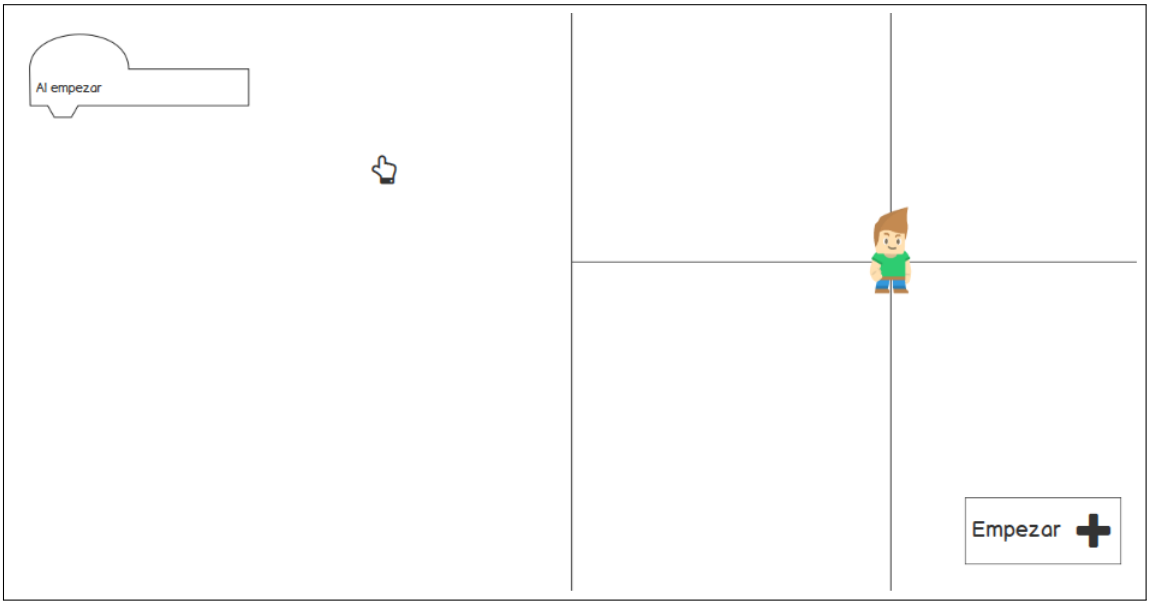


Figura 3.2: Vista nuevo proyecto, prototipo I.

El cursor

Para poder interactuar con los bloques y el personaje en esta vista se utiliza el *cursor*, que se puede ver en la figura 3.4. El cursor se puede mover a lo largo de toda la vista utilizando la palanca izquierda del Joy-Con izquierdo. Este movimiento es tal que si se mueve la palanca en una dirección, el cursor lo hace en esta misma dirección. También el cursor permite realizar lo siguiente:

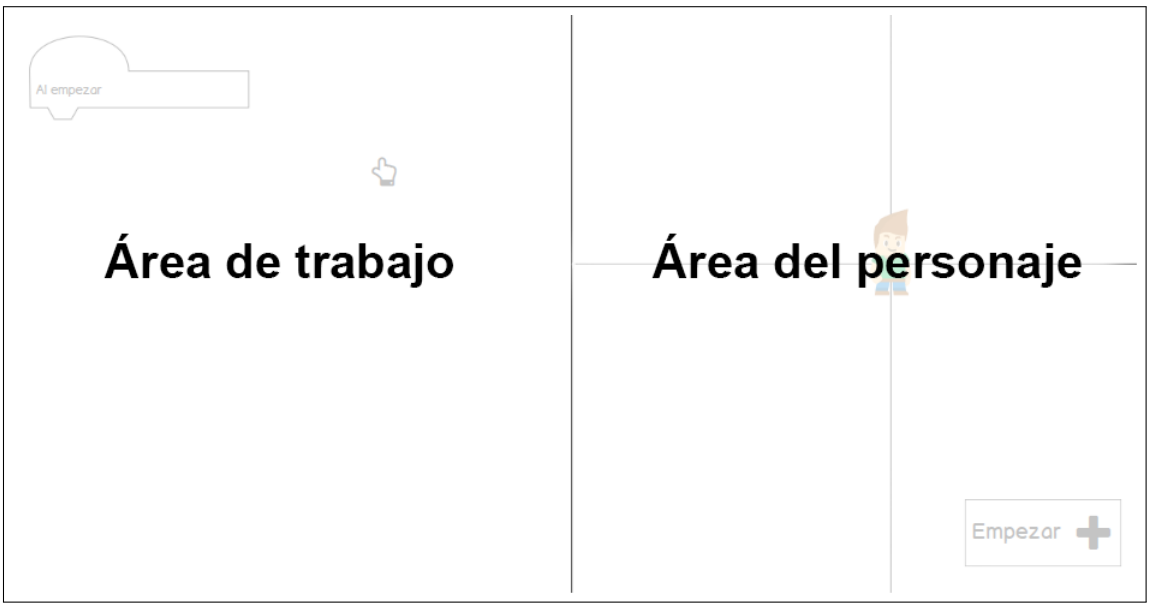


Figura 3.3: Áreas de la vista nuevo proyecto, prototipo I.

- Arrastrar bloques o al personaje de un lado a otro
- Borrar bloques
- Seleccionar inputs que contengan los mismos bloques
- Seleccionar botones en la pantalla.

Figura 3.4: El cursor, prototipo I.

El personaje

El *personaje* se puede ver en la figura[1] 3.5. Este personaje es quien va a recibir las instrucciones que fueron programadas con los bloques. Tal como se mencionó anteriormente, el cursor puede interactuar con el personaje al poder arrastrarlo de un lado a otro. Sin embargo, este se encuentra restringido a ser arrastrado solo en el área delimitada por el *área del personaje*.

El área del personaje además se representa como un sistema de coordenadas cartesianas de dos dimensiones, teniendo el centro del área como el origen. En efecto, tal como se muestra en la figura 3.2, el origen corresponde a la intersección entre los dos ejes.

Manipulación y conexión de bloques

Los *bloques* del IDE son los responsables de entregar las instrucciones para el *personaje*. Estos se pueden mover de un lugar a otro si se arrastran utilizando el cursor y su posicio-

[1]La imagen del personaje fue obtenida de `www.kenney.nl`. Los assets en esta página son distribuidos con la licencia *Creative Commons* y son *royalty free*.

Figura 3.5: El personaje, prototipo I

.

namiento está delimitado por el *área de trabajo*. También el cursor permite borrar cualquier tipo de bloque salvo el *bloque starter* que se define más adelante.

Para conectar los bloques entre sí, se encuentran distintos tipos de conexiones. Primero se tienen las conexiones entre puertos *hembra* y *macho*, que se pueden apreciar en la figura 3.6. En estas un puerto hembra solo se puede conectar a otro que sea macho y viceversa. Ejemplos de bloques con este tipo de puertos conectados entre sí, se pueden ver en la figura 3.7.

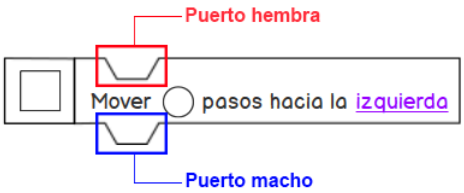


Figura 3.6: Bloque con puertos macho y hembra, prototipo I.

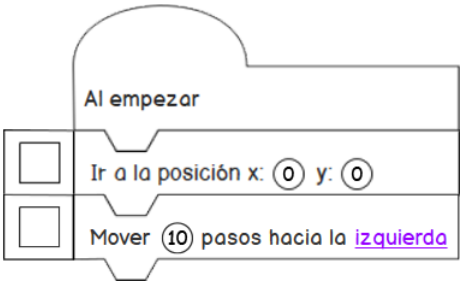


Figura 3.7: Tres bloques conectados por medio de conexiones de puerto macho-hembra, prototipo I.

Los bloques también pueden recibir *inputs*, donde los usuarios ingresan algún tipo de valor. Este valor, depende del *tipo de input* que sea. Primero se tienen los *inputs numéricos*, que tienen una forma circular. Aquí los usuarios pueden ingresar valores numéricos dentro del input. Además, este input, cuenta con un puerto en el que se pueden colocar *bloques circulares* (bloques que tienen una forma circular). Estos bloques tienen la característica de retornar valores numéricos. En la figura 3.8, se puede ver un bloque con un input numérico, un bloque circular y cómo quedan cuando son conectados.

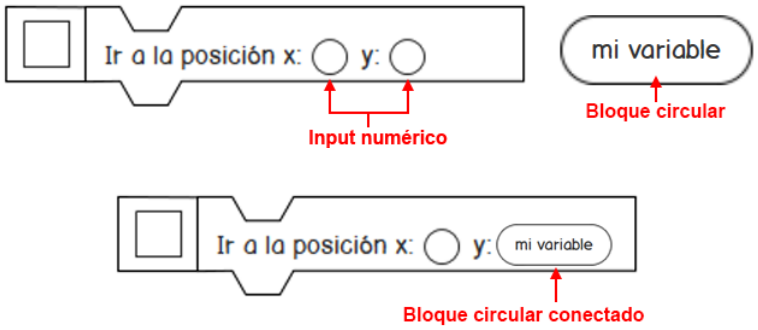


Figura 3.8: Bloque circular, input numérico y la conexión de ambos, prototipo I.

El otro tipo de input es el *input booleano*, que tiene una forma triangular. En este se pueden elegir los valores *verdadero* y *falso*. Para elegir entre estos dos valores, el input al seleccionarlo despliega una lista donde el usuario puede elegir alguna de estas dos opciones. Este input se comporta también como un puerto donde se pueden colocar *bloques triangulares* (bloques que tienen una forma triangular). Estos bloques van a retornar valores booleanos. En la figura 3.9 se puede ver un bloque con un input booleano y un bloque triangular.

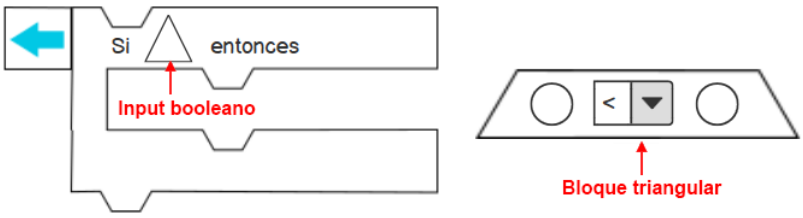


Figura 3.9: Input booleano y bloque triangular, prototipo I.

Para poder generar estas conexiones entre bloques se tiene que utilizar el *cursor*. Primero, se tiene que iniciar el arrastre de un bloque con el cursor, que se puede hacer cuando el cursor se encuentra en una posición cercana a la del bloque y se presiona el botón *A* del Joy-Con. Mientras que esté siendo arrastrado el bloque, este se va a mover la misma distancia en la que se mueve el cursor. Luego, el bloque con el cursor se mueven hacia algún puerto en el que se pueda conectar el bloque arrastrado. Por último, cuando el bloque arrastrado se encuentra cerca de la posición del puerto, se finaliza el arrastre al presionar de nuevo el botón *A* del Joy-Con, lo que causa que el bloque se suelte y genere la conexión con el puerto.

Tipos de Bloques

En el lenguaje del IDE existe una gran variedad de bloques que ocupar. Estos están divididos por distintos tipos, que son los siguientes:

1. Starter
2. Numérico
3. Booleano
4. Movimiento

5. Condicional
6. Ciclo

Cabe mencionar que la inspiración para la forma de los bloques del IDE viene del lenguaje de programación *Scratch*. La razón por la cual se utilizó Scratch como inspiración, es debido a que es un lenguaje ampliamente usado por niños[14]. Por lo tanto, Scratch además va a servir como un *benchmark* para comparar la usabilidad, utilidad percibida y rendimiento de este IDE.

Por otro lado, el uso de las metáforas de interacción dentro del IDE está estrechamente relacionado con los tipos de bloques. Antes de explicar las metáforas y su relación, se describen a continuación los bloques disponibles dentro de cada tipo.

Bloque starter

El *bloque starter* se puede ver en la figura 3.10. Este bloque es el que marca el inicio del programa, lo que significa que el código se corre a partir de él. Como se puede ver en la figura, este tiene un puerto macho. Solamente puede haber un bloque starter en un proyecto y no puede ser borrado.

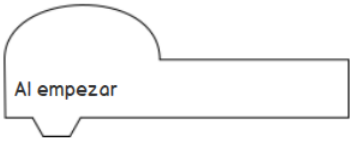


Figura 3.10: Bloque starter, prototipo I.

Bloques numéricos

Los bloques de tipo *número* se pueden ver en la figura 3.11. Como bien dice su nombre, este tipo va a agrupar a todos los bloques que tengan que ver con números, como puede ser la representación u operación con números.

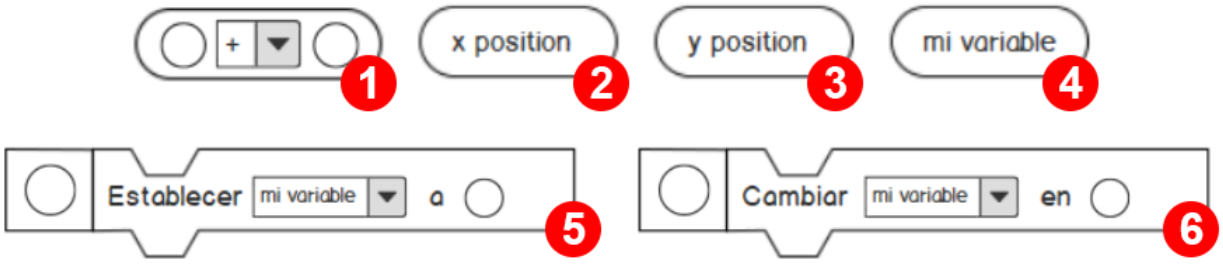


Figura 3.11: Bloques numéricos, prototipo I.

De este tipo de bloques se tiene el bloque *numeric operation* (figura 3.11-1), que es un bloque circular que contiene dos inputs numéricos. El valor retornado por este bloque es el de la operación entre dos números por medio de suma, resta, multiplicación o división. La elección del tipo de operador se realiza a través de una lista desplegable, que contiene los 4 operadores. Los inputs numéricos van a ser los dos operandos.

Dentro de los bloques de números también se tienen los bloques *X Position* e *Y Position* (figura 3.11-2 y 3). Estos son bloques circulares que retornan la posición en el eje x e y respectivamente, del personaje. La utilidad de estos bloques es que permiten que los usuarios puedan chequear las posición del personaje y dependiendo del valor obtenido realizar distintas acciones. El comportamiento anterior se puede obtener si estos bloques se utilizan en conjunto con los bloques booleanos y condicionales, que se describen más adelante en este mismo capítulo.

Los bloques 4, 5 y 6 de la figura 3.11 sirven para manejar las variables numéricas dentro del IDE. El detalle de cada uno de estos bloques se describe más adelante en esta misma sección.

Bloques booleanos

Los bloques de tipo booleano se pueden ver en la figura 3.12. Este tipo va a agrupar a todos los bloques que tengan que ver con booleanos, como puede ser la representación u operación con booleanos.

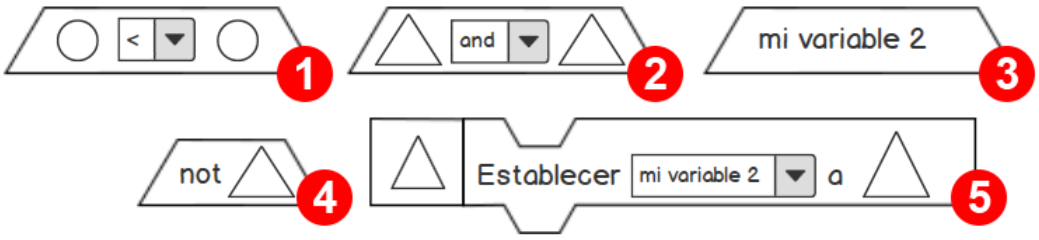


Figura 3.12: Bloques booleanos, prototipo I.

Primero se tiene el bloque *boolean operation between numbers* (figura 3.12-1). Este es un bloque triangular con dos inputs numéricos y puede retornar el valor de la operación mayor, menor o igual ($>$, $<$, $=$) entre valores numéricos. Los dos inputs son los operandos.

Luego se tiene el bloque *boolean operation between booleans* (figura 3.12-2). Este es un bloque triangular con dos inputs booleanos. Retorna el valor de la operación *and* o *or* entre valores booleanos. Los dos inputs son los operandos. Asimismo, se tiene el bloque *not* (figura 3.12-4), que es un bloque triangular con un input booleano. Retorna la negación del valor que se encuentra en el input.

Los bloques 3 y 5 de la figura 3.12 sirven para manejar las variables booleanas dentro del IDE. El detalle de estos dos bloques se describe más adelante en esta misma sección.

Bloques de movimiento

Los *bloques de movimiento* se pueden ver en la figura 3.13. Este tipo va a agrupar los bloques que puedan mover al personaje.

El primer bloque (de izquierda a derecha) de la figura 3.13 es el bloque *Move To* que permite colocar al personaje en un punto dado en el eje x e y. Este bloque posee un puerto

macho y uno hembra, y dos inputs numéricos. Estos inputs sirven para colocar el nuevo valor en el eje x e y del personaje.

El segundo bloque de la figura 3.13 es el bloque *Move By* que permite mover al personaje en una dirección y valor dados por el usuario. Este bloque posee un puerto macho y uno hembra, y un input numérico, este último para establecer el valor en que se va a mover. El personaje se puede mover en 4 direcciones: arriba, abajo izquierda y derecha.

Figura 3.13: Bloques de movimiento, prototipo I.

Bloques condicionales

Los bloques condicionales se pueden ver en la figura 3.14.

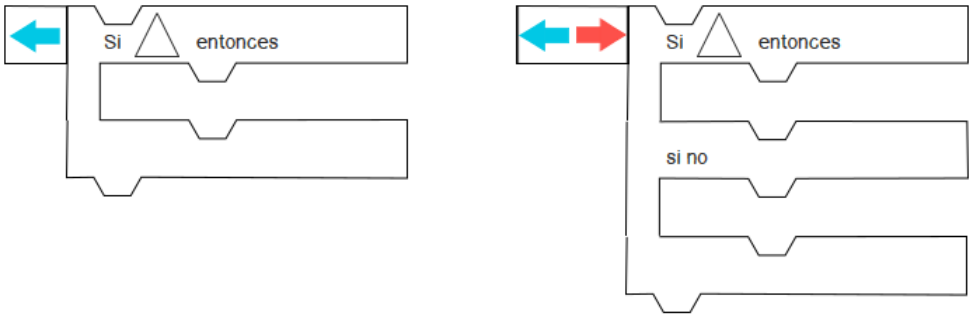


Figura 3.14: Bloques condicionales, prototipo I.

El primer bloque (de izquierda a derecha) de la figura 3.14 es el bloque *If* o *Condicional #1* que permite dado una condición, si se cumple, ejecutar el código dado, y en el caso contrario, no ejecutarlo. Este bloque tiene dos puertos macho y dos hembra. Un conjunto de un puerto macho con otro hembra sirve para conectar los bloques de la rama verdadera. También cuenta con un input booleano, que sería la condición.

El segundo bloque (de izquierda a derecha) de la figura 3.14 es el bloque *IfElse* o *Condicional #2* que permite, dado una condición, si se cumple, ejecutar el código dado, y en el caso contrario, ejecutar otro código. Este bloque tiene tres puertos macho y tres hembra. Un conjunto de un puerto macho con otro hembra sirve para conectar los bloques de la rama *verdadera*, y otro conjunto de puertos de las mismas características, para conectar los bloques de la rama en el caso *falso*. También cuenta con un input booleano, que sería la condición.

Bloques de Ciclo

Finalmente se tienen los bloques de ciclo, que se pueden ver en la figura 3.15. Este tipo va a agrupar los bloques que generen ciclos en el código.

El primer bloque (de izquierda a derecha) de la figura 3.15, permite generar una repetición de un código dado, una cierta cantidad de veces. El usuario puede ingresar la cantidad de

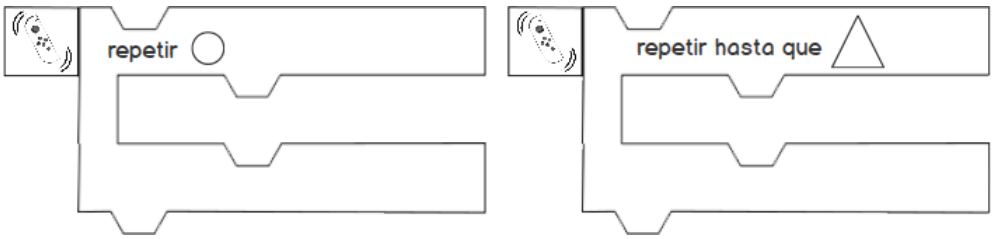


Figura 3.15: Bloques de ciclo, prototipo I.

veces en que se repite el código a través de un input numérico. Este bloque cuenta con 2 puertos macho y dos hembra. Un conjunto de un puerto macho con otro hembra sirve para conectar los bloques que van a ser parte del ciclo.

El segundo bloque (de izquierda a derecha) de la figura 3.15, permite generar una repetición de un código mientras se cumpla una condición dada por el usuario. La condición es un input booleano. Este bloque cuenta con dos puertos macho y dos hembra. Un conjunto de un puerto macho con otro hembra sirve para conectar los bloques que van a ser parte del ciclo.

Variables del lenguaje

En el lenguaje del IDE también existen *variables*. Estas son de dos tipos, las que retornan un valor numérico (*variables numéricas*) o las que retornan un valor booleano (*variables booleanas*). Se utilizan solamente estos dos tipos de datos en esta primera versión del IDE para que exista una mayor simplicidad en la construcción del lenguaje y para facilitar la adopción del lenguaje por parte de los usuarios.

Para poder manipular las variables numéricas, se utilizan los bloques 4, 5 y 6 de la figura 3.11. El primero es un bloque circular que retorna el valor que contiene la variable numérica (en este caso, el valor de la variable *mi variable*). El segundo, permite asignar un nuevo valor a la variable numérica elegida por el usuario. La variable se elige a través de la lista desplegable que posee el bloque. El último bloque permite incrementar una variable numérica en un determinado valor. El valor es ingresado por el usuario a través de un *input numérico*.

Por el otro lado, para manipular las variables booleanas, se van a utilizar los bloques 3 y 5 de la figura 3.12. El primero es un bloque triangular que retorna el valor que contiene la variable booleana (en este caso, el valor de la variable *mi variable 2*). El segundo, permite asignar un nuevo valor a la variable booleana elegida por el usuario. La variable se elige a través de la lista desplegable que posee el bloque y su nuevo valor, a través de un input booleano.

Para crear las variables, se utiliza un modal (figura 3.16) donde se establece el nombre de la variable, y el valor inicial que va a tener. Un modal es una ventana que aparece sobre la ventana principal de una aplicación. Cuando aparecen, desactivan la interacción con la ventana principal, y solo se permite interactuar con el modal. Luego, cuando se cierra el modal, se puede volver a interactuar con la ventana principal

Todos los bloques mencionados hasta ahora, se diseñaron para esta versión del IDE debido a que estos definen los elementos mínimos que permiten la enseñanza de conceptos impor-

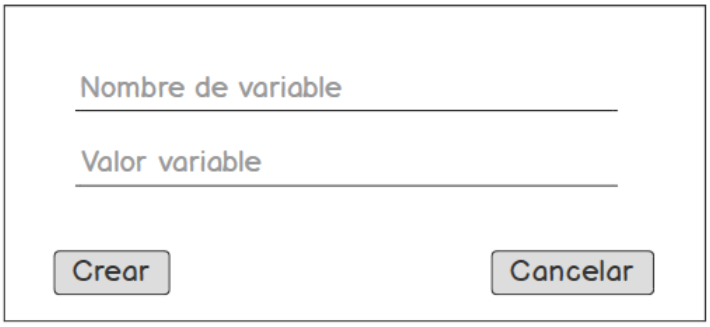


Figura 3.16: Modal para creación de variables, prototipo I.

tantes y básicos de la programación, tales como la idea de secuencialidad, lógica condicional e iteración[15].

Correr el código

Después de haber conectado los bloques deseados, el usuario puede correr el código. Para hacer esto, puede simplemente presionar el botón + del Joy-Con derecho o presionando con el cursor el botón *Empezar* que se encuentra en la vista. Cabe destacar que el orden de ejecución de los bloques es desde arriba hacia abajo, empezando por el bloque *starter*. Esto se puede ver gráficamente en la figura 3.17, donde las flechas rojas representan el flujo de la ejecución del código.

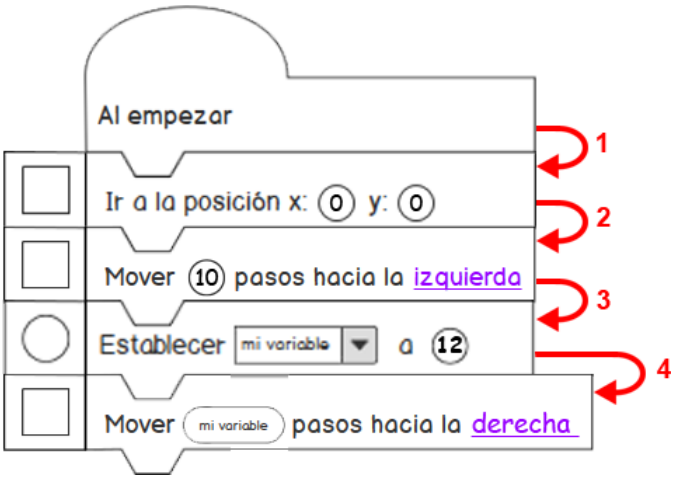


Figura 3.17: Orden de ejecución de los bloques, prototipo I.

Creación de bloques y gestos

Lo que se ha descrito hasta ahora es bastante similar a cómo funcionan típicamente los lenguajes de programación por bloques, tales como *Scratch* o *Snap!*[2], solamente cambiando las entradas de mouse y teclado, por las de los controles Joy-Con. Sin embargo, la creación de bloques es algo totalmente distinto en este IDE y se ocupan especialmente los sensores de

[2]Página oficial de Snap!: `https://snap.berkeley.edu/`

movimiento de los controles, basándose en un paradigma de diseño de interfaces de interacción natural.

Para la creación de bloques, se utilizan metáforas de interacción que se llamarán *gestos* en este trabajo de título. Un gesto corresponde a un movimiento determinado que se realiza con los Joy-Con. Por ejemplo, un gesto sería realizar un movimiento circular con uno de los Joy-Con (figura 3.18).

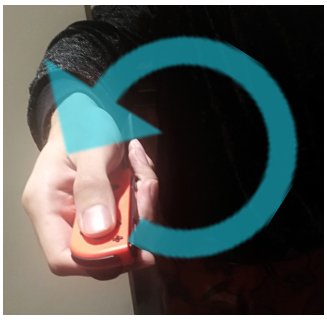


Figura 3.18: Movimiento circular con el Joy-Con derecho.

Luego, en el lenguaje que se presenta en este trabajo de título, un gesto específico mapea a la creación de bloques de programación. Cada gesto tiene asociado 1 o más bloques, por lo que, si el usuario ejecuta un gesto con los controles correctamente, crea el bloque asociado o puede seleccionar entre uno de los bloques asociados, y así crearlo.

Los pasos que se deben realizar para detectar los gesto en el IDE son los siguientes:

1. El usuario presiona el botón gatillo del Joy-Con (el botón ZL en el Joy-Con izquierdo, ZR en el derecho).
2. Manteniendo presionado el gatillo, el usuario realiza el gesto deseado con el Joy-Con (el mismo Joy-Con con el que se está presionando el gatillo).
3. El usuario suelta el gatillo.

Si el gesto detectado tiene solamente un bloque asociado, se crea el bloque inmediatamente. Si el gesto detectado tiene más de un bloque asociado, se abre un menú donde el usuario elige cual bloque quiere crear. Finalmente, si no se detecta ningún gesto, no se crea nada y no se despliega ningún tipo de retroalimentación al usuario.

En la figura 3.19 se encuentran los gestos disponibles en el IDE diseñado en este trabajo de título. Cada uno posee una forma distinta, siendo ese el movimiento que debe realizar el usuario. Es posible notar que hay distintos tipos de gestos a partir de los colores con los que se identifican. En efecto, si un gesto es de color *azul*, solamente se puede realizar con el Joy-Con *izquierdo* (que físicamente es de ese color, de acuerdo a la semántica definida por la consola Nintendo Switch). De igual manera, si el gesto es de color *rojo*, entonces solamente se puede realizar con el Joy-Con *derecho*. Finalmente, si es de color *negro* se puede realizar con cualquiera de los dos mandos.

También hay restricciones en los gestos dependiendo si tienen una *flecha*. Esto restringe la dirección con la que se realiza un gesto. Por ejemplo, en la figura 3.19, el gesto número 3

es una flecha que tiene una dirección hacia la izquierda, por lo que, al realizar el gesto, se va a detectar solamente si se realiza el movimiento del control hacia la izquierda. También, como en este caso la flecha es azul, se realiza el gesto con el mando izquierdo.

Un tipo de gestos son los que se realizan con los dos controles al mismo tiempo. El gesto número 5 de la figura 3.19 es un ejemplo de esto. En este gesto hay que mover el mando izquierdo hacia la izquierda y el derecho, hacia la derecha, ambos al mismo tiempo, para realizar correctamente el gesto.

Por último, cabe mencionar que el gesto número 6 de la figura 3.19 es el agitar el Joy-Con. Debido a que es de color negro, se puede realizar con cualquiera de los dos mandos.

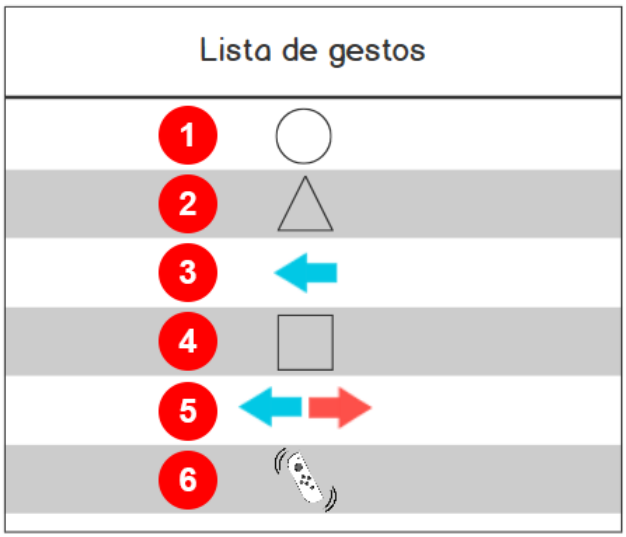


Figura 3.19: Lista de gestos, prototipo I.

En la figura 3.20 se muestra una tabla con los gestos y su tipo de bloque asociado. Esto significa entonces, que al realizar uno de estos gestos, se puede elegir entre alguno de los bloques de este tipo asociado a través de un menú. Los menús de los bloques se pueden ver en las figuras 3.22, 3.23, 3.24 y 3.25

Por otro lado, en la figura 3.21 se tienen los demás gestos con el bloque asociado que tienen. Como se puede ver, a la izquierda de los bloques en la figura, tienen una pequeña etiqueta con un gesto. Este mismo gesto es el que se realiza para crear ese bloque y sirve de recordatorio para el usuario.

Diseño de los gestos

El diseño que hay detrás de cada gesto depende del tipo de bloque con el que está asociado. Primero que todo, en el caso de los bloques numéricos, su gesto asociado es una moción circular (figura 3.20), debido a que estos (con excepción de los que son para manejar variables) tienen una forma circular y el input numérico, que es donde se pueden conectar, también posee esta misma forma. Esto tiene como consecuencia que sea fácil para un usuario saber que necesita realizar este gesto para encontrar los bloques numéricos y colocarlos en el input, ya que todos comparten la misma forma.

Figura 3.20: Lista de gestos con su tipo de bloque asociado, prototipo I.

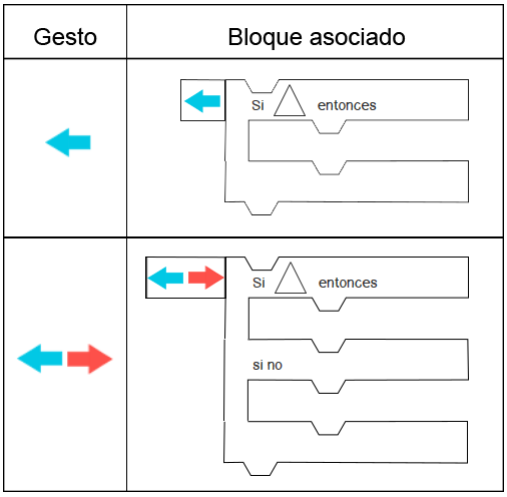


Figura 3.21: Lista de gestos con su bloque asociado, prototipo I.

En el caso de los bloques booleanos, el mecanismo es análogo, donde su gesto asociado es una moción triangular (figura 3.20). Estos bloques (con excepción de los que son para manejar variables) tienen una forma triangular y el input booleano, que es donde se pueden conectar, también posee la misma forma.

Por otro lado, los bloques condicionales se dividen en dos: un gesto para crear el bloque *If* y otro para el *If Else* (figura 3.21). El gesto del bloque *If*, es mover hacia la izquierda el Joy-Con izquierdo; el del *If Else*, es el de realizar al mismo tiempo el movimiento anterior y el de mover el Joy-Con derecho hacia la derecha. La razón detrás del diseño de estos gestos es que el movimiento de cada Joy-Con representa una rama del bloque: el movimiento del Joy-Con izquierdo, representa a la creación de la rama verdadera, y el del derecho, de la rama falsa. Por lo tanto, si se quiere crear solamente una rama verdadera, se crea un bloque *If*, y si se quieren crear ambas ramas, el *If Else*.

También se tienen los bloques de ciclos, que su gesto asociado es agitar alguno de los dos Joy-Con (figura 3.20). En este caso, la razón del diseño del gesto, es que agitar algo es un movimiento repetitivo, por lo cual, se puede asociar el concepto de repetición a este gesto. Debido a que los bloques de ciclos sirven para repetir código, tiene sentido asociar este

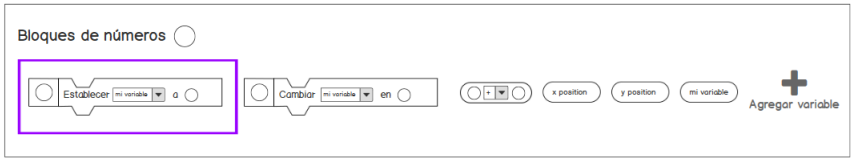


Figura 3.22: Menú bloques numéricos, prototipo I.

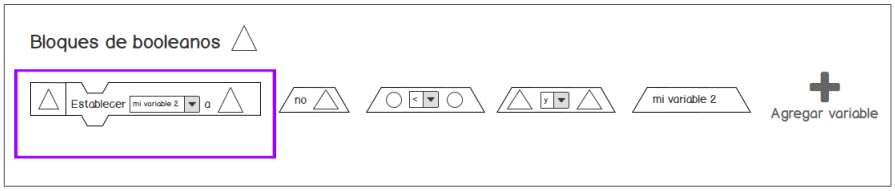


Figura 3.23: Menú bloques booleanos, prototipo I.

movimiento.

Finalmente, el único tipo de bloque en la que no hay una metáfora concreta detrás de su diseño de gesto es en el de movimiento. En consecuencia, se le asoció un gesto que fuera fácil de realizar y distinto a los demás.

Dado lo anterior, la forma en que se diseñó la creación de gestos fue para que los usuarios puedan asociar los gestos a un concepto. Esta es la contribución principal de este trabajo de título y es la que lo diferencia de otros lenguajes ya existentes en la literatura.

El uso de gestos también se va a ocupar en el bloque *Move By* para asignar la dirección en la cual se va a mover el personaje. Para realizar esto, hay que seleccionar con el cursor el campo que contiene su dirección el cual se encuentra resaltado dentro del bloque. Después de esto se muestra un mensaje en pantalla (figura 3.26) y se realiza un gesto con alguno de los Joy-Con con la dirección deseada para el personaje. Las direcciones disponibles son solamente arriba, abajo, izquierda y derecha.

Tutorial

La primera vez que se lance la vista de *nuevo proyecto*, se va a mostrar en pantalla un tutorial. Este módulo tiene como propósito el enseñarle al usuario un poco acerca del lenguaje y de cómo puede crear bloques. Esto resulta necesario debido a lo distinto que es crear bloques en este lenguaje en comparación a otros más tradicionales. En el Apéndice A se encuentra el tutorial de esta versión del prototipo.

Modo pantalla completa

Esta vista también permite mostrar en pantalla completa el *área de trabajo* o la del *personaje*.

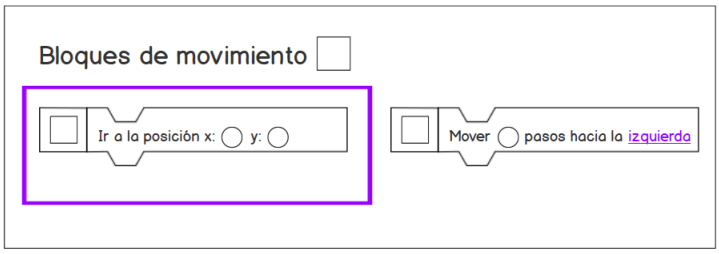


Figura 3.24: Menú bloques de movimiento, prototipo I.

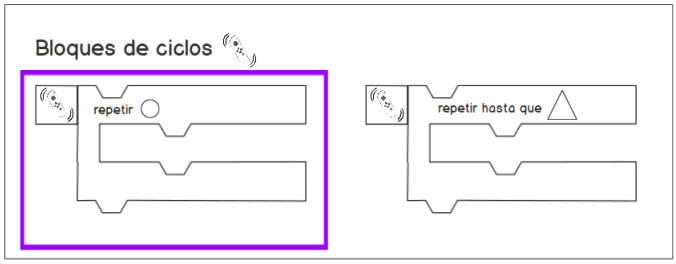


Figura 3.25: Menú bloques de ciclos, prototipo I.

Menú de opciones

Este es un pequeño menú que cuenta con tres botones: *Lista de gestos*, *Volver al menú principal* y *Salir*. Al presionar el primero, se accede a una ventana con la lista de gestos, el segundo, vuelve a la vista principal y el tercero, sale de este menú. En la figura 3.27 se tiene una imagen del menú.

3.1.3. Controles

Los controles que se ocupan para navegar dentro del IDE son los siguientes. Primero para mover el *cursor*, se utiliza la palanca del Joy-Con izquierdo. Para seleccionar botones en pantalla y arrastrar bloques o al personaje se utiliza el botón A de los Joy-Con. Para borrar bloques, se coloca el cursor por encima del bloque a borrar, y se presiona el botón X.

En el *área de trabajo* se puede hacer *panning* con la palanca derecha. El *panning* es el método de desplazar el área visible de una imagen, en este caso, la imagen es el el área de trabajo. Para entrar al modo pantalla completa del *área de trabajo*, se presiona la palanca izquierda. Por otro lado, para entrar a este mismo modo pero en el *área del personaje*, se presiona la palanca derecha. Finalmente, para salir del modo pantalla completa y volver a la vista normal, se presiona la misma palanca con la cual se entro a ese modo.

Para navegar entre botones en pantalla, se utiliza también la palanca izquierda, y para seleccionar una opción, se utiliza el botón A de los Joy-Con. Asimismo, para poder escribir en los campos de input numérico o en el modal de creación de variables, pasa lo siguiente. Al seleccionar algún campo con el botón A, se muestra en pantalla un teclado virtual que es controlado por el mismo mando. En la figura 3.28 se tienen los dos tipos de teclados que pueden aparecer, uno numérico y el otro con letras. Finalmente, para entrar al menú de opciones, se utiliza el botón $-$.

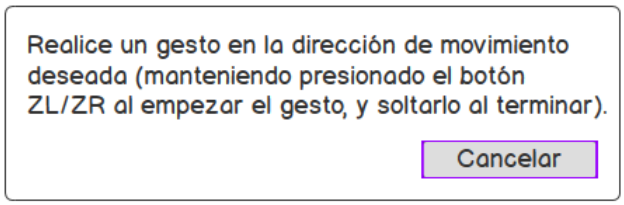


Figura 3.26: Mensaje con las instrucciones para asignar la dirección del bloque *Move By*, prototipo I.

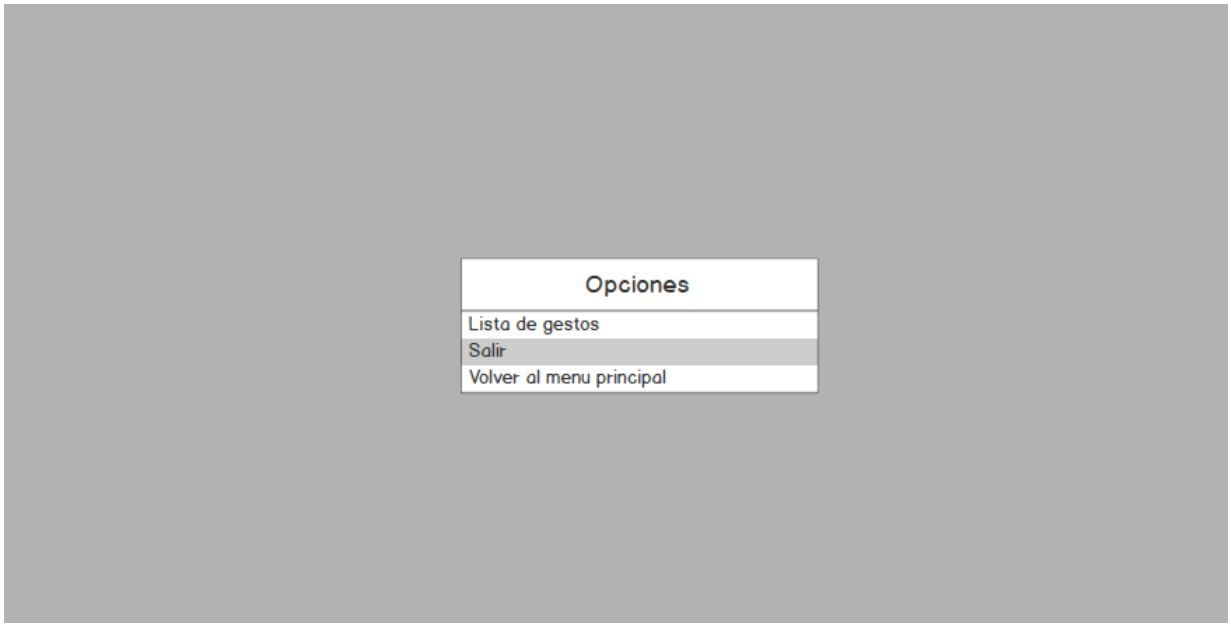


Figura 3.27: Menú de opciones, prototipo I.

3.2. Validación del prototipo I

Con un primer prototipo ya diseñado, se procedió a validar su aplicabilidad y aceptabilidad con un panel de usuarios expertos. Para ello, se siguió un protocolo sistemático basado en tareas.

En este protocolo de evaluación, se realizó una entrevista entre una persona que guía el test (entrevistador) y el sujeto de prueba (entrevistado). Estas se realizaron de forma remota, a través de una videollamada mediada por la plataforma *Zoom*. Durante el test, el entrevistado tenía que realizar una serie de tareas con el prototipo y después de haberlas terminado, entregar una retroalimentación acerca de su experiencia con el prototipo. Para que el sujeto de prueba pueda realizar las tareas con el prototipo, se utilizó la técnica llamada *Mago de Oz*[10], la cual es ampliamente utilizada para validar prototipos en etapas tempranas del desarrollo de una aplicación interactiva.

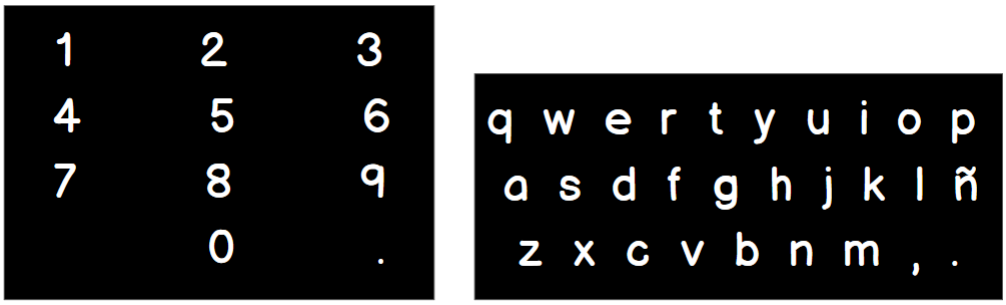


Figura 3.28: Teclado numérico (izquierda) y alfabético (derecha), prototipo I.

3.2.1. Participantes

Para la ejecución de la validación, se reclutó a 5 usuarios expertos en el dominio de aplicación (es decir, enseñar programación a niños). Para ello, se contactó a monitores del Taller de Desarrollo de Pensamiento Computacional del Departamento de Ciencias de la Computación de la Universidad de Chile, coordinado por el Profesor Guía de este trabajo de título. Entre los 5 participantes, 2 fueron mujeres y 3 hombres. Si bien algunos de ellos poseían un mando Joy-Con, no todos estaban familiarizados con esta tecnología.

3.2.2. Procedimiento de evaluación

Siguiendo el protocolo Mago de Oz, se le pidió a los participantes que interactuaran con un sistema que creen es autónomo, pero en realidad, está siendo operado (total o parcialmente) por otra persona. En este caso, el prototipo es totalmente controlado por el entrevistador. Esto tuvo que adaptarse a un formato en línea, donde el entrevistador compartía la pantalla con el prototipo y el usuario entregaba las instrucciones de lo que quería hacer. Para complementar los resultados del proceso, se aplicó igualmente la técnica *Thinking Aloud*, que consiste en que el usuario evaluado verbaliza sus decisiones, apreciaciones y acciones mientras interactúa con el sistema[12].

En términos operativos, se pidió a los participantes su consentimiento explícito para grabar la sesión con la finalidad de poder realizar un posterior análisis. La razón por la que se optó por un formato en línea fue debido principalmente a los efectos de la pandemia del COVID-19 que se vivió durante el desarrollo de este trabajo de título. Las medidas como el distanciamiento social y las cuarentenas limitaron las formas en que se podían realizar las pruebas, obligando entonces a adoptar este formato.

Las tareas que tuvieron que llevar a cabo los usuarios durante la evaluación fueron:

1. Realizar el tutorial
2. Generar un bloque asociado a cada gesto
3. Realizar lo siguiente:
   (a) Hacer el gesto de los bloques de booleanos, y elegir el bloque de “mi variable”.
   (b) Colocar el bloque de “mi variable 2” en el campo del bloque de condicional if
4. Realizar lo siguiente:

(a) Hacer el gesto de los bloques de números, y elegir el bloque de "mi variable".
(b) Colocar el bloque de "mi variable" en el campo del bloque de movimiento elegido
5. Borrar un bloque
6. Mover un bloque
7. Correr el código
8. Pasar a pantalla completa del código y del personaje.
9. Mover al personaje (sin tener que correr el código de los bloques, de forma manual)
10. Crear una variable

3.2.3. Resultados obtenidos

La retroalimentación fue obtenida a partir de observaciones que se hicieron respecto a la entrevista junto a los comentarios de los participantes. Además se analizaron las grabaciones de las videollamadas, para obtener aún más información.

Lo primero que hay que destacar es que todos los participantes lograron realizar todas las tareas de la actividad sin que requirieran una mayor intervención. También, las impresiones que hubo acerca del uso de gestos dentro del IDE fueron bastante positivas, donde los usuarios valoraron que estos eran bastante simples de realizar y distintos entre ellos. Incluso algunos participantes notaron la relación en las formas de algunos gestos y sus bloques asociados, lo que valoraron positivamente.

Una pequeña observación dentro de las entrevistas era la forma en que los participantes realizaban los gestos. La mayoría de ellos eran diestros, con una persona siendo ambidiestro. Como se mencionó anteriormente, los gestos que son de color negro se pueden realizar con cualquiera de los dos mandos, por lo que estos gestos los diestros siempre los hacían con el Joy-Con derecho. En el caso del participante ambidiestro, no mostró preferencia en cual mando ocupar para realizarlos.

Sin embargo, se observaron dificultades en algunas tareas y críticas respecto al diseño del prototipo por parte de los participantes. Primero que todo, la forma en que se borraban los bloques resultaba poco intuitiva para los usuarios. Todos trataban de seleccionar el bloque y luego presionar el botón de borrado, cuando lo que tenían que hacer era colocar el cursor encima del bloque y luego presionar el botón.

También, se detectó un problema con los usuarios cuando tuvieron que realizar el gesto con forma de cuadrado (el gesto asociado a los bloques de movimiento). Durante el tutorial, se pedía a los usuarios realizar este mismo gesto, y muchos de ellos quedaron un poco confundidos en un principio con que movimiento tenían que realizar con los controles. Luego, los usuarios comunicaron que la causante de esto fue la simbología del gesto.

Otro comentario recibido fue respecto a la lista de gestos dentro de la interfaz. Resultaba incómodo para los usuarios que no estuviera visible cuando tenían que realizar los gestos y la única forma de acceder a ella era dentro del tutorial y el menú de opciones.

Por último, una de las mayores críticas que hubo por parte de los usuarios fue que no había ninguna retroalimentación visual cuando realizaban los gestos. Incluso, cuando se equivoca-

ban al realizar un gesto que no era detectado por el IDE, no tenían retroalimentación. Esto resultó ser bastante negativo, debido a que sentían que era importante saber en todo momento como están haciendo un gesto, y si se llegan a equivocar, puedan saber el por qué.

En conclusión, a partir de las observaciones y comentarios de los participantes, se puede interpretar que en el prototipo actual los gestos elegidos resultaron ser usables por los participantes. Sin embargo, hay algunos problemas con el IDE, como la retroalimentación visual que falta al realizar los gestos, que deberían ser corregidos en un nuevo prototipo.

3.3. Prototipo II

A partir del análisis de los resultados obtenidos de la validación del primer prototipo, se procedió a efectuar ajustes de diseño con miras a su posterior implementación. Así pues, el *prototipo II* corresponde a una versión mejorada del prototipo original, en la que se tomó en cuenta de manera explícita la retroalimentación recibida por los usuarios expertos que evaluaron el IDE original. En lo que sigue de esta sección, se mencionan los cambios más significativos del Prototipo II respecto a la versión original.

El primer cambio que se realizó fue el de la simbología de los gestos. Debido a la confusión que hubo durante las entrevistas en la ejecución de uno de los gestos, estos se actualizaron y se pueden ver en la figura 3.29. La mayor diferencia es que ahora todas cuentan con una flecha para representarlos, con un punto que marca donde se debe iniciar el gesto. Cabe aclarar que en los gestos de círculo, triángulo y cuadrado no importa la dirección en que se haga el gesto.

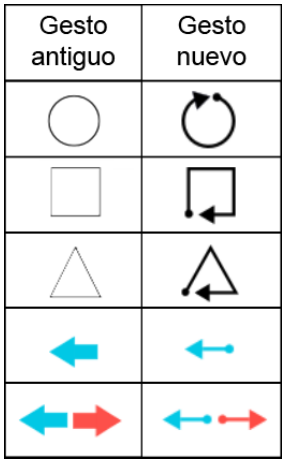


Figura 3.29: Simbología de gestos actualizados, prototipo II.

Otro cambio efectuado al diseño del prototipo se encuentra en la vista de un nuevo proyecto, como se puede apreciar en la figura 3.30. Primero, en la zona superior izquierda se tiene una lista con los gestos y el nombre del tipo de bloque asociado. Esto va a servir para recordar al usuario los gestos disponibles y que no tenga que buscarlos al menú de opciones. Segundo, se tienen dos botones en la vista, uno para la pantalla completa del área de trabajo y otro para la pantalla completa del área del personaje. Finalmente, en la zona inferior, se tiene una barra, llamada *barra de estado*, con los botones disponibles que puede utilizar el usuario en ese momento, así como su funcionalidad. Esta barra va a cambiar sus valores dependiendo del estado del proyecto en ese momento. Por ejemplo, en la figura 3.31, el usuario

está arrastrando un bloque con el cursor y ahora en la barra aparece la opción de soltar y borrar. Que la opción de borrar un bloque se encuentre solo cuando uno lo arrastra, es otro cambio con respecto al prototipo anterior.

Figura 3.30: Vista de un nuevo proyecto, prototipo II.

Figura 3.31: Vista de un nuevo proyecto con el cursor arrastrando un bloque, prototipo II.

Para borrar bloques en esta versión se tienen dos opciones. La primera consiste en que, para borrarlo, hay que primero arrastrar el bloque y luego presionar el botón de borrado. La segunda opción es que, al arrastrar el bloque, aparece un ícono de un basurero en pantalla, tal como se puede ver en la figura 3.31. Si el bloque arrastrado, se suelta donde se encuentran el basurero este es borrado.

Otro cambio efectuado corresponde a la creación de una nueva vista para crear gestos.

En este modo, que se puede acceder desde la vista nuevo proyecto, el usuario puede realizar los gestos para crear los bloques o asignar la dirección a los bloques *Move By*. Esto antes se hacía en la vista de nuevo proyecto, pero se cambió para agregar retroalimentación visual al usuario.

Para acceder esta vista para crear bloques, hay que estar en la vista de nuevo proyecto y se tienen que presionar cualquiera de los dos gatillos (los botones ZL y ZR). En el caso en que se este asignando una dirección al bloque *Move By*, se accede cuando se presiona su campo de dirección.

Inicialmente, al abrir la vista se vería la figura 3.32. Aquí se puede notar que hay dos punteros, uno rojo y uno azul, los cuales son controlados por el Joy-Con derecho e izquierdo respectivamente. Estos son controlados con los sensores de movimiento de cada control. Así, si se mueve el mando derecho hacia arriba, también lo hace el puntero rojo. Los punteros van a servir de guía para el usuario cuando realicé los gestos para crear bloques.

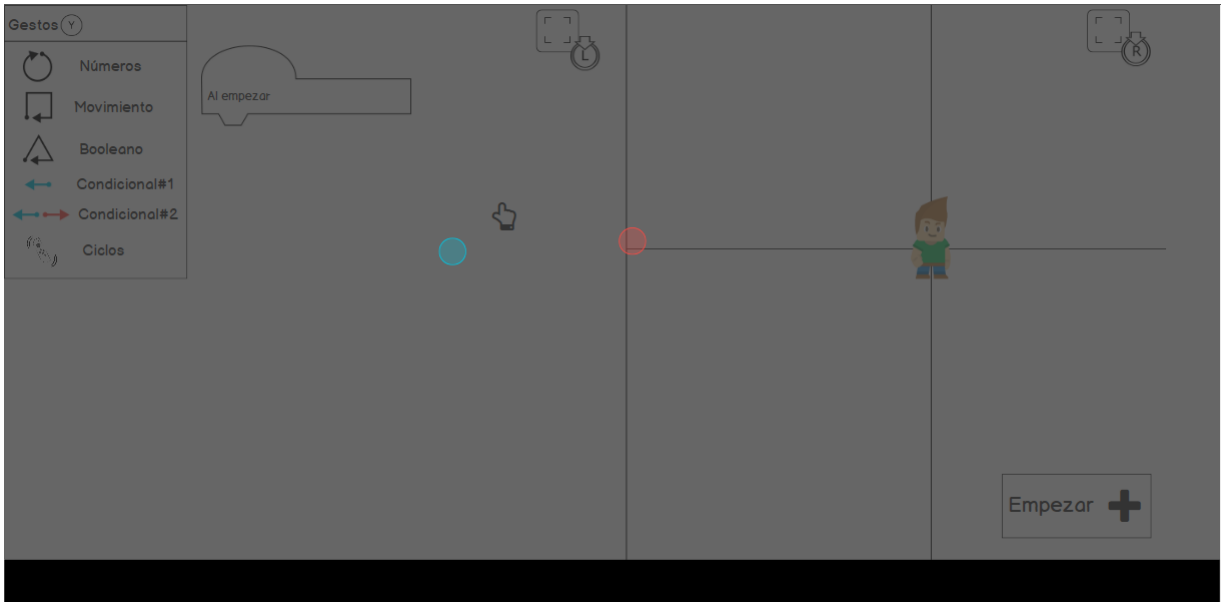


Figura 3.32: Vista creación de gestos, prototipo II.

Para que un usuario comience a dibujar un gesto con alguno de los mandos tiene que presionar el gatillo del mando respectivo. Luego debe mantener presionado el botón hasta que termine de realizar su gesto. El puntero del respectivo mando, mientras se esté presionando el gatillo, va a ir dejando un rastro detrás de él, como se ve en la figura 3.33. Al terminar de dibujar el gesto, si es detectado, se realiza la acción asociada a ese gesto y en el caso contrario, se muestra un mensaje en pantalla de "gesto no detectado". Finalmente, para salir de esta vista y volver a la vista de nuevo proyecto, se presiona el botón *B*.

3.3.1. Ajustes adicionales al diseño

Al experimentar con la implementación del IDE a modo exploratorio, se encontraron distintos aspectos a mejorar del diseño del prototipo II. Estas mejoras se tradujeron en ajustes que se hicieron al diseño, los que se detallan a continuación.

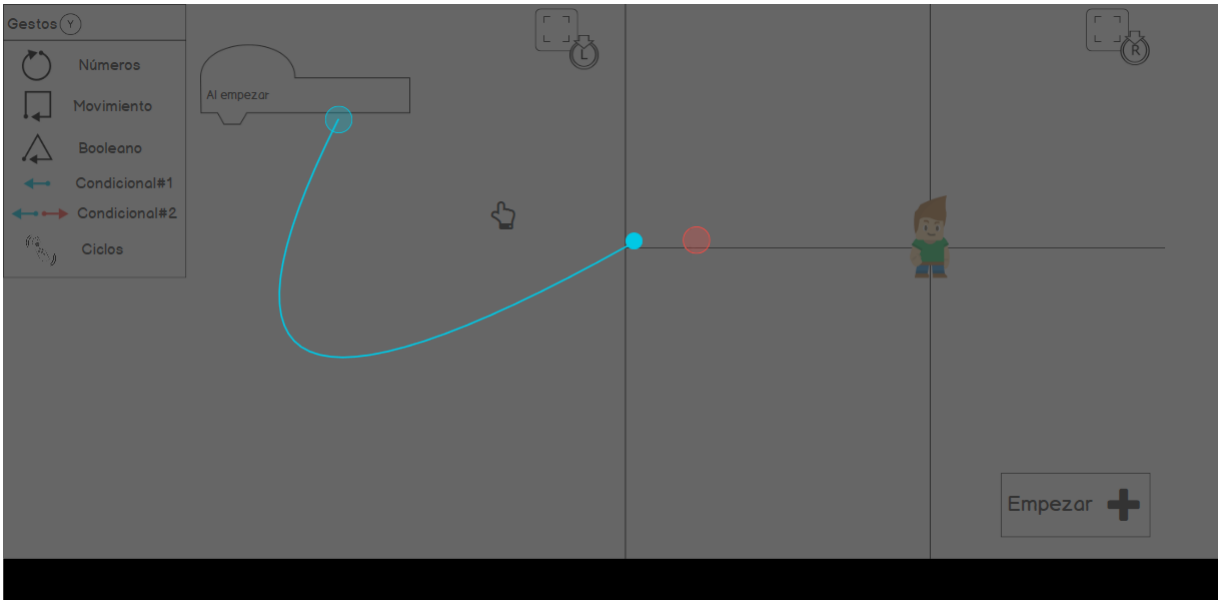


Figura 3.33: Trazo que deja atrás el puntero izquierdo.

La primera modificación que se hizo fue a la lista de gestos que se encontraba en la vista de nuevo proyecto. Esta se movió desde esta vista a la de creación de gestos. Este cambio se hizo para que la lista, que sirve como guía para recordar los gestos y qué es lo que hacen, solamente se encuentre cuando se necesite, que es cuando se crean los gestos. También, dentro del diseño del prototipo II, había solamente una lista con los gestos para la creación de bloques pero faltaba una con los gestos que entregan la dirección al bloque *Move By*. Por lo tanto, se agregó esta última que se puede ver en la figura 3.34. Ahora que hay dos listas, solamente una de ellas se va a mostrar en la vista de creación de gestos, dependiendo de si se está creando un bloque o si se está asignando la dirección a algún bloque *Move By*.

Luego de realizar pruebas preliminares sobre la implementación de los gestos, se encontró que al detector de gestos implementado le costó detectar el gesto asociado a los bloques de movimiento. Este gesto tiene forma cuadrada y el detector lo confundía con el gesto de los bloques numéricos, que tiene forma circular. Por lo tanto, para solucionar este problema, se reemplazó el gesto originalmente asociado al bloque de movimiento por un gesto que tiene forma de **L**. La lista de gestos para la creación de bloques actualizada se puede ver en la figura 3.35.

Figura 3.34: Lista de gestos para asignar la dirección al bloque *Move By*.

Otra modificación que se hizo fue a los bloques de operaciones, que son: *numeric operation*, *boolean operation between numbers* y *boolean operation between booleans*. Estos, en lugar de tener listas desplegables para elegir el tipo de operador, se dividieron cada una de las opciones en nuevos bloques. La razón de este cambio fue para que ahora el usuario pueda encontrar la

Figura 3.35: Lista de gestos para la creación de bloques actualizada.

información de los bloques cuando los crea de forma más rápida, ya que no tiene que navegar entre las opciones de los operadores. El cambio se puede ver en la figura 3.36. Los nombres de estos nuevos bloques y la operación que realizan, se encuentran en la siguiente lista:

1. Del bloque *numeric operation*:
   (a) Bloque *Sum*: operación de suma.
   (b) Bloque *Substract*: operación de resta.
   (c) Bloque *Multiply*: operación de multiplicación.
   (d) Bloque *Divide*: operación de división.
2. Del bloque *boolean operation between numbers*:
   (a) Bloque *Greater Than*: operación mayor que.
   (b) Bloque *Lesser Than*: operación menor que.
   (c) Bloque *Equals To*: operación de igualdad.
3. Del bloque *boolean operation between booleans*:
   (a) Bloque *And*: operación AND.
   (b) Bloque *Or*: operación Or.

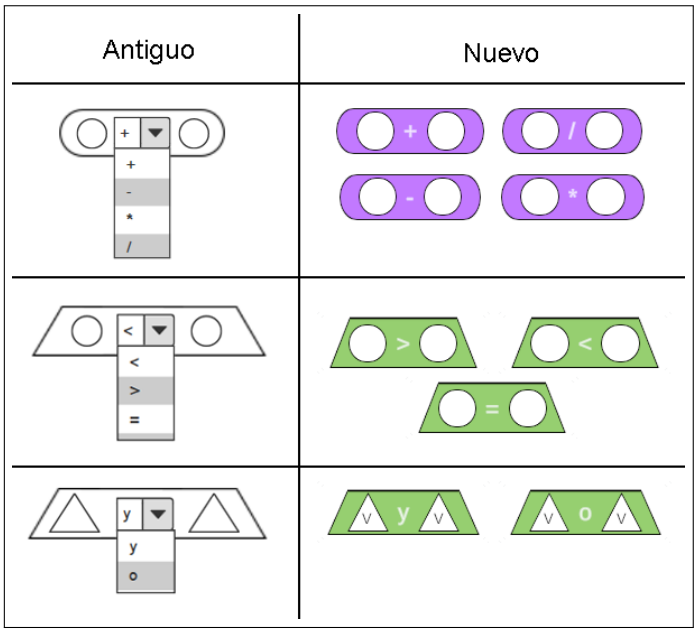


Figura 3.36: Modificación a los bloques de operaciones.

Por otro lado, se agregó en la vista de nuevo proyecto un rectángulo que contiene las coordenadas en los ejes x e y en las que se encuentra el personaje. Esto permite que los

usuarios puedan saber la posición exacta del personaje y así sea más fácil utilizar los bloques de movimiento.

Otra modificación en el diseño se hizo sobre los inputs booleanos. Ahora, en lugar de mostrar sus alternativas en la lista desplegable como *Verdadero* y *Falso*, se muestran como V y F respectivamente. Este cambio es para que el input, cuando se seleccione una de estas alternativas, mantenga su forma de triángulo y se parezca más al gesto asociado a los bloques booleanos (recordar que su gesto es un triángulo).

También se realizó un cambio con respecto a los controles. El botón que se utiliza para arrastrar los bloques, que es el A se reemplazo por el Y. La razón de este cambio se debe a que se prefirió dividir las funcionalidad de selección con la de arrastrar, para que así cada uno tenga asignado su propio botón.

Por otra parte, se realizaron modificaciones al contenido del tutorial. Este originalmente tenía solamente información acerca del lenguaje y de como se podían crear bloques. Sin embargo, al tutorial le faltaba algo que permitiera probar que el usuario había entendido el correcto funcionamiento del IDE. Es por esto que se agregó una tarea guiada en el tutorial. Esta tarea consiste en que el usuario tiene que construir un pequeño código dentro del IDE. En el caso en que el usuario se equivoque en la creación del código durante esta tarea, se muestra un mensaje en pantalla notificando de cual fue el error. La actualización del tutorial se puede ver en el Apéndice C.

Un último cambio fue que se le asignó un color distinto a cada tipo de bloques, para que así puedan ser más fáciles de identificar y recordar. A los bloques numéricos se les asignó el color morado, a los booleanos el verde, al starter azul, a los de movimiento naranjo, al condicional #1 celeste y al condicional #2 rojo. Si bien Scratch fue una fuente de inspiración para el diseño de los bloques dentro del IDE, no se podían utilizar los mismos colores de los bloques que tiene ese lenguaje. Esto es debido a que las categorías en que Scratch divide sus bloques (que también le asocian un color a cada categoría), son distintas a las de este IDE. Por lo tanto, la única razón tras la elección de los colores de los bloques, es que los colores fueran fáciles de diferenciar.

En el Apéndice B se encuentran las vistas implementadas del IDE, que incluyen todos estos ajustes al diseño.

Capítulo 4

Implementación

En este capítulo se abordarán las decisiones técnicas principales tomadas durante el desarrollo del prototipo. Para ello, se describirán los criterios para la selección de *drivers* para el control de los Joy-Cons, así como se presentará el motor de desarrollo y extensiones programadas durante la implementación del IDE.

4.1. Elección del motor de desarrollo

Para la implementación del IDE se utilizó el motor de videojuegos Unity por las siguientes razones. La primera razón es que ya existía familiaridad con el uso de esta herramienta previo a este trabajo de título, por lo que ya había conocimiento de sus funcionalidades básicas. Por otra parte, Unity es popular dentro del rubro del desarrollo de videojuegos, lo cual ha hecho que cuente con un extenso soporte por parte de la comunidad. Finalmente, Unity cuenta con soporte para desarrollar videojuegos 2D e interfaces de usuario. Además, incluye una gran cantidad de documentación en su página web[1] para el desarrollo de estos. Debido a que dentro del IDE solamente se manejan objetos 2D, Unity permite desarrollar un software de estas características.

4.2. Elección de driver para los Joy-Cons

Para conectar los controles Joy-Con a un computador se tiene que hacer vía bluetooth. Si bien se puede realizar esta conexión, no existe un driver oficial que permita obtener el input entregado por los mandos. Esto se debe a que los controles fueron diseñados solamente para ser conectados a la consola Nintendo Switch. Incluso, dentro del motor de videojuegos Unity, no existe soporte nativo para los Joy-Con. Por lo tanto, para el desarrollo del IDE se necesitaba algún driver que permitiera al computador interactuar con los Joy-Con. Para solucionar esto, se buscaron drivers desarrollados por terceros que cumplieran esta función, y entre estos, elegir el que mejor se adecuará a los requerimientos IDE. Estos requerimientos son permitir la detección de la presión de botones, la detección analógica de las palancas y

[1] https://docs.unity3d.com/Manual/index.html

tener acceso a los valores devueltos por el giroscopio y acelerómetro del Joy-Con, para así poder generar los controles de movimiento.

Los drivers encontrados fueron los siguientes: *JoyConLib*, *BetterJoy*[2] y *JoyCon-Driver*[3]. El primero de estos, JoyConLib, solamente funciona dentro de un proyecto desarrollado en Unity. Un análisis de este driver se encuentra en el capítulo 2. Por el contrario, los otros dos drivers sirven no solo para conectar los Joy-Con a Unity, sino de forma que todo el computador tenga acceso (computadores de sistemas operativos Windows solamente).

Luego, se probaron las tres distintas alternativas para elegir el driver más apropiado para la implementación del IDE. La mejor opción y la que fue elegida para utilizar con el IDE, resulto ser la biblioteca JoyConLib, por las razones que se detallan a continuación. Primero que todo, el mecanismo de vibración puede ser controlado con más libertad, permitiendo controlar el tiempo de la vibración, la amplitud, las bajas y altas frecuencias del control, mientras que las otras no pueden controlar la amplitud. Lo segundo, y más importante, es que este era el único driver que permitía acceder a los valores devueltos por el giroscopio y acelerómetro de los Joy-Con. En el caso del driver BetterJoy, permite acceder al valor del giroscopio solamente para controlar el cursor del computador, por lo que no cumplía con el requerimiento del IDE. Finalmente, los otros drivers solo permiten ocupar los controles en computadores con sistema operativo Windows, mientras que esta opción permite la conexión con Windows y macOS. Esto es algo deseable debido a que permite que el IDE pueda ser utilizado por un público mayor, al incluir a los usuarios que tienen un computador con el sistema operativo macOS.

Cabe mencionar que JoyConLib tiene un problema al no poder detectar si se esta presionando el botón de captura del Joy-Con izquierdo. Sin embargo, esto no es vital para la implementación del IDE, pero sí es bueno tenerlo en cuenta en el caso que se quieran añadir nuevas funcionalidades al IDE. En la tabla 4.1 se resume la comparación de los 3 drivers.

4.3. Funcionalidades no implementadas

En el plazo que inicialmente se estableció para realizar la implementación el IDE, no se alcanzaban a desarrollar todas las funcionalidades mencionadas en el diseño (capítulo 3). Por lo tanto, para poder cumplir con las fechas, se optó por no implementar algunos aspectos del IDE. Primero que todo, no se implementaron los bloques de ciclos y los bloques que manejan variables. Esto significo que en el IDE se perdiera el uso de variables, al igual que la capacidad de crear ciclos. Tampoco se desarrolló uno de los métodos de borrado de bloques que consistía en arrastrar los bloques hacia un basurero que aparecía en pantalla. La razón por la cual no se desarrollaron estas funcionalidades en lugar de otras, es debido a que sin estas todavía se estaba logrando un producto mínimo viable. Sin embargo, se deja como trabajo futuro añadir estas funcionalidades al IDE.

[2] `https://github.com/Davidobot/BetterJoy`
[3] `https://github.com/fossephate/JoyCon-Driver`

Tabla 4.1: Comparación de los 3 drivers de Joy-Cons.

Características\ Controlador	JoyConLib	BetterJoy	JoyCon-Driver
Se puede conectar más de un Joy-Con	Sí	Sí	Sí
Detección análogica de las palancas	Sí, de ambas palancas	Sí, de ambas palancas	Sí, de ambas palancas
Detecta la presión de botones	Sí, de todos los botones excepto el de captura	Sí, de todos	Sí, de todos
Puede emitir vibraciones	Sí, pudiendo controlar duración de la vibración, amplitud, bajas y altas frecuencias. También se pueden activar o desactivar las vibraciones con un comando.	Sí, pudiendo controlar duración de la vibración, bajas y altas frecuencias.	Sí, pudiendo controlar duración de la vibración, bajas y altas frecuencias.
Detecta valores giroscopio	Sí, de ambos Joy-Cons (en los 3 ejes en radianes por segundo)	Solamente para controlar el cursor del computador	No
Detecta valores acelerómetro	Sí, de ambos Joy-Cons (en los 3 ejes en Gs)	No	No
Sistemas operativos soportados	Windows y macOS	Windows	Windows

4.4. Unity

Antes de continuar con la explicación de la implementación, hay que entender las siguientes características que tiene Unity como motor de juego.

4.4.1. Objetos

En el capítulo 2, se mencionaron varios aspectos de Unity incluyendo los *objetos* (GameObjects). Estos son la unidad básica dentro de las escenas, y los que van a poblarlas. Dentro de las escenas los objetos se ordenan de forma jerárquica, lo que permite incluso que estos puedan ser *anidados*. Esto último quiere decir que un objeto puede ser asignado como el hijo de otro. En la figura 4.1, se tiene un ejemplo de la jerarquía de objetos dentro de una escena. En esta escena el `Objeto_1`, es el padre de los objetos `Objeto_2`, `Objeto_4` y `Objeto_5`, el `Objeto_2` es el padre del `Objeto_3` y el `Objeto_7` es el padre del `Objeto_8`.

Los objetos dentro de la jerarquía de una escena se pueden *mover* a lo largo de ella, colocar un objeto dentro de otro para que sea su *hijo* e incluso *destruirlos*. Todas estas acciones se

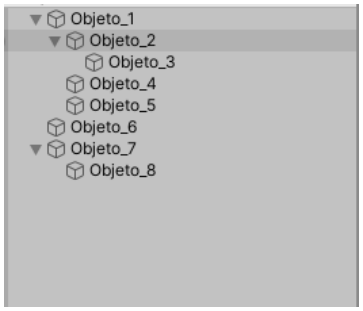


Figura 4.1: Ejemplo de la jerarquía de objetos dentro de una escena.

pueden hacer al editar la escena que lo contiene o por medio de código durante la ejecución de la aplicación.

Una de las propiedades que van a tener los objetos es si se encuentran *activados* o *desactivados*. Si un objeto se encuentra *desactivado*, las componentes que contenga no van a poder correr su lógica hasta que este se *active*. Se puede controlar este estado de los objetos mientras se edita una escena o a través de código dentro de la aplicación.

Otra propiedad de los objetos son las *capas* (layers). Las capas son etiquetas que permiten agrupar un grupo de objetos que tengan características similares. Un objeto puede tener asignada solamente *una* capa.

4.4.2. Componentes, MonoBehaviour y orden de ejecución

Como se mencionó en la subsección anterior, para que los *objetos* dentro de Unity tengan lógica se les tiene que añadir *componentes*. Se pueden añadir una gran cantidad de componentes a un objeto, y cada una se puede encargar de describir distintos aspectos de él, como puede ser su detección de colisiones, su posición, cómo se renderiza, entre otros.

Para poder crear nuevas componentes dentro de Unity, basta con crear una clase en C# y se extiende la clase `MonoBehaviour`. Esta clase viene incluida en Unity, y es la base para trabajar dentro del motor.

Al igual que los objetos, las componentes pueden estar *activadas* o *desactivadas*. Esta característica dentro de las componentes, afectan la forma en que son llamados algunos métodos dentro de la clase `MonoBehaviour`.

`MonoBehaviour` contiene una gran cantidad de métodos que se les puede hacer `override`. Muchas de estas tienen que ver con la ejecución de código mientras se corre la aplicación, por lo que son muy importantes. Algunas de estas son las siguientes:

- `void Awake()`: Este método se ejecuta solamente la *primera vez* que se inicializa el objeto.
- `void OnEnable()`: Este método se ejecuta *cada vez* que se active el objeto.
- `void Start()`: Este método se ejecuta solamente la *primera vez* que la componente esté activa.

- `void Update()`: Este método se ejecuta una vez en cada fotograma en que corre la aplicación, solamente si el objeto y la componente están *activadas*.
- `void OnDisable()`: Este método se ejecuta *cada vez* que se desactive el objeto.
- `void OnDestroy()`: Este método se ejecuta cuando el objeto es destruido.

Todos los métodos mencionados no retornan valor alguno. En la figura 4.2 se muestra un diagrama de flujo del orden de ejecución de estos métodos. Dentro de la figura anterior, se muestran cada uno de estos métodos conectados entre ellos por medio de flechas. Estas flechas representan el orden en que se tienen que ejecutar estos métodos. Por ejemplo, ya que hay una flecha desde el método `Awake` al `OnEnable`, se ejecuta primero `Awake` y después `OnEnable`.

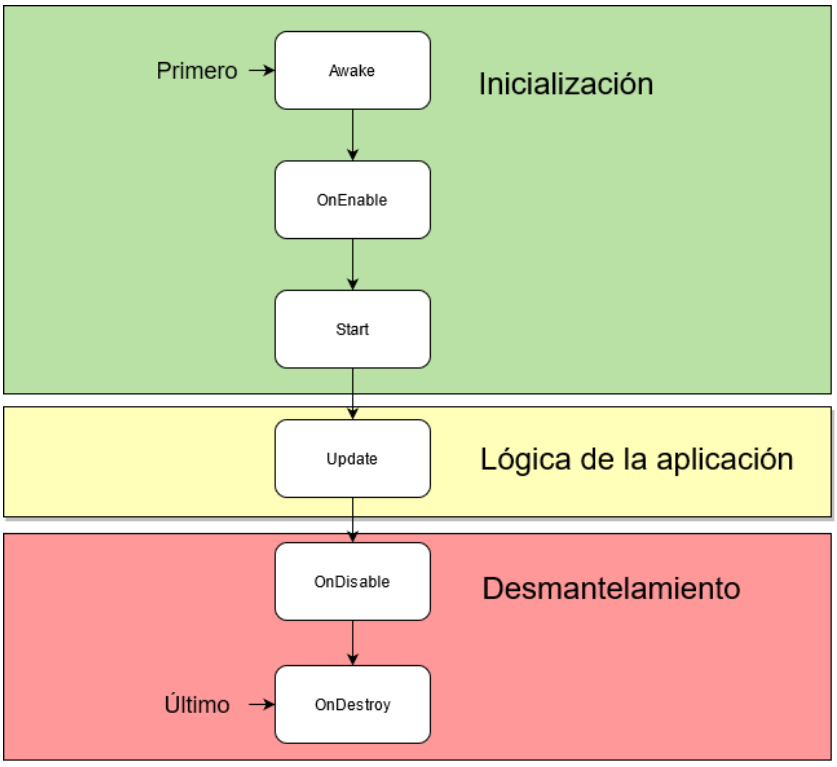


Figura 4.2: Orden de ejecución dentro de Unity.

Es importante destacar que el orden en que se ejecuta el código de las componentes dentro de Unity es *aleatorio*. Esto quiere decir, por ejemplo, que no se sabe qué componente va a ejecutar primero su método `Start`. Por otro lado, si una componente está ejecutando su método `Start`, se sabe que ya todas ejecutaron su método `Awake`.

El orden del código, sin embargo, se puede forzar a través de la herramienta *ScriptExecutionOrder* que viene incluida dentro de la misma plataforma. Con esta, se colocan las componentes que se deben ejecutar antes o después de las demás.

4.4.3. Transform

Una de las componentes predeterminadas más importantes es `Transform`. Esta componente la tiene cada objeto dentro de una escena, y permite manipular su posición, rotación

y escala.

4.4.4. Física y colisionadores

Uno de los aspectos fundamentales para los motores de videojuego son las *físicas*. El poder simular comportamientos físicos como lo puede ser la aceleración de un objeto, su gravedad, que lo afecten las colisiones con otros objetos y otras fuerzas, resulta de vital importancia al momento de desarrollar algún videojuego. Debido a esto, Unity incluye dos motores de físicas para manejar las simulaciones físicas. Estos son dos, debido a que Unity distingue entre físicas en una aplicación 2D y otra 3D.

Para agregar propiedades físicas a objetos dentro de Unity se tienen que ocupar ciertas componentes. Una de ellas son los *colisionadores* (`Colliders`) que permiten entregar la *forma física* al objeto. Esta forma es completamente invisible mientras corre la aplicación y permite entregar detección de colisiones al objeto. Hay distintas formas que se pueden agregar al objeto y se van a distinguir entre que sean 2D (cuadriláteros y círculos) y 3D (hexaedros y esferas).

Una de las utilidades que tienen los colisionadores, es si se mezclan con las *capas* de los objetos. Las capas van a permitir filtrar las colisiones entre ellos. Esto sería que un objeto solo puede colisionar con otro objeto dependiendo de que capa tiene asignada.

Otro uso que tienen los colisionadores con las capas es el método `Physics2D.OverlapBox`. Este permite definir un área rectangular en una determinada posición de la escena, y devuelve todos los colisionadores, que se encuentran o tocan esta área. Debido a que se obtienen los colisionadores, se puede determinar a qué capa pertenece cada una, y así poder hacer algo distinto dependiendo de la capa del colisionador.

4.4.5. Unity User Interface (Unity UI), GraphicRaycaster e imágenes

Unity posee un sistema especializado para manejar las interfaces de usuario llamado *Unity UI*. Esta entrega ciertas ventajas como es el posicionamiento de objetos, objetos de interfaz de usuario por defecto como botones, listas desplegables (dropdowns), inputs de texto, la detección de inputs del usuario por medio de un *sistema de eventos*, entre otros.

Para que el sistema de eventos detecte si el puntero del mouse está sobre un objeto o a qué objeto le hace clic, se utiliza un *Raycaster*. El raycaster permite utilizar *raycasting*, que es el proceso de disparar una línea invisible desde un punto en una dirección específica, para detectar si hay algún colisionador en el camino de la línea. Esto mismo se hace con el puntero, donde se realiza raycasting desde su posición para saber si hay algún objeto debajo de él. En el sistema de UI se utiliza un raycaster especial llamado `GraphicRaycaster` y es una componente dentro de Unity.

Una de las componentes básicas que van a tener los objetos es la de *imagen* (`Image`). Esta, como su nombre lo indica, permite asignar una imagen al objeto y renderizar en la posición donde se encuentre. Además, hay una variación de esta componente llamada `UI.Image`. Esta se utiliza especialmente para renderizar elementos de interfaz de usuario (UI) dentro de

una escena. Por esto incluye varias funcionalidades adicionales, donde destaca la propiedad *raycast target*. Esta propiedad puede estar activada o desactivada. En el primer caso, significa que la imagen que se le asigna al objeto, puede colisionar con el *raycast* generado por el `GraphicRaycaster`, lo que permite que el objeto pueda ser detectado. El segundo caso, simplemente no colisiona con el raycast, por lo que no puede ser detectado por este.

Con todos estos aspectos de Unity mencionados, ahora se puede analizar el resto de la implementación del prototipo.

4.5. Escenas

Para la implementación, se crearon dos escenas que van a manejar el IDE. La primera es la del *menú principal*, que, como su nombre lo indica, contiene la vista del menú principal del prototipo. Esta escena solo contiene la navegación entre los distintos botones que hay en pantalla para entrar a un nuevo proyecto, empezar el tutorial o salir del IDE.

La segunda escena es la de *nuevo proyecto*, que permite manejar las vistas de nuevo proyecto, creación de gestos y el tutorial. En las siguientes secciones se analiza la implementación de los objetos y características más importantes de esta escena, que son el cursor, los bloques y la detección de gestos.

4.6. Cursor

La principal función que tiene el cursor en el IDE es que permite interactuar con los objetos de la UI: los bloques, el personaje y los botones. Unity de forma nativa permite controlar botones en pantalla por medio del puntero del mouse, donde se detecta si este hizo clic sobre un botón o área en específico. También puede detectar si el mouse está intentando arrastrar un objeto. Sin embargo, no se tiene esto para mandos de videojuegos (incluyendo el Joy-Con), por lo que se diseñó un sistema que controle el cursor.

Para el cursor, primero se crea un objeto en la escena. A este se le asigna una imagen del cursor, y un script llamado `VirtualCursor`. `VirtualCursor` es la clase encargada de manejar la lógica del cursor. En la implementación del `VirtualCursor`, se aplican dos patrones de diseño: *Singleton* y *State*. Por un lado, el patrón Singleton, se utilizó para que solo hubiera una instancia única del cursor, lo que tiene sentido ya que no puede haber dos o más cursores en pantalla. El patrón State, por otro lado, se utilizó debido a que el cursor se comporta de forma distinta dependiendo del contexto. Por ejemplo, el cursor puede seleccionar botones normalmente, pero si está arrastrando un bloque, no puede hacerlo. Es por esto, que se dividió en 3 estados distintos:

1. `VCActive`: Este el estado por defecto del cursor. Aquí el usuario puede empezar a arrastrar bloques o al personaje, mover el cursor, entrar al modo creación de gestos , seleccionar botones en pantalla, entrar al menú de opciones, ir a la pantalla completa del área de trabajo y correr el código.
2. `VCDragging`: Este estado ocurre cuando el cursor está arrastrando un bloque o al personaje. Aquí el usuario puede soltar el objeto arrastrado, intentar el borrado del objeto

y mover el cursor junto al objeto.

3. `VCDisable`: Este estado ocurre cuando el cursor está desactivado, por lo que el usuario no puede interactuar con el cursor.

Cada estado posible del cursor es una clase que tiene que extender a la clase `VCState`. En la figura 4.3 se tiene un diagrama de jerarquía de las clases `VirtualCursor` y `VCState`. Como se puede apreciar en la figura, la clase `VirtualCursor` extiende a la clase `MonoBehaviour`, esto para que se comporte como componente y se pueda añadir a un objeto de Unity. También le permite utilizar el método `Update`, que se llama una vez en cada fotograma. En el `VirtualCursor` este método se le hace `override` para que, en cada fotograma, se tomen los inputs del usuario, y dependiendo del estado, realizar distintas acciones.

Figura 4.3: Diagrama de herencia de las clases `VirtualCursor` y `VCState`.

Por ejemplo, en el caso del estado `VCActive`, en cada llamada `Update` se va a comprobar el input entregado por la palanca del Joy-Con izquierdo, para mover el cursor en esa cantidad. También en este estado, se van a comprobar distintos botones del Joy-Con que están atados a otras acciones, como lo es seleccionar objetos en la pantalla.

4.7. Selección de objetos y Selectable

Los objetos dentro de la escena que pueden ser seleccionados deben tener una componente que implemente la interfaz `Selectable`. Esta interfaz diseñada cuenta con solamente un método: `void OnSelect()`. Las componentes que implementan esta interfaz colocan en este método que tiene que pasar cuando el objeto es seleccionado.

Para poder seleccionar objetos en pantalla con el cursor, se hace algo similar a como funcionan los clics de un mouse dentro de Unity. Se utiliza primero un `GraphicRaycaster` para realizar un raycast en el punto donde se encuentra el cursor. Luego, se verifica el primer objeto con el cual colisionó el raycaster. Si este objeto implementa la interfaz `Selectable`, se llama al método `OnSelect()`, causando entonces la selección del objeto. En caso contrario, no pasa nada. Es importante notar que para que los objetos puedan ser detectados por el raycaster deben tener una componente de `Image.UI` con la propiedad *raycast target* activada. Con esta componente también definen la imagen que tiene el objeto. Por lo tanto, los objetos que tengan la componente `Selectable` deben tener las características anteriores para que puedan ser seleccionados.

4.8. Arrastre de objetos y Draggable

De manera análoga a la selección de objetos, los objetos que se pueden arrastrar deben tener una componente que implemente la interfaz `Draggable`. Esta interfaz tiene varios mé-

todos, pero los que más destacan son los siguientes: `void OnDrag()` y `void Drop()`. En el primer método, las componentes que implementen esta interfaz colocan qué sucede cuando se comienza a arrastrar el objeto. Por el otro lado, en el segundo, colocan lo que sucede cuando se suelta.

La forma en que el cursor comienza a arrastrar objetos es la siguiente: con la componente `VirtualCursor` realiza, de forma análoga a la selección, un raycast con el `GraphicRaycaster` y se busca al primer objeto con el que colisionó. Se diferencia del método de selección, en que ahora se pregunta si el objeto implementa la interfaz `Draggable`. En el caso de que así sea, se llama al método `OnDrag()` del objeto. También, el `VirtualCursor` tiene que guardar una referencia del `Draggable` que está actualmente arrastrando. Para esto, tiene un campo llamado `draggedObj` de este mismo tipo, por lo que se le asigna la referencia del objeto en este campo.

El cursor mientras esté arrastrando al `Draggable`, cada vez que se mueva el cursor, el `Draggable` se moverá junto a él. Esto va a pasar hasta que se suelte el `Draggable`. Para soltarlo, el `VirtualCursor` llama al método `DropObject()`, el que quita la referencia del `Draggable` y llama al método `Drop()` de este último. Al igual que para la componente `Selectable`, los objetos que tengan la componente `Draggable`, deben tener además una componente `Image.UI` con la propiedad *raycast target* activada, para que puedan ser arrastrados.

4.9. Personaje

La componente que controla el comportamiento del personaje es el `CharacterController`. Esta clase implementa la interfaz `Draggable` para que pueda ser arrastrado por el cursor. También utiliza el patrón *Singleton*, debido a que solo puede haber un único personaje.

4.10. Bloques

La unidad fundamental del IDE desarrollado son los bloques. Para controlar el comportamiento de los bloques se utilizan distintas clases dependiendo del tipo de bloque. Sin embargo, todas estas clases **tienen** que implementar la interfaz `Block`. Esta es simplemente una extensión de la interfaz `Draggable` que no otorga métodos adicionales, pero sirve para distinguir los bloques de un objeto arrastrable. En la figura 4.4, se muestra un diagrama de herencia de la interfaz `Block`, cuyas componentes se detallan a continuación.

Los bloquen tienen una componente distinta dependiendo de la forma y valor que retornan. Estas componentes deben extender alguna de las siguientes componentes o implementar una de estas interfaces:

1. `CircleBlock`: Componente que representa a los bloques que tengan una forma circular y retornan valores numéricos en formato de punto flotante (`float`).
2. `TriangleBlock`: Componente que representa a los bloques que tengan una forma triangular y retornan valores booleanos.
3. `IMorFBlock`: Interfaz que representa a los bloques que tienen puertos macho **o** puertos hembra, y van a retornar valores vacíos (`void`).

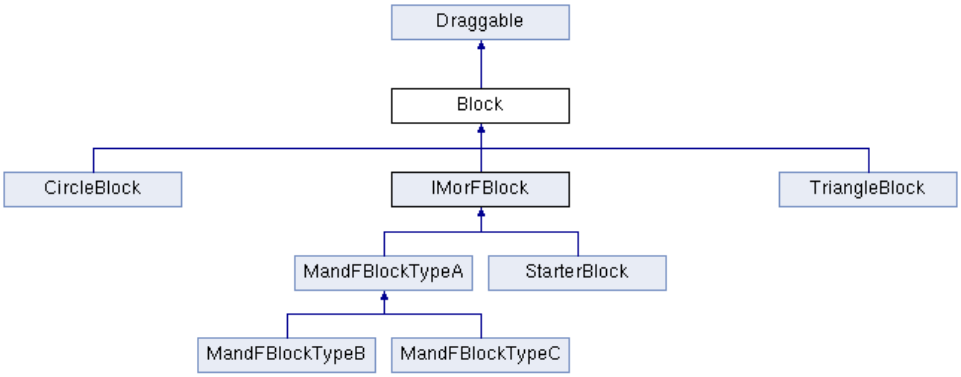


Figura 4.4: Diagrama de herencia de la interfaz `Block`.

Cada una de estas componentes e interfaz extienden o implementan, respectivamente, la clase `Block`. Además todas tienen un método llamado `Calculate()`, que evalúa y retorna la expresión del bloque. Sin embargo, se puede notar que el tipo que retorna este método, se va a diferenciar entre las componentes. En el caso de `CircleBlock`, retorna un número en formato de punto flotante, en el caso `TriangleBlock`, un booleano, y en los `IMorFBlock`, retorna vacío.

Los bloques que tienen puertos macho y hembra, pueden estar en dos tipos de posiciones. La primera posición es en la zona exterior del bloque y la segunda es la zona interior. Una representación gráfica de esto se puede ver en la figura 4.5. Un bloque si tiene un puerto interior hembra debe tener un puerto interior macho y viceversa, así que a este par de puertos se llama de forma más resumida *puerto interior*.

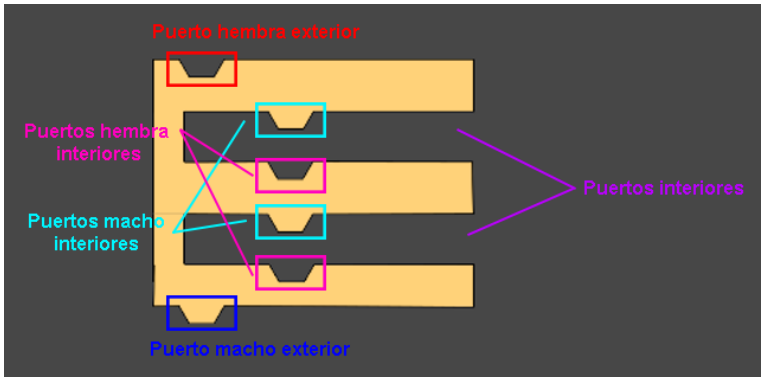


Figura 4.5: Puertos machos y hembra, exteriores e interiores.

Para poder diferenciar los bloques que tienen una cantidad distinta de puertos macho o hembra, se crean clases distintas que implementan directa o indirectamente la interfaz `IMorFBlock`. Estas son las siguientes:

1. `MandFBlockTypeA`: Componente que representa a los bloques que tienen 1 puerto macho y 1 hembra, ambos exteriores.
2. `MandFBlockTypeB`: Componente que representa a los bloques que tienen 1 puerto macho exterior, 1 hembra exterior y 1 puerto interior.

3. `MandFBlockTypeC`: Componente que representa a los bloques que tienen 1 puerto macho exterior, 1 hembra exterior y 2 puertos interiores.
4. `StarterBlock`: Componente que representa al bloque *Starter* que tiene solo 1 puerto macho exterior.

En la figura 4.6, se tienen los tipos de bloques con su respectiva forma.

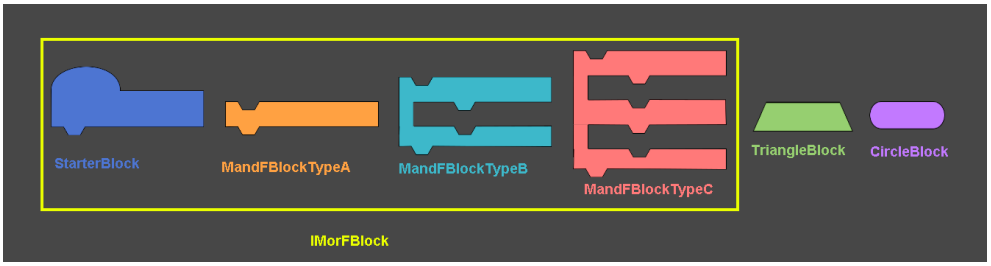


Figura 4.6: Las formas de los bloques con su respectiva clase o interfaz.

4.10.1. Detección de puertos e inputs

Debido a que la interfaz `Block` extiende a `Draggable`, se mantienen todos los métodos importantes como `Drop()` o `OnDrag()`, para las componentes (e interfaz) `CircleBlock`, `TriangleBlock` e `IMorFBlock`. El método `Drop()`, en especial, es muy importante, debido a que sirve para que, cuando se suelte un bloque, verificar si está encima de un puerto que le corresponde y conectarlo. En los siguientes párrafos se analiza cómo se logra este comportamiento.

Cada uno de los puertos que se encuentran en el IDE son un *objeto* (GameObject) dentro de una escena. Como se mencionó anteriormente, los objetos tienen una propiedad llamada *capa*, que permite agrupar a objetos de un mismo tipo. Esta funcionalidad se ocupa para distinguir un puerto de otro, por lo que, cada puerto, tiene una capa asignada dependiendo de su tipo. Las posibles capas que puede tener un puerto son las siguientes:

- *CircleLayer*: Capa de los puertos numéricos o circulares.
- *TriangleLayer*: Capa de los puertos booleanos o triangulares.
- *MaleLayer*: Capa de los puertos macho.
- *FemaleLayer*: Capa de los puertos hembra.

También, cada puerto tiene un colisionador cubriendo toda su área. Este colisionador no se ocupa para simular choques entre objetos o propiedades físicas como la gravedad. Todas estas características están desactivadas. La función del colisionador aquí es para que, cuando un bloque se suelte, este último pueda encontrarlo.

Cuando un bloque se suelta, se hace el llamado al método `Drop()` de él. Dentro de este método, cada bloque utiliza el llamado a la función `Physics2D.OverlapBox`. Como se mencionó antes, este método permite revisar si en una área rectangular en una determinada posición de la pantalla, hay colisionadores y los retorna todos en un arreglo. En este caso, se busca si el primer colisionador con el que choca, se encuentra algún puerto con el que se puede conectar. Esto último se verifica con la capa a la que pertenece el colisionador. Un

ejemplo de esto sería un `CircleBlock` que, cuando se suelta, verifica en el área que cubre si se encuentra algún colisionador con la capa *CircleLayer*. En el caso de que haya, este se conecta. Esto pasa de manera análoga con el `TriangleBlock` junto a la capa *TriangleLayer*. Por último, en el caso de los `IMorFBlock`, es algo distinto. En estos, se tiene que verificar en cada puerto macho que tenga, si en el área donde se encuentran, hay un colisionador con un *FemaleLayer*. También se tiene que verificar si en cada puerto hembra que tenga, si en el área donde se encuentran hay un colisionador con la capa *MaleLayer*. Si alguno de estos casos se cumple, el bloque se conecta.

Dependiendo del tipo de bloques, la forma en que se van a conectar es distinta. El primer caso que se tiene es la conexión entre los puertos numéricos con los `CircleBlock`. Estos puertos (al igual que los puertos booleanos) se diferencian del resto, en que actúan como input y puerto al mismo tiempo. Para lograr este comportamiento, utilizan una componente llamada `CircleInput`. Para reflejar la conexión de los `CircleBlock` con los `CircleInput`, estos últimos lo que hacen es guardar en un campo llamado `cb` una referencia del bloque que tienen conectado. En el caso que no haya ningún bloque conectado, el valor de `cb` es simplemente nulo. Finalmente, para su comportamiento como input, tienen un campo llamado `input` que permite modificar y guardar el input ingresado por el usuario. Debido a que este input puede ser seleccionado por el cursor, `CircleInput` implementa la interfaz `Selectable`. Los puertos booleanos realizan todo lo anterior de manera análoga, utilizando en lugar de `CircleInput` la componente `TriangleInput` y utilizando bloques `TriangleBlock` en lugar de `CircleBlock`.

En el caso de los puertos macho y hembra, la conexión va a ser distinta. Los bloques que poseen este tipo de puertos (es decir, que utilizan una componente que implementa `IMorFBlock`) representan la conexión entre ellos a través de arreglos. En el caso en que estos bloques se conecten solamente por sus puertos macho y hembra exteriores, se organizan como un arreglo, donde cada valor del arreglo es un bloque. El orden del arreglo es el mismo que el de la posición de los bloques conectados, recorriéndolos de arriba a abajo. En la figura 4.7, se tiene un ejemplo de bloques conectados por sus puertos exteriores y cada bloque tiene asignado un nombre que se encuentra dentro del mismo. El arreglo que representa la conexión de estos bloques es igual a:

```
[bloque_1, bloque_2, bloque_3, bloque_4, bloque_5]
```

Luego está el segundo caso, que es cuando se conectan los bloques por sus puertos interiores. Los bloques que poseen este tipo de puerto tienen un arreglo por cada puerto que tengan. En cada arreglo se guardan los bloques conectados a su respectivo puerto. En la figura 4.8 se tiene un ejemplo de bloques conectados por sus puertos interiores y exteriores. También, en la misma figura, cada bloque tiene asignado un nombre que se encuentra dentro de él. Para representar la conexión de bloques de la figura se utilizan tres arreglos:

1. `bloques`: Contiene los bloques conectados en su nivel más superficial.
2. `rama_a`: Contiene los bloques conectados al primer puerto interior (de arriba a abajo) del `bloque_3`.
3. `rama_b`: Contiene los bloques conectados al segundo puerto interior (de arriba a abajo) del `bloque_3`.

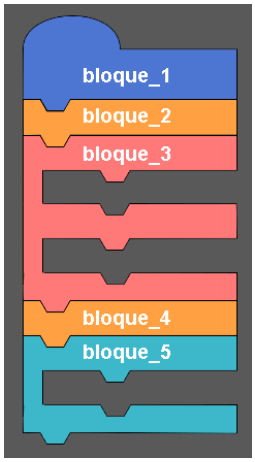


Figura 4.7: Ejemplo de bloques con puertos machos y hembra, conectados por sus puertos exteriores.

El valor de `bloques`, `rama_a` y `rama_b` son los siguientes:

```
bloques = [bloque_1, bloque_2, bloque_3, bloque_4];
rama_a = [bloque_a1, bloque_a2, bloque_a3]
rama_b = [bloque_b1, bloque_b2]
```

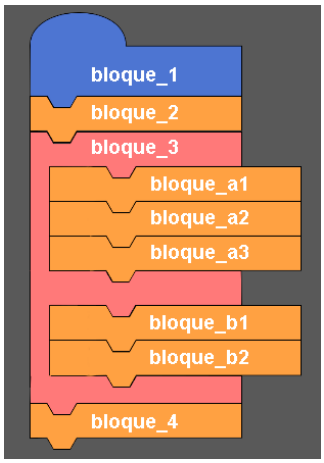


Figura 4.8: Ejemplo de bloques con puertos machos y hembra, conectados por sus puertos exteriores e interiores.

Es fácil ver que para conectar un bloque (con puertos macho y hembra) dentro de una cadena de bloques (que tienen puertos macho y hembra) conectados, simplemente se agrega al arreglo de la cadena en la posición deseada. Luego, para quitarlo, se quita del arreglo.

4.10.2. Ejecución del código de los bloques

Para realizar la ejecución del código de los bloques, se tiene que llamar al método `Calculate()` de cada uno de los bloques que están conectados al bloque `StarterBlock`. Para iniciar esto, se comienza por llamar al método `Calculate()` del mismo `StarterBlock`. En este bloque, el método lo que hace es primero obtener el arreglo que contiene los bloques que están co-

nectados al `StarterBlock`. Este se puede obtener fácilmente debido a que una cadena de bloques conectados por puertos macho y hembra exteriores, se organizan como un arreglo. Finalmente, se recorre el arreglo y se llama al método de `Calculate()` de cada uno de estos bloques.

Un detalle importante que hay que recordar de `StarterBlock` es que solamente puede haber uno en todo el proyecto. Por esto, solamente se va a llamar el `Calculate()` de un único `StarterBlock` cuando se corre el código. Para que se cumpla la unicidad del `StarterBlock`, este implementa el patrón de diseño Singleton.

Bloques `CircleBlock`

La mayoría de las componentes que utilizan los bloques que hasta ahora se han descrito (`CircleBlock`, `TriangleBlock`, `MandFBlockTypeA`, `MandFBlockTypeB` y `MandFBlockTypeC`), solamente sirven para definir la forma del bloque y el tipo de valor que retornan. Sin embargo, se necesita que estas representen el comportamiento de los distintos tipos de bloques con los que van a interactuar los usuarios, que fueron descritos anteriormente en el diseño del IDE. Es por esta razón que se crean nuevas componentes que extienden las anteriores. Cada una de estas nuevas componentes, para entregar este comportamiento, hacen `override` al método `Calculate()` de la clase que extienden, y dentro de este método describen y retornan lo que pasa al evaluar el bloque que representan (es decir, lo que pasa al ejecutar el código del bloque). En los siguientes párrafos se describen estas nuevas componentes.

Las componentes que extienden la clase `CircleBlock` son las siguientes:

1. `DivideBlock`: Representa al bloque *Divide*. Tiene 2 puertos numéricos. Su método `Calculate()` retorna la división entre los valores de sus dos puertos numéricos.
2. `MultiplyBlock`: Representa al bloque *Multiply*. Tiene 2 puertos numéricos.Su método `Calculate()` retorna la multiplicación entre los valores de sus dos puertos numéricos.
3. `SumBlock`: Representa al bloque *Sum*. Tiene 2 puertos numéricos. Su método `Calculate()` retorna la suma entre los valores de sus dos puertos numéricos.
4. `SubstractBlock`: Representa al bloque *Substract*. Tiene 2 puertos numéricos. Su método `Calculate()` retorna la resta entre los valores de sus dos puertos numéricos.
5. `XPosBlock`: Representa al bloque *X Position*. Su método `Calculate()` retorna la posición en el eje x del personaje.
6. `YPosBlock`: Representa al bloque *Y Position*. Su método `Calculate()` retorna la posición en el eje y del personaje.

En la figura 4.9 se tiene el diagrama de herencia de `CircleBlock`. Por el otro lado, en la figura 4.10 se tiene una tabla con el nombre de las componentes que extienden `CircleBlock`, junto al bloque que representan.

Bloques `TriangleBlock`

Las componentes que extienden la clase `TriangleBlock` son las siguientes:

1. `NotBlock`: Representa al bloque *Not*. Tiene 1 puerto booleano. Su método `Calculate()` retorna la operación NOT sobre el valor del puerto booleano.

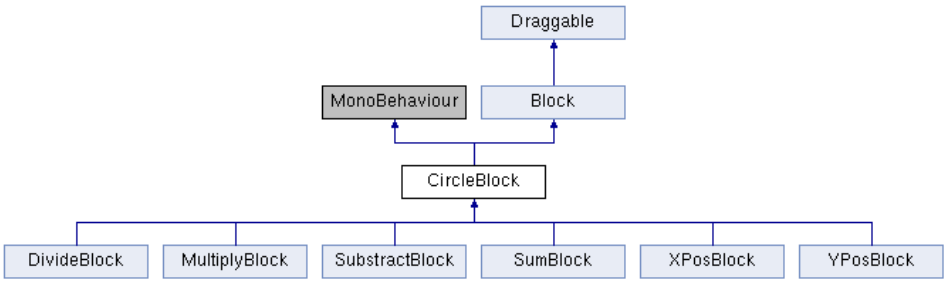


Figura 4.9: Diagrama de herencia de la clase `CircleBlock`

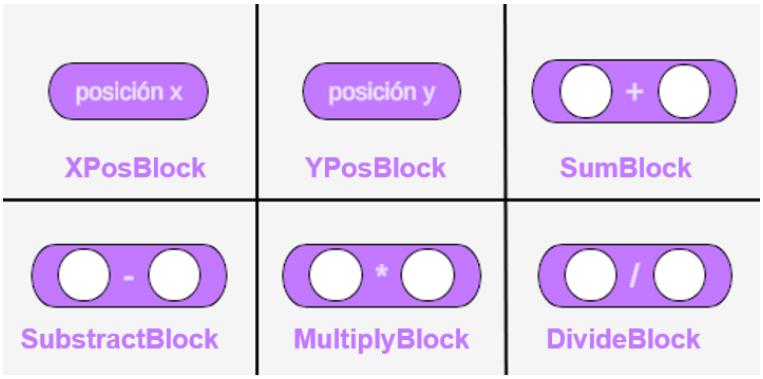


Figura 4.10: Representación gráfica de los bloques que extienden CircleBlock.

2. `AndBlock`: Representa al bloque *And*. Tiene 2 puertos booleanos. Su método `Calculate()` retorna la operación AND entre los valores de sus puertos booleanos.
3. `OrBlock`: Representa al bloque *Or*. Tiene 2 puertos booleanos. Su método `Calculate()` retorna la operación OR entre los valores de sus puertos booleanos.
4. `EqualsToBlock`: Representa al bloque *EqualsTo*. Tiene 2 puertos numéricos. Su método `Calculate()` retorna la operación igual entre los valores de sus dos puertos numéricos.
5. `GreaterThanBlock`: Representa al bloque *GreaterThan*. Tiene 2 puertos numéricos. Su método `Calculate()` retorna la operación mayor que entre los valores de sus dos puertos numéricos.
6. `LesserThanBlock`: Representa al bloque *LesserThan*. Tiene 2 puertos numéricos. Su método `Calculate()` retorna la operación menor que entre los valores de sus dos puertos numéricos.

En la figura 4.11 se tiene el diagrama de herencia de `TriangleBlock`. Por el otro lado, en la figura 4.12 se tiene una tabla con el nombre de las componentes que extienden `TriangleBlock`, junto al bloque que representan.

Bloques `IMorFBlock`

Las componentes que extienden la clase `MandFBlockTypeA` son las siguientes:

1. `MoveByBlock`: Representa al bloque *Move By*. Tiene 1 puerto numérico. Su método `Calculate()` mueve al personaje en x pasos (con x el valor de su puerto numérico) y

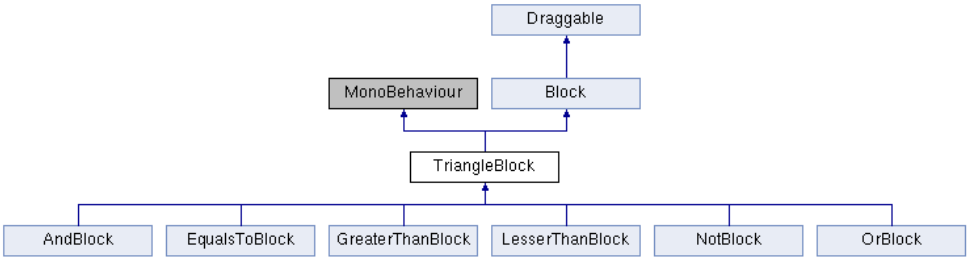


Figura 4.11: Diagrama de herencia de la clase `TriangleBlock`.

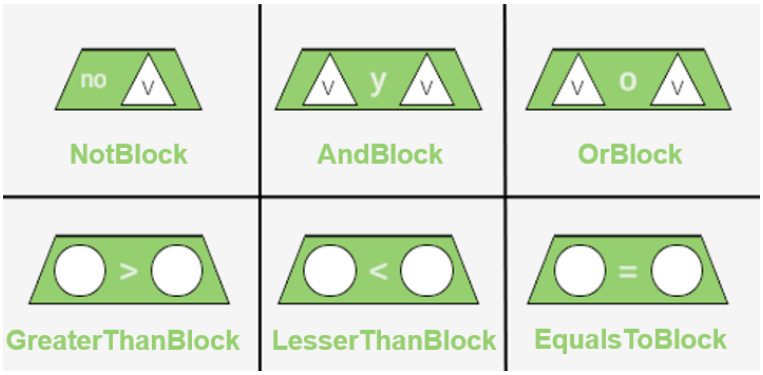


Figura 4.12: Representación gráfica de los bloques que extienden TriangleBlock.

en una dirección que puede ser hacia arriba, hacia abajo, izquierda o derecha.

2. `MoveToBlock`: Representa al bloque *Move To*. Tiene 2 puertos numéricos. Su método `Calculate()` coloca al personaje en las coordenadas (x, y) con x e y valores de cada uno de sus puertos numéricos.

La componente que extiende la clase `MandFBlockTypeB` es solamente `IfBlock`. Representa al bloque *If* o Condicional #1. Este tiene 1 puerto booleano y su método `Calculate()` verifica el valor de su puerto booleano. En el caso que sea verdadero, ejecuta el método `Calculate()` de todos los bloques conectados a su puerto interior. Esto se logra al utilizar y recorrer el arreglo que contiene los bloques conectados a este puerto. En el caso contrario, no pasa nada.

La componente que extiende la clase `MandFBlockTypeC` es solamente `IfElseBlock`. Representa al bloque *IfElse* o Condicional #2. Este tiene 1 puerto booleano y su método `Calculate()` verifica el valor de su puerto booleano. En el caso que sea verdadero, ejecuta el método `Calculate()` de todos los bloques conectados a su puerto interior que contiene la rama verdadera. En el caso contrario, se hace lo mismo pero con los bloques conectados al puerto interior que contiene la rama falsa.

En la figura 4.13 se tiene el diagrama de herencia de `IMorFBlock`. Por el otro lado, en la figura 4.14 se tiene una tabla con el nombre de las componentes que extienden `IMorFBlock`, junto al bloque que representan.

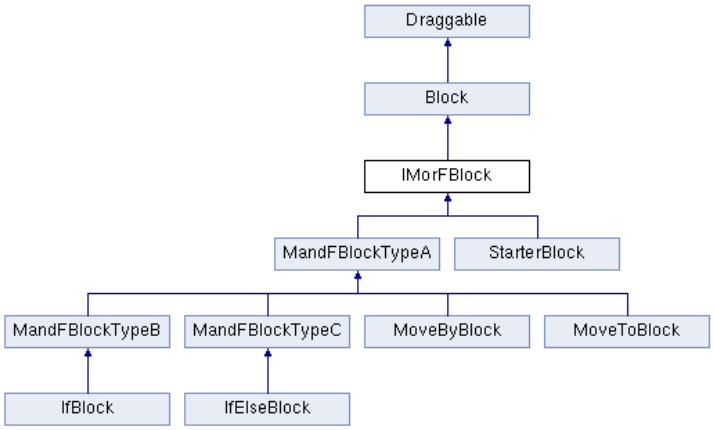


Figura 4.13: Diagrama de herencia de `IMorFBlock`.

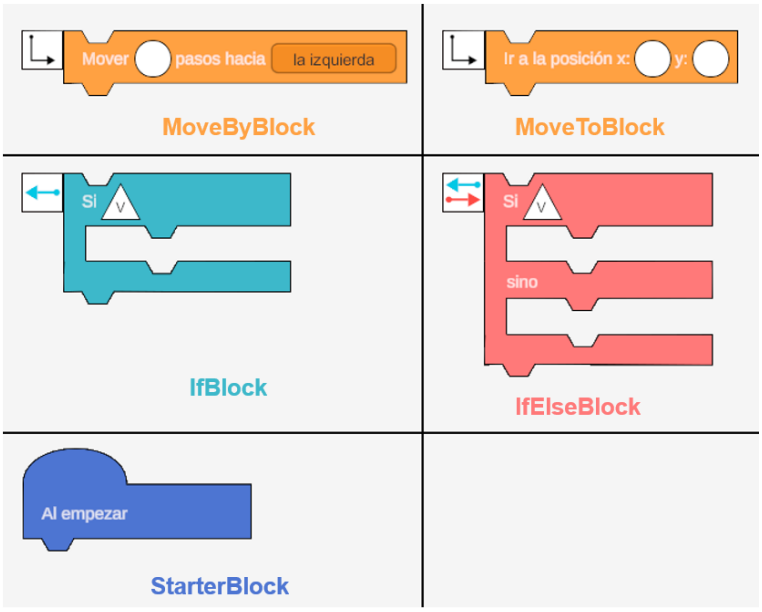


Figura 4.14: Representación gráfica de los bloques que implementan `IMorFBlock`.

4.10.3. Evaluar el valor de los puertos numéricos y booleanos

Para obtener el valor de los puertos numéricos, su componente `CircleInput` tiene un método llamado `CalculateFloat()`, que retorna un número en formato de punto flotante. Este lo que hace es verificar si es que hay un bloque `CircleBlock` conectado, donde pueden haber dos casos:

1. No hay un bloque `CircleBlock` conectado, por lo que retorna el campo que contiene el valor que el usuario ingreso para el puerto.
2. Hay un bloque `CircleBlock` conectado, por lo que retorna el valor de llamar `Calculate()` sobre el bloque conectado.

En el caso de los puertos booleanos, es análogo, pero con la componente `TriangleInput`, y su método en lugar de llamarse `CalculateFloat()`, se llama `CalculateBool()`, que retorna

un booleano.

Para que los bloques que tienen algún tipo de estos puertos puedan obtener su valor con estos métodos (por ejemplo, cuando realizan su método `Calculate()`), estos poseen campos donde guardan las referencias a sus puertos.

4.10.4. Análisis del lenguaje

El lenguaje del IDE diseñado con los bloques cuenta con distintas fortalezas y debilidades que lo definen. Lo primero que se puede notar es que el lenguaje no necesita de un parser para pasar de una sintaxis concreta a una abstracta. Esto se debe a que el código de los bloques ya se encuentra escrito en una sintaxis abstracta y los usuarios interactúan directamente con los bloques. Esto se puede ver como una fortaleza, debido a que se evita la necesidad de un parser.

Otra fortaleza del lenguaje es que es fácil de añadir nuevos bloques. Debido a que ya se tienen establecidos componentes que definen las formas de los bloques, estos se pueden extender para generar nuevos bloques. Cuando se extiende hay que agregar que es lo que haría su método `Calculate()`, que sería la funcionalidad del nuevo bloque y si tiene puertos numéricos o booleanos.

Una gran debilidad del lenguaje es que es muy susceptible a *stack overflows*. Esto sucede debido a que el lenguaje cuenta con una gran cantidad de llamadas recursivas para cumplir sus funciones básicas. Por ejemplo, para obtener los valores de los bloques `CircleBlock` que se encuentran conectados a un puerto numérico, se tienen que hacer llamadas recursivas de `Calculate()`. Sucede de manera análoga, en los `TriangleBlock` con los puertos booleanos. Finalmente, cuando se tienen conectados bloques `IfBlock` o `IfElseBlock` de forma anidada, pasan también llamadas recursivas cuando se llaman los métodos `Calculate()` de los bloques conectados a sus puertos interiores. Sin embargo, esta debilidad se puede evitar si se utiliza una técnica llamada trampolín. Esta técnica permite que se realicen optimizaciones de llamadas de cola (*Tail Call Optimization*) sobre funciones recursivas o que realicen llamados en la posición de cola [4]. Con esto, los llamados recursivos que se realizan en el lenguaje, no colapsarían el call stack, evitando los stack overflow. Esta funcionalidad no fue implementada debido a que se privilegiaron otros aspectos del IDE sobre la optimización del lenguaje.

4.11. Controles de movimiento y detección de gestos

Uno de los pilares fundamentales de este IDE es la detección de gestos. Esta sirve para la creación de bloques y para entregar una dirección a los bloques *Move By*. La clase que se encarga de la detección de gestos y los controles de movimiento realizados por los Joy-Cons es el `GestureController`. Antes de poder explicar en detalle el `GestureController`, se van a analizar los elementos importantes de este.

[4] Más información acerca de la Tail Call Optimization y el trampolín en el siguiente enlace: `https://users.dcc.uchile.cl/~etanter/recursion/Recursion__Efficiency_Considerations.html`

4.11.1. Gesture Cursors

Para que el usuario pueda ver de forma gráfica los movimientos que realiza con los Joy-Con se utilizan dos punteros, uno para el izquierdo, y otro para el derecho, que se llaman *Gesture Cursors*. Estos se mueven dependiendo del eje en que se rote el Joy-Con asociado y a la velocidad en que lo haga. También, cuando se presiona el gatillo del Joy-Con, el Gesture Cursor asociado comienza a dejar un rastro por los lugares que paso, hasta que este se suelte.

La implementación del movimiento de los Gesture Cursors se basa en el *giroscopio* que posee tanto el Joy-Con derecho como el izquierdo. El giroscopio de los Joy-Con permite obtener la velocidad angular en la que se encuentra en sus 3 ejes de rotación (X, Y, Z). Con el driver utilizado para los Joy-Con, se puede obtener este valor con el método `GetGyro()`, que entrega la velocidad en radianes por segundo[5]. En la figura 4.15, se tienen los ejes de rotación que tiene un Joy-Con.

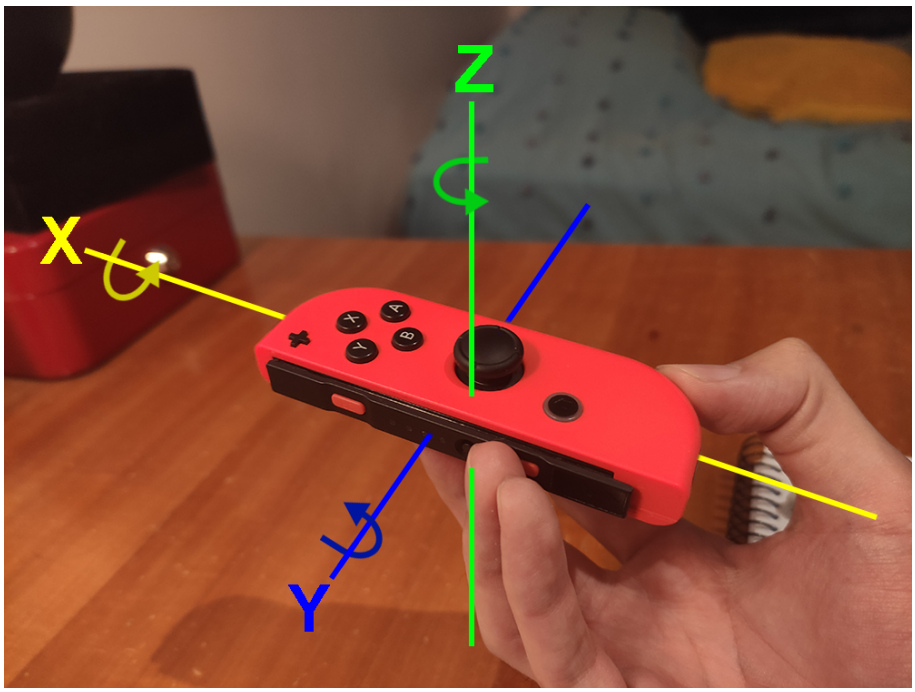


Figura 4.15: Ejes de rotación Joy-Con.

El movimiento de cada Gesture Cursor en el eje X, se hace dependiendo del movimiento de su Joy-Con asociado en el eje de rotación Z. Por el otro lado, su movimiento en el eje Y, depende del eje de rotación Y de su Joy-Con asociado.

Además, el movimiento de los Gesture Cursors se controla en las llamadas `Update()` de la clase `GestureController`. En este método se utilizan los valores de la velocidad angular de los Joy-Cons para calcular la distancia que se deben mover. Debido a que este valor es una velocidad, para pasarlo a distancia, se debe multiplicar por una magnitud de tiempo. En este caso Unity, posee el valor de `Time.deltaTime`, que es el tiempo, en segundos, entre

[5]Que la magnitud de la velocidad angular devuelta por el método `GetGyro()` sea en radianes por segundo, se obtuvo del código fuente del driver en el siguiente enlace: `https://github.com/Looking-Glass/JoyconLib/blob/master/Packages/com.lookingglass.joyconlib/JoyconDemo.cs`

el fotograma actual y el anterior. Por lo tanto, con estos dos valores, se puede obtener la distancia en que se deben mover los Gesture Cursors. También, en el caso en que se quiera ajustar la sensibilidad en que se mueven, se puede multiplicar la distancia a mover por un valor constante.

Finalmente, cada vez que se entra al modo de creación de gestos, se posicionan los Gesture Cursors en la mitad de la pantalla. Debido a que la implementación del control de los Gesture Cursors se basa en el giro de los Joy-Cons, antes de entrar a este modo, los controles deben estar apuntando hacia la pantalla. Si no se hace lo anterior, los Gesture Cursors se vuelven difíciles de controlar.

4.11.2. Creación y reconocimiento de gestos

Para la detección de gestos se utiliza la biblioteca *Gesture Recognizer*[6], que se obtuvo de la *Unity Asset Store*. Esta librería permite la creación y detección de gestos que sean hechos con el mouse. El funcionamiento de la detección de gestos es el siguiente: Se establece un área designada donde se pueden realizar los gestos. Luego, si sobre esa área se hace clic izquierdo con el mouse y se mantiene presionado, se comienza a trazar una línea por los lugares que pasa el mouse. Esto pasa hasta que se suelte el clic izquierdo, y se realiza la detección del gesto hecho por el usuario. También, se pueden seguir dibujando líneas, ya que, en esta herramienta, un gesto puede estar formado por más de una línea. Si bien, esta herramienta está hecha para funcionar solamente con mouse, se modificó para que funcionará con el movimiento de los dos Joy-Con.

Para representar los datos de un gesto, se utiliza una clase llamada `GestureData`. Esta representa a los datos de un gesto y lo único que contiene es un campo con una lista de `GestureLine`, otra clase de la librería. `GestureLine`, es simplemente una lista de puntos en un plano 2D. Lo que representan estos puntos, es la línea que se forma al conectarlos entre sí. Por lo tanto, cada `GestureLine` representa una línea que forma parte de un gesto.

Gesture Recognizer cuenta con un sistema para manejar y conservar los gestos que se van a reconocer. Esto se hace a través de un tipo especial de archivos, llamado `GesturePattern`. Cada archivo de este tipo se le coloca un identificador (que es un string) y un `GestureData`, es decir, los datos del gesto. De esta forma, se puede hacer un archivo para cada gesto que se quiere reconocer. También, a los `GesturePatttern` se les puede restringir que el gesto solamente se puede hacer en una dirección.

Para la detección de gestos, se encarga una clase llamada `Recognizer`. En esta, se ingresa una lista de los gestos que pueden ser detectados (archivos de tipo `GesturePattern`). Luego, se puede llamar a su método `Recognize`, y en su argumento se colocan los datos de un gesto (GestureData). Aquí, se comprueba a cuál gesto de la lista del `Recognizer` se asemejan más los datos ingresados, y se devuelve el identificador de este, y el porcentaje de semejanza entre ambos.

Un detalle importante del método `Recognize` es que no entrega su respuesta inmediatamente, porque tarda un poco la detección del gesto. Este tiempo crece a medida que la lista

[6] `https://assetstore.unity.com/packages/tools/input-management/gesture-recognizer-86410`

de patrones de gestos sea mayor. Es por esto que la ejecución de este método se hace en un *thread* aparte.

Además de estas clases, la biblioteca incluye otra llamada `DrawDetector`, que se encarga de capturar el gesto que dibuja el usuario con el mouse, guardarlo y mostrar en pantalla que patrón dibujo. Sin embargo, ya que se quiere utilizar los Joy-Cons y no el mouse, esta clase no se utiliza y es reemplazada por la clase `GestureController`.

En `GestureController` se realiza la detección de gestos con los Joy-Cons. Para esto, lo que hace es que, cuando se presiona el gatillo de un Joy-Con (es decir, se comienza a realizar el gesto), se crea un objeto `GestureData`, y se van almacenando en él las posiciones que va recorriendo su `GestureCursor` asociado. Cuando se suelta el gatillo, se manda a detectar por el `Recognizer` con el método `Recognize`, el gesto realizado con el Joy-Con. Finalmente se utiliza la respuesta del `Recognizer` para crear un bloque, mostrar el menú de bloques o asignar una dirección al *Move By*, que este asociado a ese patrón.

Hay que recordar que en el IDE existen dos conjuntos de gestos que se pueden realizar. El primero consisten los gestos que permiten crear bloques, y los segundos, los que entregan la dirección del bloque *Move By*. Para diferenciar estos dos conjuntos en el IDE, simplemente se instancian dos clases `Recognizer`, una para cada conjunto de gestos. Luego, cuando se tiene que realizar la detección, se elige cual `Recognizer` se tiene que utilizar, dependiendo de si se quieren crear bloques o asignar direcciones al bloque.

Un último detalle de la detección de gestos son los gestos que se realizan con los dos Joy-Con. Este tipo de gestos se llaman *gestos combo* y para detectarlos se tiene que aplicar el `Recognizer` dos veces, una vez en el gesto realizado con el Joy-Con izquierdo, y otro para el derecho. Luego, se comprueba en el `GestureController` que ambos gestos fueron realizados correctamente.

4.11.3. Estados del GestureController

Para manejar la detección de gestos, el `GestureController` utiliza el patrón State. Los estados que tiene son los siguientes:

1. `GCActive`: Es el estado por defecto del `GestureController`. Aquí el usuario puede comenzar a dibujar algún gesto con alguno de sus dos controles Joy-Con, o salir del modo creación de gestos. Dependiendo si presiona el gatillo izquierdo, derecho o ambos, pasa al estado `GCReadingLeft`, `GCReadingRight` y `GCReadingBoth`, respectivamente.
2. `GCReadingRight`: En este estado, el usuario se encuentra actualmente dibujando un gesto con el control derecho. El usuario puede mover los Gesture Cursors pero si suelta el gatillo derecho, comienza a detectar el gesto que realizó, y pasa al estado `GCWait`. En el caso que presione el gatillo izquierdo, inicia un gesto combo y pasa al estado `GCReadingBoth`.
3. `GCReadingLeft`: En este estado, el usuario se encuentra actualmente dibujando un gesto con el control izquierdo. El usuario puede mover los Gesture Cursors pero si suelta el gatillo izquierdo, comienza a detectar el gesto que realizó, y pasa al estado `GCWait`. En el caso presione el gatillo derecho, inicia un gesto combo y pasa al estado `GCReadingBoth`.

4. `GCReadingBoth`: En este estado, el usuario se encuentra actualmente dibujando un gesto con ambos controles. El usuario puede mover los Gesture Cursors pero si suelta ambos gatillos, comienza a detectar el gesto que realizó y pasa al estado `GCWait`.
5. `GCWait`: En este estado, el usuario está a la espera del resultado de la detección de gestos. El usuario puede mover los cursores pero no trazar un nuevo gesto con los gatillos. Al finalizar la detección vuelve al estado `GCActive`.

Cada estado posible del `GestureController` es una clase tiene que extender a la clase `GCState`. En la figura 4.16, se tiene un diagrama de herencia de las clases `GestureController` y `GCState`.

Figura 4.16: Diagrama de herencia de `GestureController` y `GCState`.

Capítulo 5

Prueba de Concepto

Con el objetivo de poner a prueba el IDE realizado, se realizó un experimento sobre un grupo de usuarios. Estos usuarios se caracterizaban por ser personas de edades entre 22 y 27 años que ya tenían conocimientos de programación. El experimento consistió en que los usuarios tenían que interactuar con el IDE desarrollado y realizar sobre él, ciertas tareas. Luego de terminar todas las tareas, estos tenían que responder un cuestionario con respecto a su experiencia y primeras impresiones sobre el IDE.

5.1. Metodología

El objetivo de esta validación tiene un carácter preliminar, debido a que solamente se quiere comprobar el correcto funcionamiento del IDE. En consecuencia, el experimento toma la forma de una *prueba de concepto*. Por esta misma razón, los participantes del experimento no fueron los usuarios objetivo (niños de edades entre 10 y 12 años), sino personas mayores que ya tenían conocimientos de programación. Por ende, el objetivo principal de esta evaluación fue explorar si el protocolo experimental llevado a cabo puede ser extendido para aplicarlo a una muestra que sea más grande y que cuente solamente con los usuarios objetivo.

Un aspecto importante que debía tener este experimento, es que se realizara de forma remota. Esto debido a que en el momento en que se realizaron las pruebas, todavía se encontraba la pandemia del COVID-19. Las medidas como el distanciamiento social y las cuarentenas limitaron las formas en que se podían realizar las pruebas, obligando entonces a este formato.

5.1.1. Participantes

Se reclutó un total de 6 personas para que participaran en las pruebas. Esta muestra de usuarios estuvo compuesta por estudiantes y egresados de la Facultad de Ciencias Físicas y Matemáticas de la Universidad de Chile, presentándose 1 mujer y 5 hombres de edades entre 22 y 27 años.

Debido a la pandemia y al formato remoto en que se realizaron las pruebas, la selección de participantes exigía que estos ya contaran con todos los materiales. En los casos en que

su computador no contará con un adaptador de bluetooth, se les podía enviar uno que se conectará vía USB a sus casas. El que todos los participantes tuvieran los mandos Joy-Con para participar, tuvo como consecuencia que todos ya tuvieran experiencia ocupando los controles antes de haber realizado la prueba.

Otra característica importante de esta muestra es que la mayoría tenía altos conocimientos de programación ya que estos estudiaban o habían estudiado en el Departamento de Ciencias de la Computación de la Facultad de Ciencias Físicas y Matemáticas.

5.1.2. Materiales

Para las pruebas, cada usuario tenía que contar con un computador con las siguientes características:

1. Sistema operativo *Windows 10*
2. Entrada de *bluetooth*
3. La aplicación para videollamadas *Zoom*
4. Cámara web
5. Micrófono

Si bien el IDE es compatible con el sistema operativo *macOS*, solamente se había probado el correcto funcionamiento del IDE sobre Windows 10 y no sobre este. Por lo tanto, todas las pruebas se hicieron sobre este último sistema operativo para evitar inconvenientes.

Además del computador, el usuario debía contar con los mandos Joy-Con y una conexión a internet estable.

5.1.3. Procedimiento

Antes que todo, cada prueba consistió en una videollamada 1 a 1 con cada participante, por lo que se hicieron un total de 6 pruebas. Para iniciar la prueba, primero se contactó al participante a través de una videollamada en la aplicación *Zoom*. Luego se realizó una breve introducción sobre el contexto de esta prueba, cuál era su objetivo y que forma parte de un trabajo de memoria. Dentro de esta introducción, se explicó el concepto del pensamiento computacional y por qué es importante que los niños lo aprendan. También, se explicó el funcionamiento de los lenguajes de programación por bloques que actualmente existen, tomando como ejemplo *Scratch*. Finalmente, se explicó el IDE que se desarrolló, en qué consiste y cómo podría ser una nueva alternativa para que niños puedan desarrollar el pensamiento computacional.

Después de la introducción, al participante se le explicaron las actividades que tenía que realizar a lo largo de la prueba. Estas se tuvieron que realizar dentro del IDE, el cual fue enviado a ellos previamente a la reunión. También para la actividad, se les pidió a los usuarios que siguieran el método *Thinking Aloud*, que consiste en que digan en voz alta sus pensamientos mientras realizan las tareas.

Las tareas a realizar por cada participante fueron las siguientes:

1. Realizar el tutorial.
2. Hacer un bloque de cada gesto.
3. Conectar bloques para que al correrlos hagan lo siguiente:

   "*Primero se debe verificar si la posición del personaje en el eje X es menor a 120.*

   - *En el caso que sea menor: Mover el personaje a la derecha 20 + 10 pasos hacia la derecha.*
   - *En el caso que sea mayor o igual: Posicionar el personaje en las coordenadas x: -200 y: 0*

   Luego de hacer eso, verificar si la posición del personaje en el eje X es mayor a la posición en el eje Y.

   - *Si se cumple esta condición: Mover al personaje 10 pasos hacia arriba*
   - *Si no se cumple: nada"*

Entre las tareas 1 y 2 se dan 5 minutos al participante para que pruebe las distintas funcionalidades del IDE.

La solución de la tarea 3 se encuentra en la figura 5.1.

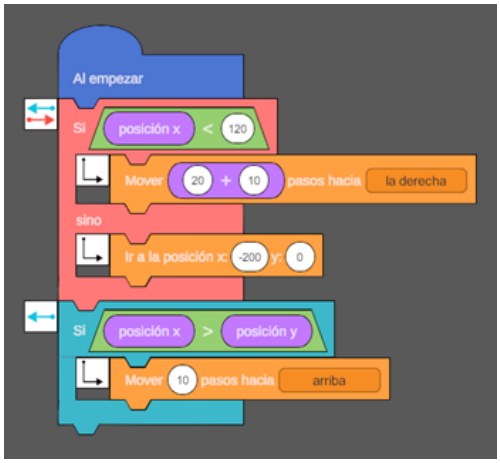


Figura 5.1: Solución tarea 3 de la prueba de concepto.

Antes de empezar las tareas, se le dieron instrucciones al usuario para que conecte sus Joy-Con vía bluetooth a su computador. Al realizar esto puede iniciar el IDE y empezar la actividad. Para que se pueda observar el desarrollo de la actividad, a los usuarios se les pidió compartir su pantalla y activar la cámara web de su computador. También se les pidió si la sesión podía ser grabada, para un posterior análisis.

Una vez terminadas las tareas, a cada participante se le solicitó completar un cuestionario para evaluar la usabilidad del IDE y su experiencia con él. El cuestionario cuenta con distintos ítems, y cada uno contiene una afirmación donde el usuario marca su evaluación en una escala Likert de cinco puntos: *muy de acuerdo*, *de acuerdo*, *neutro*, *en desacuerdo* o *muy en desacuerdo*. Cada uno de estos valores se califican en un puntaje del 1 al 5, tal que: Muy en Desacuerdo = 1 punto y Muy de Acuerdo = 5 puntos. Las afirmaciones están hechas tales que mientras mayor sea el puntaje asignado por el usuario, más usable es considerado el IDE. Este cuestionario el usuario lo rellena durante la videollamada, y cada vez que marque

un ítem debe entregar la justificación tras su decisión. Además, se encuentra dividido en 3 partes, donde se evalúan distintos aspectos del IDE, y estos son: *Ítems acerca del tutorial*, *ítems acerca de los controles* e *ítems acerca de la aplicación en general*. El cuestionario se puede ver en la tabla 5.1.

El cuestionario utilizado se basa en parte en el *System usability scale* desarrollado por John Brooke[4], que es un cuestionario de 10 ítems que también se evalúan por medio de la escala de Likert, y es un estándar de industria para medir la usabilidad de un sistema.

Tabla 5.1: Cuestionario prueba de concepto.

Ítems acerca del tutorial	**Muy de Acuerdo**	**De Acuerdo**	**Neutro**	**En Desacuerdo**	**Muy en Desacuerdo**
1. El tutorial es una buena guía para empezar a trabajar sobre la aplicación					
2. Las instrucciones del tutorial fueron muy claras					
3. El tutorial es de una extensión razonable.					

Ítems acerca de los controles	**Muy de Acuerdo**	**De Acuerdo**	**Neutro**	**En Desacuerdo**	**Muy en Desacuerdo**
1. Los gestos fueron muy fáciles de recordar					
2. Los gestos fueron muy intuitivos					
3. La detección de gestos fue muy confiable					
4. Los controles (sin contar los de movimiento) fueron muy intuitivos dentro de la aplicación					
5. Los controles de movimiento fueron muy intuitivos dentro de la aplicación					

Ítems acerca de la aplicación en general	**Muy de Acuerdo**	**De Acuerdo**	**Neutro**	**En Desacuerdo**	**Muy en Desacuerdo**
1. La aplicación es fácil de navegar.					
2. Es fácil de encontrar la información deseada.					
3. La aplicación es muy fácil de usar					
4. Un niño de 10 años aprendería rápidamente cómo utilizar esta aplicación					
5. Hay mucha consistencia en la aplicación					
6. Las distintas funcionalidades estan bien integradas					

5.2. Resultados

A continuación se presentan los resultados principales obtenidos durante la prueba de concepto. Es importante mencionar que estos resultados solo deben interpretarse como pertenecientes a la muestra observada y no buscan generalizarse a alguna población, dado lo limitado del tamaño muestral.

Primero, todos los participantes lograron realizar todas las tareas entregadas, en un tiempo promedio de 21 minutos, donde se incluyen los 5 minutos que hay entre las tareas 1 y 2. También, la mayoría de los usuarios encontraron el IDE entretenido, sencillo y que se sentía como una experiencia distinta a otros lenguajes de programación.

5.2.1. Cuestionario

El promedio de los resultados del cuestionario por ítem, se pueden ver en la tabla 5.2. Lo primero que se puede apreciar, es que en la mayoría de los ítems se obtuvo un puntaje ≥ 4. Lo

anterior indica que la mayoría de los participantes estaban de acuerdo con las afirmaciones.

Tabla 5.2: Resultado promedio del cuestionario.

Ítems acerca del tutorial	**Puntaje Promedio**	**Desviación Estándar**
1. El tutorial es una buena guía para empezar a trabajar sobre la aplicación	4,167	0,408
2. Las instrucciones del tutorial fueron muy claras	4,833	0,408
3. El tutorial es de una extensión razonable.	5	0

Ítems acerca de los controles	**Puntaje Promedio**	**Desviación Estándar**
1. Los gestos fueron muy fáciles de recordar	4,5	0,837
2. Los gestos fueron muy intuitivos	4	0,894
3. La detección de gestos fue muy confiable	3,5	1,049
4. Los controles (sin contar los de movimiento) fueron muy intuitivos dentro de la aplicación	4	0,632
5. Los controles de movimiento fueron muy intuitivos dentro de la aplicación	4,167	0,408

Ítems acerca de la aplicación en general	**Puntaje Promedio**	**Desviación Estándar**
1. La aplicación es fácil de navegar.	4,5	0,548
2. Es fácil de encontrar la información deseada.	4	0,894
3. La aplicación es muy fácil de usar	4,5	0,548
4. Un niño de 10 años aprendería rápidamente cómo utilizar esta aplicación	4,167	0,753
5. Hay mucha consistencia en la aplicación	4,833	0,408
6. Las distintas funcionalidades estan bien integradas	4,833	0,408

En los siguientes párrafos se explican los puntajes y justificaciones entregados por los usuarios dentro de cada parte del cuestionario.

Ítems acerca del tutorial

Los resultados obtenidos en esta parte del cuestionario fueron bastante altos, siendo el más bajo el del primer ítem *El tutorial es una buena guía para empezar a trabajar sobre la aplicación*, con un total de $4,167$ puntos. La razón de este puntaje es que los usuarios encontraron que si bien el tutorial era bastante completo, faltaban algunos detalles. Lo primero que faltaba era información acerca los demás bloques, debido a que solamente se explicaban los bloques de números dentro del tutorial. Otro detalle que también faltaba era una instrucción que tiene que ver con el *modo de creación de bloques*. Cuando los usuarios entran a este modo antes tienen que apuntar el control hacia la pantalla para que se controlen correctamente los Gesture Cursors. Sin embargo, no había tal indicación dentro del tutorial, por lo que había que avisar a los usuarios durante las pruebas, acerca de esta funcionalidad.

En los otros dos ítems de esta parte hubo puntajes promedio muy altos, por lo que los usuarios encontraban que el tutorial era claro y de una extensión razonable. Este último punto es bastante importante, debido a que si el tutorial fuera demasiado largo, podría aburrir a los usuarios, resultando en que lo salten y pierda su utilidad.

Ítems acerca de los controles

En el primer ítem, se obtuvo en promedio un puntaje de 4,5 puntos. Este alto puntaje se debe a que los usuarios sintieron que la mayoría de los gestos eran bastante simples y

distintos entre ellos, lo que hacía que fueran más fáciles de recordar. Además, mencionaron que tener en la pantalla de modo de creación de gestos, la lista con los gestos disponibles, ayuda mucho a recordarlos. También, los bloques que contienen una etiqueta con el gesto con el que se crearon, cumplían esta misma función.

El segundo ítem obtuvo un puntaje promedio de 4 puntos. Este puntaje se debe a que muchos usuarios encontraron que había una relación entre los gestos y los bloques que se creaban. Sin embargo, un aspecto negativo que notaron algunos participantes era que el gesto de Condicional#2, que es mover ambos Joy-con hacia afuera, resultaba poco intuitivo y que sería mejor que fuera al revés al moverlos hacia dentro.

El puntaje promedio más bajo se obtuvo en el tercer ítem, con un total de $3,5$ puntos. Este puntaje fue más bajo en comparación con el resto, debido a problemas que hubieron con la detección de gestos. En casi todas las pruebas, había un problema para detectar el gesto asociado a los bloques de movimiento (que tiene una forma de **L**). En general, cuando los participantes intentaban realizar este gesto, el programa no lo detectaba como un gesto que tuviera disponible. Por lo tanto, los participantes tuvieron que realizar varias veces este gesto para que finalmente el programa lo detectará. Esto no sucedió con los demás gestos, por lo que habría que arreglar la detección de este gesto en particular.

En la detección de gestos había otro problema que impactó a que se realizarán correctamente. Esto fue en los controles de movimiento para controlar los Gesture Cursors. En **algunos** casos, estos se movían un poco sin que el usuario girara sus Joy-Cons. Este movimiento lo tenía solamente uno de los cursores, y afectaba a que el usuario no pudiera controlar a la perfección los Gesture Cursors, causando que fuera más difícil realizar los gestos. Esto finalmente impactó a la percepción de los usuarios de como detectaba los gestos el IDE. La razón de este comportamiento pudo ser un problema del hardware debido a que solamente sucedió con algunos participantes y todos estaban ocupando equipos distintos para realizar las pruebas.

En el cuarto ítem, se obtuvo un puntaje promedio de 4 puntos. En general, los usuarios opinaban que los controles para navegar dentro del IDE, resultaban todos bastante intuitivos con excepción del botón de arrastrar bloques. Este era el botón *Y*, y todos los participantes sentían que en lugar de este botón, debió haber sido el botón *A*, ya que en general este botón se utiliza para interactuar con casi todos los elementos dentro del IDE, excepto con los bloques.

El último ítem de esta parte obtuvo un puntaje promedio de 4,167 puntos. La mayoría de los participantes opinaba que el control de los Gesture Cursors que se realizaba por medio de los controles de movimiento era adecuado. Sin embargo, había un problema con los Gesture Cursors, cuando se entraba al modo de creación de gestos sin apuntar los controles hacia la pantalla. Cuando pasa esto, los cursores no se encuentran bien posicionados en la pantalla, por lo que había que reiniciar las posiciones de los cursores. La única forma de hacer esto era saliendo y volviendo a entrar al modo de creación de gestos. Hacerlo de esta forma no fue muy cómodo para los usuarios.

Ítems acerca de la aplicación en general

El primer ítem de esta parte obtuvo un puntaje promedio de 4,5 puntos. Esto se debe a que los usuarios primero que todo destacaban las diversas formas en las que se podía navegar dentro del IDE. Por ejemplo, en la vista de nuevo proyecto, si se quería entrar al modo pantalla completa, uno podía seleccionar el botón que se encuentra en pantalla con el cursor o presionar la palanca izquierda que era un atajo para esta funcionalidad. También, otra funcionalidad que se destaco en esta área había sido la barra de estado de la vista de nuevo proyecto. Esta barra muestra lo que hacen los botones disponibles dependiendo del contexto, lo que hacía que la navegación por el IDE fuera también más sencilla. Un problema de navegación que notaron los usuarios era que no se podía salir de los menús de bloques, a menos que se creara un bloque. Esto resultaba incómodo cuando estaban buscando algún bloque en específico, ya que tenían que crear un bloque, eliminarlo, y luego volver al modo de creación de gestos para crear bloques.

El segundo ítem obtuvo un puntaje promedio de 4 puntos. Por lo tanto, los participantes estuvieron en general de acuerdo con este punto, pero encontraron un par de falencias. Primero que todo, notaron que cuando querían cambiar la dirección del bloque *Move By*, la vista era muy parecida a cuando querían hacer los gestos para crear bloques, por lo que algunos pensaron que habían ingresado a esta última por error. También había un bloque que resulto un poco difícil de encontrar para algunos usuarios, fueron los bloques para obtener la posición del personaje *X Position* e *Y Position*. Algunos pensaban que este debería encontrarse en los bloques de movimiento en lugar de los de números. Sin embargo, hubo opiniones divididas sobre esta característica, a algunos les gustaba como estaban organizados los bloques dentro del IDE y otros no.

El tercer ítem obtuvo un puntaje promedio de 4,5 puntos. Este puntaje se debe a que, en general, los usuarios encontraron que el IDE era sencillo y no habían complejidades innecesarias.

El cuarto ítem obtuvo un puntaje promedio de 4,167 puntos. Este puntaje se relaciona mucho con el del ítem anterior, que es el de la aplicación es muy fácil de usar. Debido a que la mayoría estaba de acuerdo con esta afirmación, sentían que no debería ser mucho más complicado para un niño aprender a utilizar este IDE. Algunos participantes sentían que el IDE actualmente podría ser un poco intimidante para niños que nunca antes habían programado. Por esto, sugerían que si se expandía el tutorial, o si se agregaba algún modo que permita al usuario aprender más acerca de como se podían utilizar los bloques, podría resultar en un IDE mucho más amigable y completo.

El quinto ítem obtuvo un puntaje promedio de 4,833 puntos. Este puntaje se debe a que los usuarios sentían que había una gran consistencia en el uso de colores y formas dentro del IDE. Por ejemplo, el dividir en distintos colores las distintas categorías de bloques, resultaba bastante útil para recordarlas. Otro aspecto que destacaban es como se relacionan las formas de algunos bloques con el gesto al que tienen asociados, como es en el caso de los bloques de números o booleanos.

Finalmente, el último ítem obtuvo un puntaje promedio de 4,833 puntos. Este puntaje se debe a que los usuarios en general estaban satisfechos con el funcionamiento del IDE.

5.3. Discusión

A partir de los resultados anteriores se obtienen las siguientes conclusiones con respecto a el estado actual del IDE. Primero que todo, a partir de los altos puntajes obtenidos y comentarios en los ítems 1 y 2 de la sección *ítems acerca de los controles*, se puede concluir que el uso de gestos dentro del IDE es considerado usable, al menos dentro de la muestra en que se realizo la prueba.

En el caso de la funcionalidad en la detección de gestos, fue buena con la excepción de uno de los gestos que era difícil de detectar por parte del IDE. Este gesto es el que esta asociado a los bloques de movimiento, y hay que corregirlo para una futura versión.

Por otro lado se puede concluir que la funcionalidad de los controles de movimiento también fue buena, aunque en algunos casos no funcionó a la perfección. Esto tuvo como consecuencia que en algunas pruebas los usuarios no controlaran bien el movimiento de los Gesture Cursors dentro del IDE. Este problema se puede deber al hardware utilizado por los participantes, ya que cada prueba se realizó con un computador y Joy-Cons distintos. Por lo tanto, hay que identificar cual fue el causante de este problema y como se puede arreglar.

Con respecto al manejo de bloques dentro del IDE, se puede concluir que se considero usable y funcional para los participantes. Esto se ve reflejado en que no hubo críticas por parte de los participantes con respecto a los bloques dentro del IDE. La única crítica negativa que existió fue sobre el botón con el que se arrastraban los bloques, que es algo que se puede cambiar.

Finalmente, el IDE en general su usabilidad fue bien evaluada. Esto se ve reflejado en los altos puntajes asignados y comentarios hechos por los usuarios en los ítems de la sección *ítems acerca de aplicación en general* del cuestionario. Incluso destacaron la sencillez del IDE y que es entretenido.

Es importante recordar que todas estas conclusiones se obtienen sobre una prueba que fue realizada sobre un pequeño número de usuarios (6 en total). Además, estos usuarios no eran los usuarios objetivo de este IDE. No obstante lo anterior, dado el éxito de los resultados, se podría considerar que el estado del IDE es funcional y usable para usuarios que tengan características similares a la muestra. Por lo tanto, habría que en un futuro diseñar una prueba que tenga un tamaño de muestra mayor al actual y que cuente con los usuarios objetivos, para probar la usabilidad del IDE con este tipo de usuarios.

5.4. Posibles mejoras a incorporar en el IDE

Las mejoras que hay que hacer al IDE, tomando en cuenta la retroalimentación de los participantes, son las siguientes:

1. Mejorar la detección del gesto asociado a los bloques de movimiento, que es el que tiene forma de **L**. Esto se puede realizar de muchas formas, siendo una disminuir el porcentaje de aceptación de la detección de este gesto en particular. Otra alternativa podría ser cambiar el gesto por otro.

2. Cambiar el botón para arrastrar los bloques, reemplazando el botón Y por el A.
3. Agregar a los menús de bloques un botón que permita salir de este menú sin que los usuarios tengan que crear un bloque.
4. Agregar un botón en el modo de creación de gestos que permita centrar los Gesture Cursors en la pantalla.
5. Identificar el problema que causa que en algunos equipos, los Gesture Cursors se muevan sin que el usuario mueva sus Joy-Con. Para esto hay que probar con distintos computadores y controles el IDE, y buscar cual puede ser el causante de este problema.
6. Expandir el tutorial o añadir algún modo que permita obtener más información acerca de lo que se puede hacer con los bloques dentro del IDE.

Como se mencionó anteriormente, una meta para el futuro es la de diseñar y realizar una prueba en donde los participantes sean los usuarios objetivo, para probar la usabilidad del IDE con este tipo de usuarios. Por esta razón sería ideal que algunos de los cambios mencionados se realizarán antes de tomar la prueba, para así tener una mejor versión del IDE.

Capítulo 6

Conclusiones y Trabajo Futuro

A lo largo del labor realizado en este trabajo de memoria, se puede notar cómo el desarrollo de un software tiene que ser un proceso iterativo para lograr una solución que sea funcional y usable. Esto se ve reflejado con las actualizaciones que se fueron haciendo a los prototipos de este proyecto en función a los experimentos que se iban realizando, que permitieron llegar a una versión final del IDE.

A continuación se retoman los objetivos específicos propuestos, además de una discusión acerca del cumplimiento para cada uno de estos:

1. **Diseñar, siguiendo una estrategia de diseño participativo, una biblioteca de metáforas de interacción natural que sean consideradas aceptables por niños de entre 10 y 12 años:** Se determinó una biblioteca de metáforas de interacción natural la cual su usabilidad fue evaluada y comprobada por un grupo de usuarios expertos.
2. **Diseñar un prototipo de entorno de desarrollo integrado que pueda ser usado por niños de entre 10 y 12 años para crear aplicaciones de software utilizando la biblioteca de metáforas de interacción natural propuesta:** Se diseñó un prototipo el cual fue evaluado por un grupo de usuarios expertos, y posteriormente ajustado en base a su retroalimentación.
3. **Desarrollar una interfaz de software que permita mapear metáforas de interacción natural con un prototipo de entorno de desarrollo integrado de aplicaciones, de tal manera que la interacción sea considerada usable por niños de entre 10 y 12 años:** Se desarrolló una interfaz de software con estas características, la cual fue evaluada por un pequeño grupo de usuarios que no eran los objetivo, en donde se comprobó la funcionalidad correcta del IDE. También se comprobó en parte la usabilidad de esta.

Tomando en cuenta que todos los objetivos anteriores fueron alcanzados con excepción del último que se encuentra incompleto, el objetivo principal de este trabajo de título se cumplió casi en su totalidad. Esta falta se debe a que la prueba de concepto que se realizó con la implementación del IDE, no fue hecha sobre los usuarios objetivo del trabajo, por lo que no se pudo comprobar su completa usabilidad. Sin embargo, se pudo validar la funcionalidad

del IDE, con lo que se pudo establecer una base para un trabajo futuro.

Para el futuro de este trabajo, lo primero que se podría hacer es aplicar al IDE las posibles mejoras que fueron obtenidas a partir de los resultados de la prueba de concepto. Estas fueron listadas en el capítulo anterior, y pueden mejorar la usabilidad que tiene actualmente el IDE.

Otra meta para el futuro es diseñar y realizar una segunda prueba de concepto en la que solamente participen los usuarios objetivo. Con esta, se podría comprobar en su totalidad la usabilidad que tiene el IDE, al igual que encontrar nuevas fortalezas y debilidades de él.

Por otra parte, cabe recordar que dentro del IDE hay componentes que quedaron pendientes por desarrollar. Estas son los bloques de ciclos, los bloques de variables, el método de borrado de bloques a través del basurero y la optimización del lenguaje del IDE. Para implementar los bloques de ciclo y de variable, se puede empezar por extender las clases ya existentes que representan las formas de bloques. En el caso del método de borrado, se puede desarrollar utilizando el sistema de colisiones de Unity, con el cual se puede detectar si un bloque se encuentra cerca de la posición del basurero. Finalmente, para optimizar el lenguaje del IDE, se puede utilizar el método del trampolín mencionado en el capitulo 4. Por lo tanto, también queda como trabajo futuro implementar todas estas funcionalidades.

Finalmente, uno de los aspectos más importantes que motivó a esta memoria fue la de desarrollar una herramienta alternativa a las existentes para enseñar el pensamiento computacional a niños, que sea mejor o que sea una alternativa a las que existen actualmente. Es por esta razón que resulta vital comprobar también el nivel de impacto que tiene este IDE en el aprendizaje de esta habilidad. Por lo tanto, en un futuro se debería diseñar y realizar un experimento que permita verificar el punto anterior.

Bibliografía

[1] Al-Akash, I. Nintendo da vinci: Implementing a novel control system to improve performance in robotic assisted surgery – a pilot study. *PeerJ* (04 2019), 10.

[2] Bocconi, S., Chioccariello, A., Dettori, G., Ferrari, A., Engelhardt, K., Kampylis, P., and Punie, Y. Developing computational thinking in compulsory education. implications for policy and practice. *EUR - Scientific and Technical Research Reports* (12 2016).

[3] Brackmann, C. P., Román-González, M., Robles, G., Moreno-León, J., Casali, A., and Barone, D. Development of computational thinking skills through unplugged activities in primary school. In *Proceedings of the 12th Workshop on Primary and Secondary Computing Education* (New York, NY, USA, 2017), WiPSCE '17, Association for Computing Machinery, p. 65–72.

[4] Brooke, J. Sus: A quick and dirty usability scale. *Usability Eval. Ind. 189* (11 1995).

[5] CODE. *About | Code.org*, [consulta: 04 mayo 2020].

[6] Constantinou, V., and Ioannou, A. Development of computational thinking skills through educational robotics. In *EC-TEL* (2018), p. 8.

[7] Department of Education UK. *National curriculum in England: computing programmes of study*, 11 de septiembre, 2013. [consulta: 04 mayo 2020]. `https://www.gov.uk/government/publications/national-curriculum-in-england-computing-programmes-of-study/national-curriculum-in-england-computing-programmes-of-study`.

[8] Edwards, S. New concepts of play and the problem of technology, digital media and popular-culture integration with play-based learning in early childhood education. *Technology, Pedagogy and Education 25*, 4 (2016), 513–532.

[9] Hewett, T. T., Baecker, R., Card, S., Carey, T., Gasen, J., Mantei, M., Perlman, G., Strong, G., and Verplank, W. Acm sigchi curricula for human-computer interaction. Tech. rep., ACM SIGCHI, New York, NY, USA, 1992.

[10] Kelley, J. F. An iterative design methodology for user-friendly natural language office information applications. *ACM Trans. Inf. Syst. 2*, 1 (Jan. 1984), 26–41.

[11] Mineduc. *Plan Nacional de Lenguajes Digitales - Mineduc*, [consulta: 4 mayo 2020]. http://sitios.mineduc.cl/lenguajesdigitales/.

[12] Nielsen, J. *Usability Engineering*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1994.

[13] Palts, T., and Pedaste, M. A model for developing computational thinking skills. *Informatics in Education 19* (03 2020), 113–128.

[14] Resnick, M., Maloney, J., Monroy-Hernández, A., Rusk, N., Eastmond, E., Brennan, K., Millner, A., Rosenbaum, E., Silver, J., Silverman, B., and Kafai, Y. Scratch: Programming for all. *Commun. ACM 52*, 11 (Nov. 2009), 60–67.

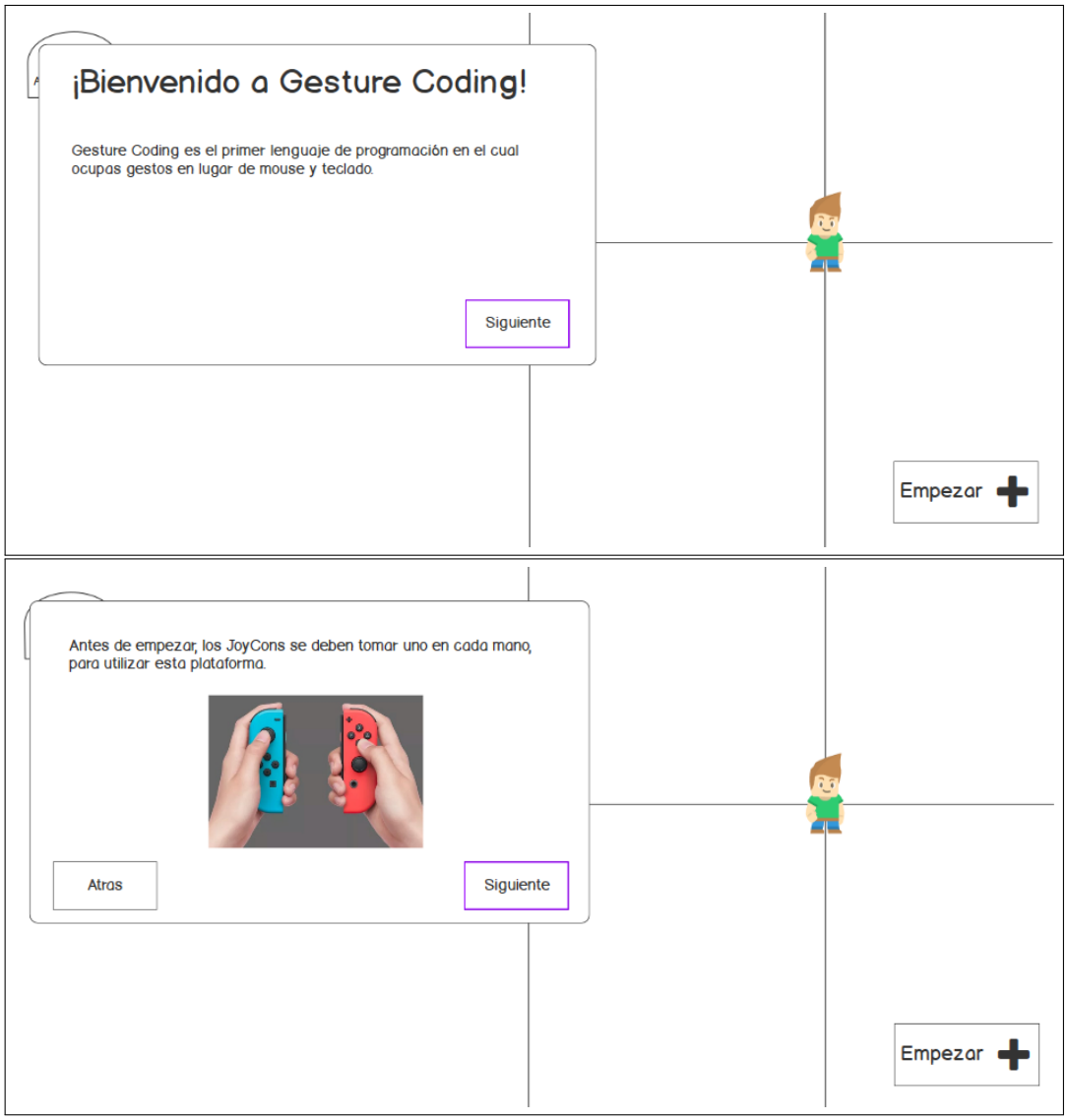
[15] Rich, K. M., Strickland, C., Binkowski, T. A., Moran, C., and Franklin, D. K-8 learning trajectories derived from research literature: Sequence, repetition, conditionals. In *Proceedings of the 2017 ACM Conference on International Computing Education Research* (New York, NY, USA, 2017), ICER '17, Association for Computing Machinery, p. 182–190.

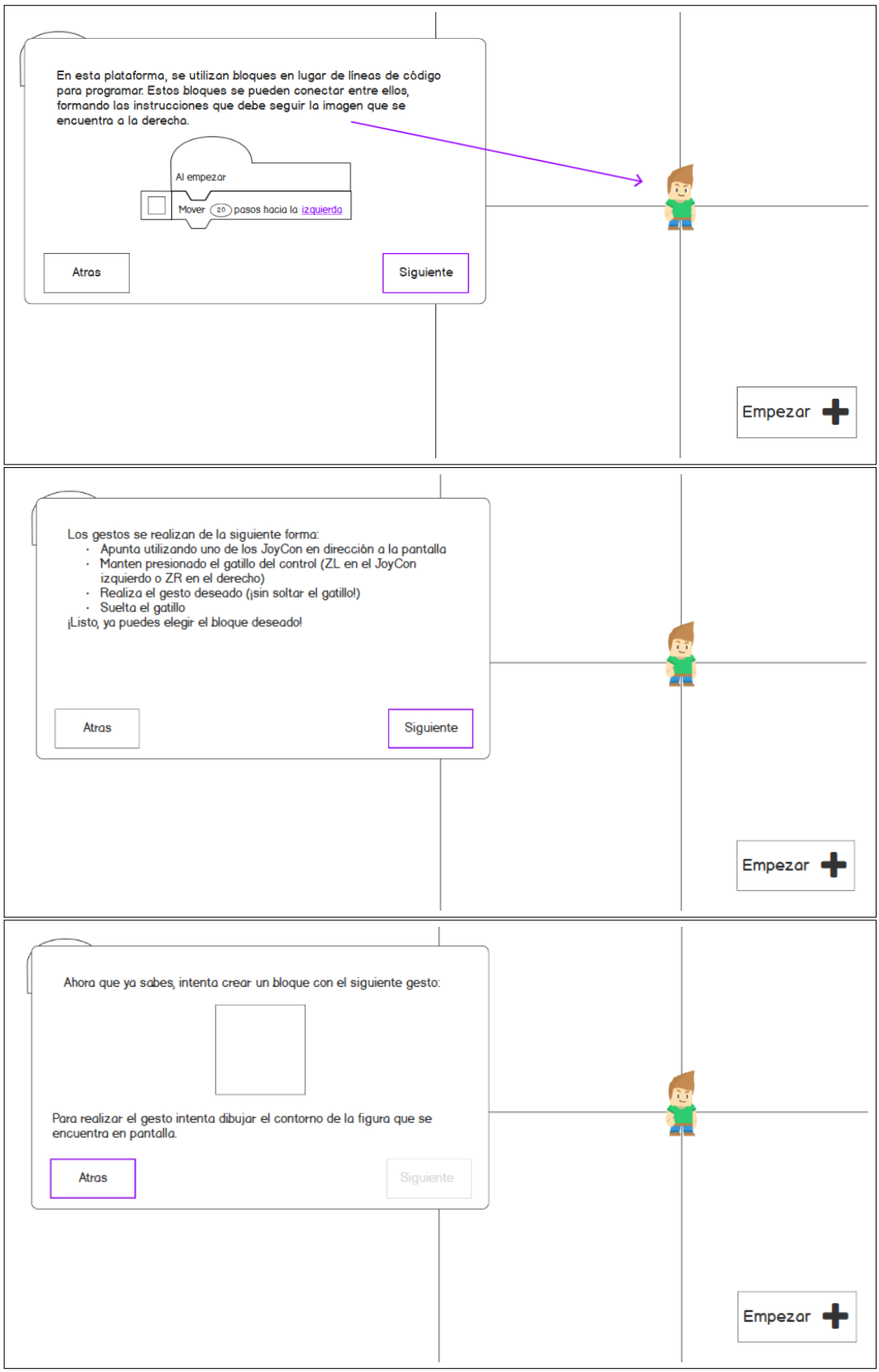
[16] Saritepeci, M., and Yildiz-Durak, H. *Analyzing the Effect of Block and Robotic Coding Activities on Computational Thinking in Programming Education*. St. Kliment Ohridski University, 05 2017, pp. 464–473.

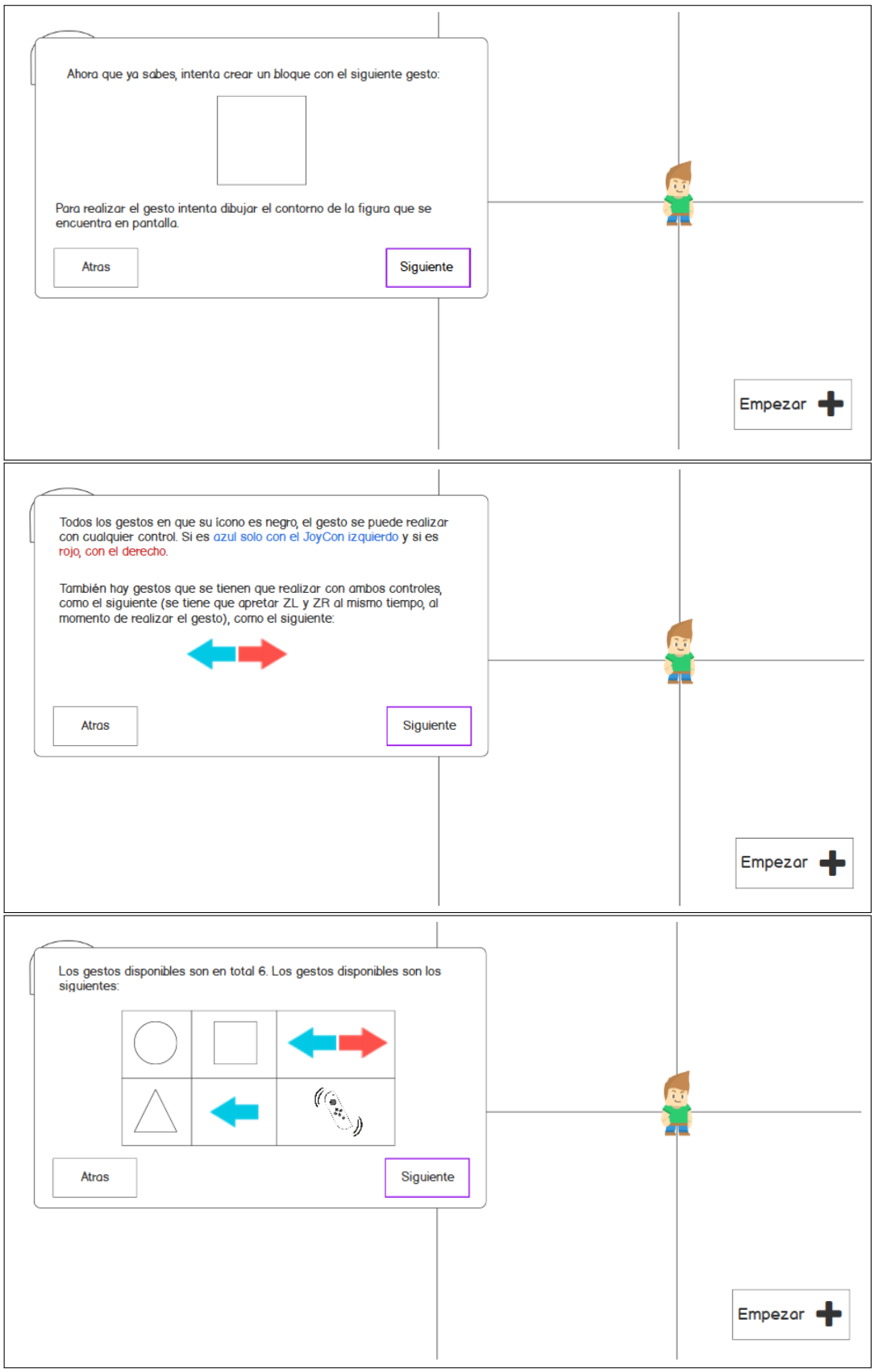
[17] Wing, J. M. Research notebook: Computational thinking: What and why? In *The Link Magazine* (2011), pp. 20–23.

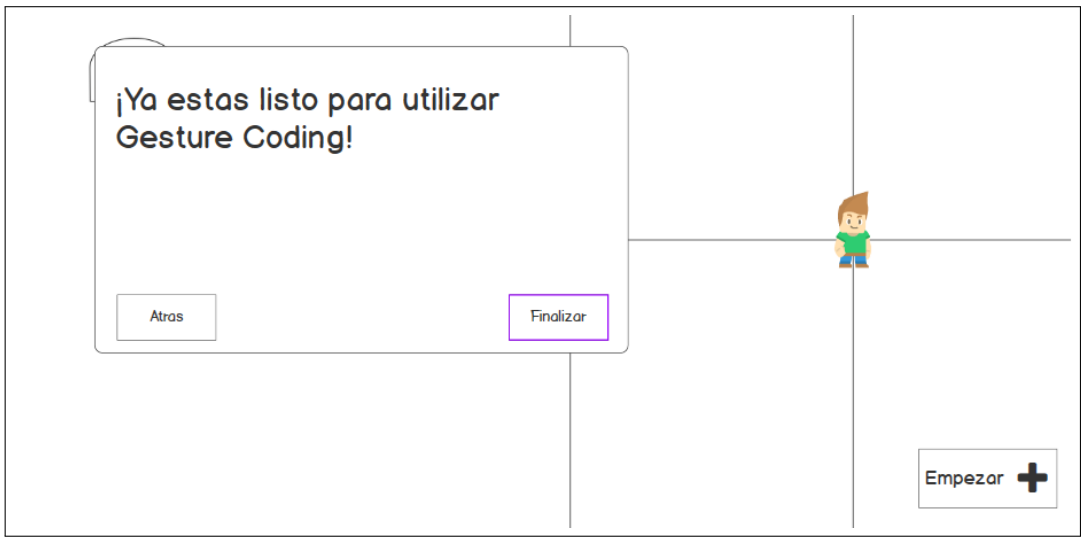
Apéndices

Apéndice A: Tutorial prototipo I



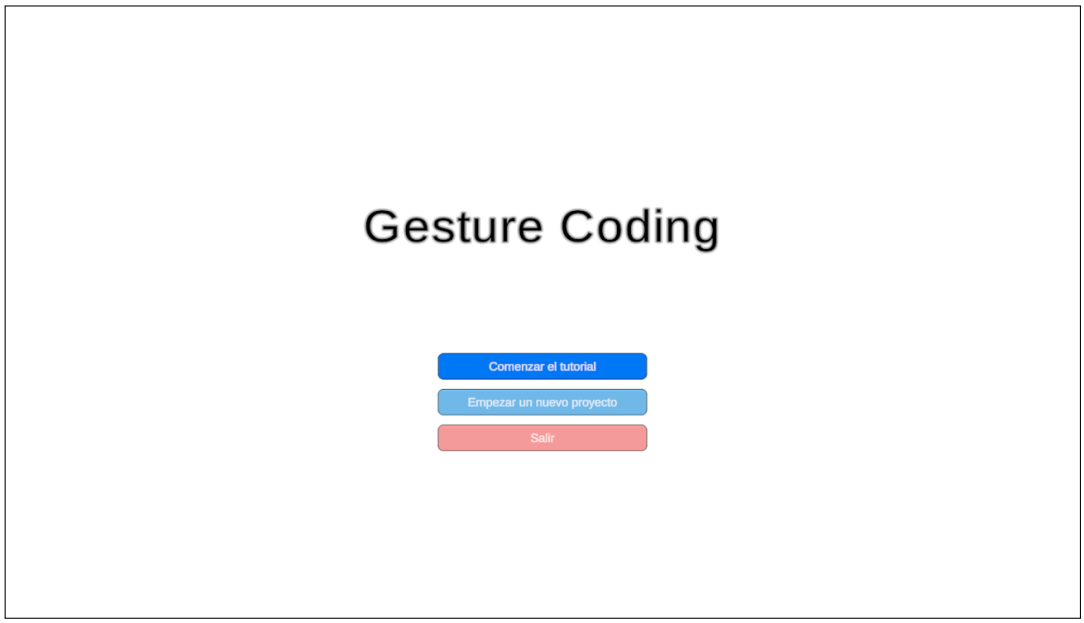

En esta plataforma, se utilizan bloques en lugar de líneas de código para programar. Estos bloques se pueden conectar entre ellos, formando las instrucciones que debe seguir la imagen que se encuentra a la derecha.
Al empezar
Mover 20 pasos hacia la izquierda
Atras
Siguiente
Empezar
Los gestos se realizan de la siguiente forma:
Apunta utilizando uno de los JoyCon en dirección a la pantalla
Manten presionado el gatillo del control (ZL en el JoyCon izquierdo o ZR en el derecho)
Realiza el gesto deseado (¡sin soltar el gatillo!)
Suelta el gatillo
¡Listo, ya puedes elegir el bloque deseado!
Atras
Siguiente
Empezar
Ahora que ya sabes, intenta crear un bloque con el siguiente gesto:
Para realizar el gesto intenta dibujar el contorno de la figura que se encuentra en pantalla.
Atras
Siguiente
Empezar


Ahora que ya sabes, intenta crear un bloque con el siguiente gesto:
Para realizar el gesto intenta dibujar el contorno de la figura que se encuentra en pantalla.
Atras
Siguiente
Empezar
Todos los gestos en que su ícono es negro, el gesto se puede realizar con cualquier control. Si es azul solo con el JoyCon izquierdo y si es rojo, con el derecho.
También hay gestos que se tienen que realizar con ambos controles, como el siguiente (se tiene que apretar ZL y ZR al mismo tiempo, al momento de realizar el gesto), como el siguiente:
Atras
Siguiente
Empezar
Los gestos disponibles son en total 6. Los gestos disponibles son los siguientes:
Atras
Siguiente
Empezar

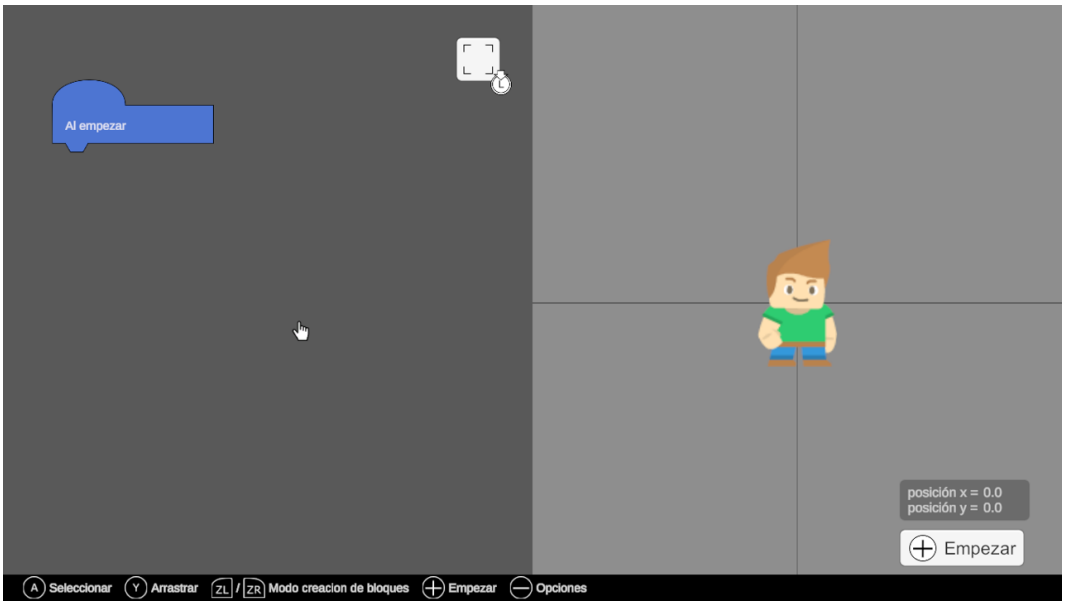


Apéndice B: Interfaces y objetos del IDE implementado

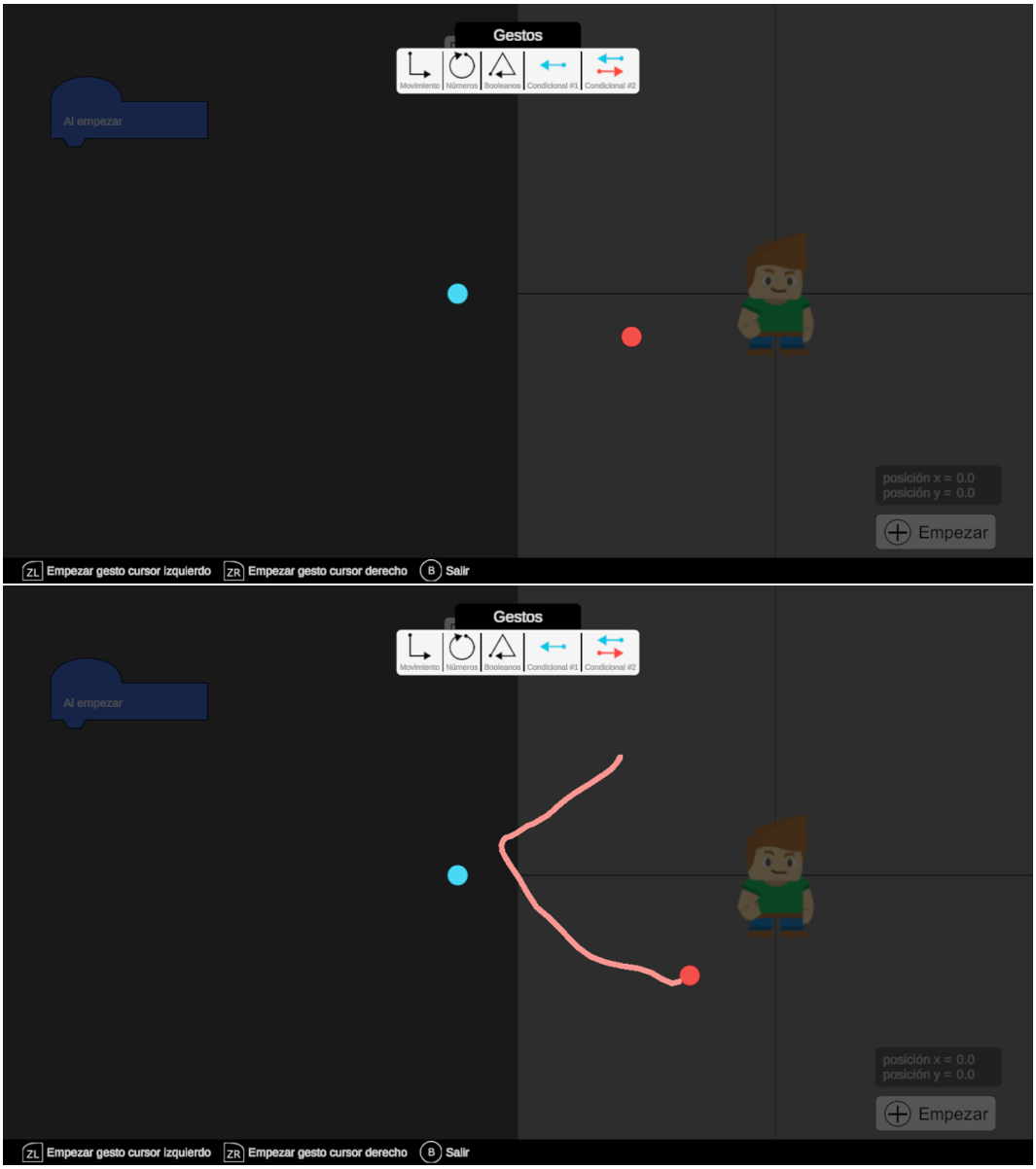
Vista Inicial



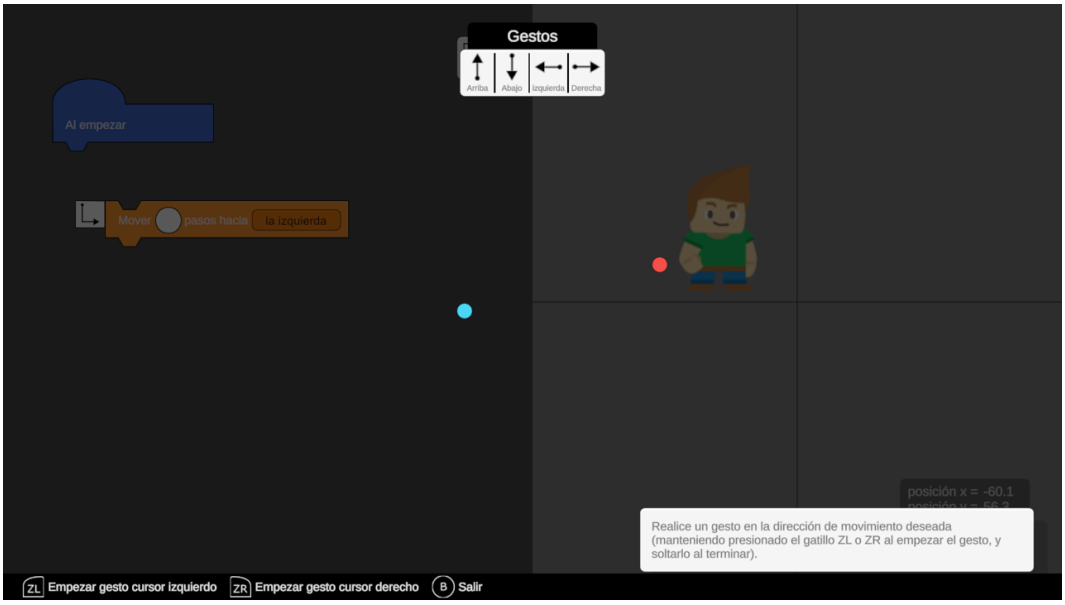
Vista Nuevo Proyecto



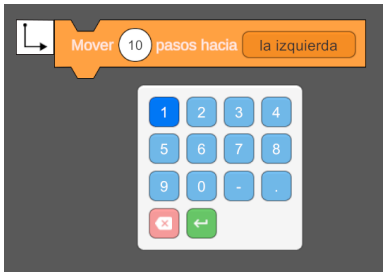
Modo creación de gestos - Crear bloques



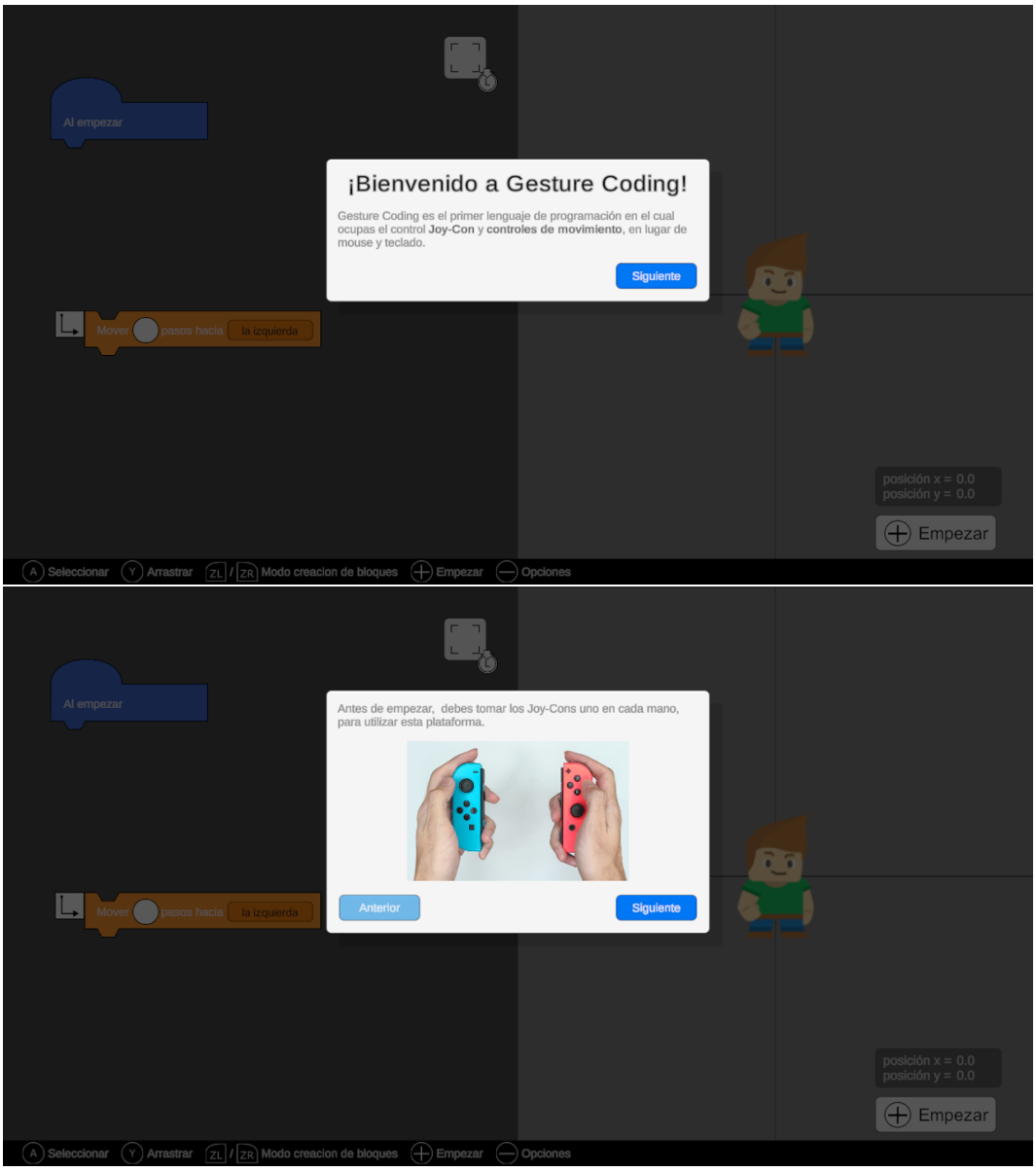
Modo creación de gestos - Asignar dirección al bloque MoveByBlock

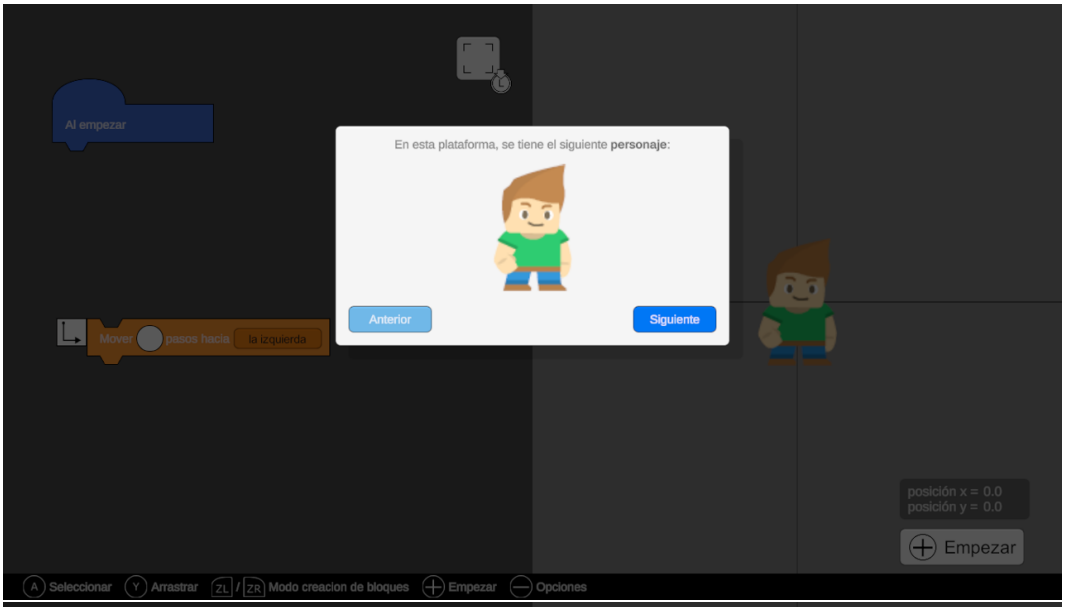


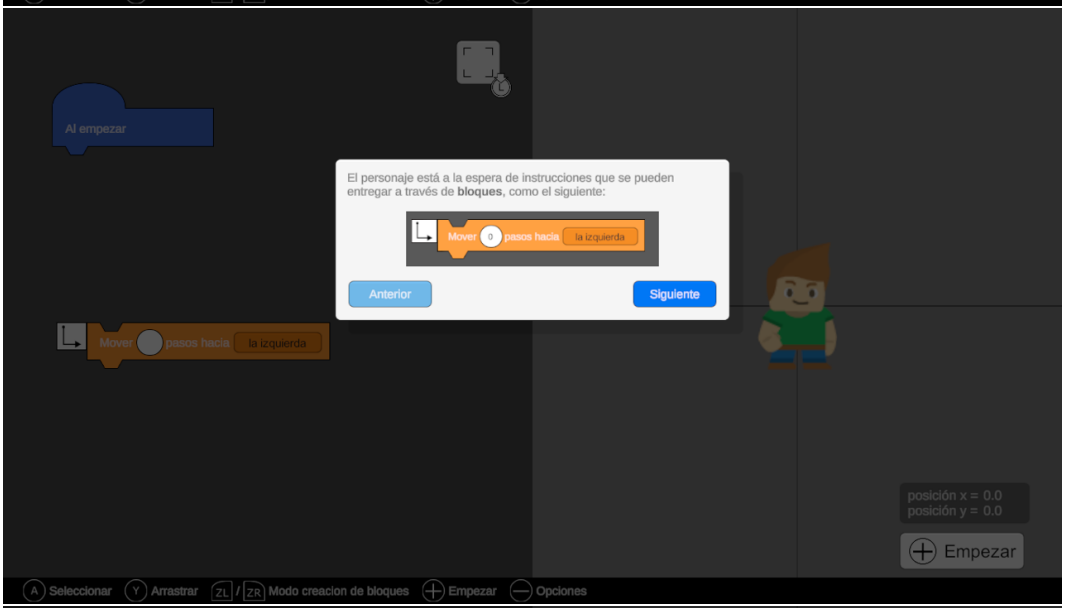
Teclado

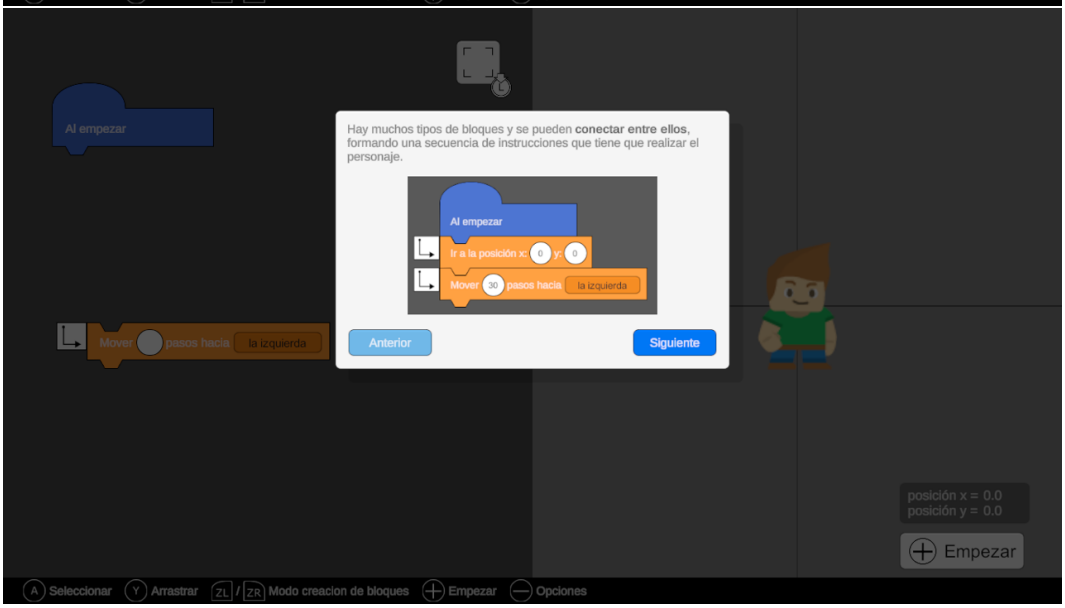


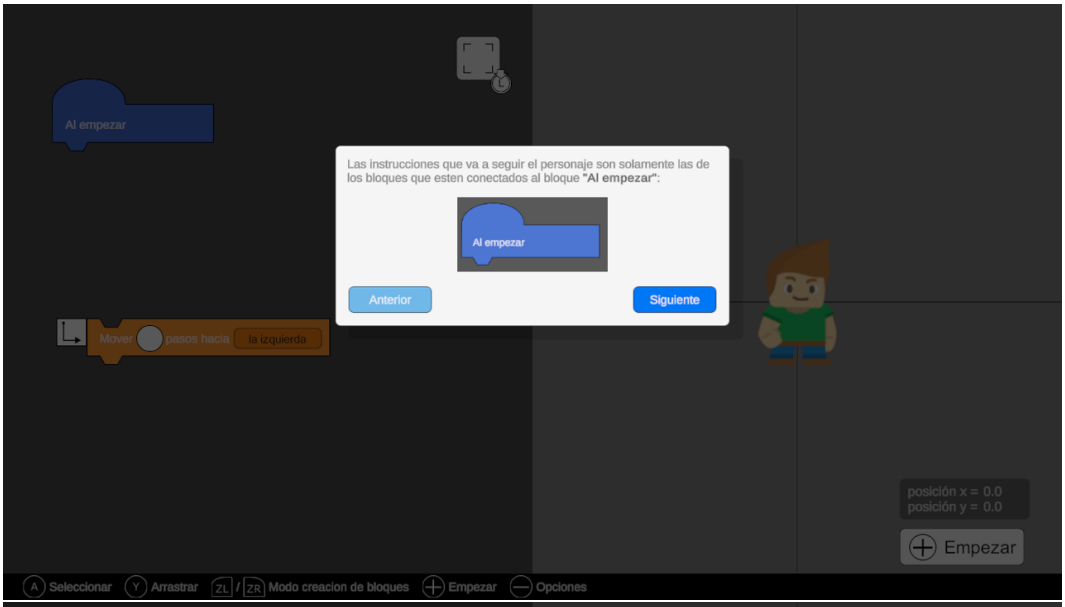
Apéndice C: Tutorial del IDE implementado

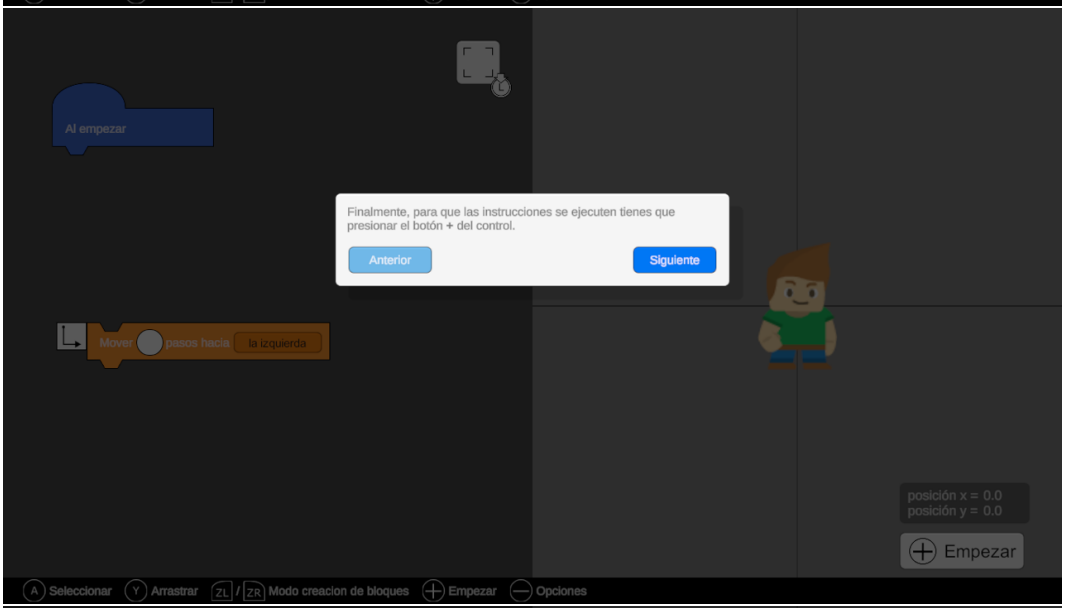


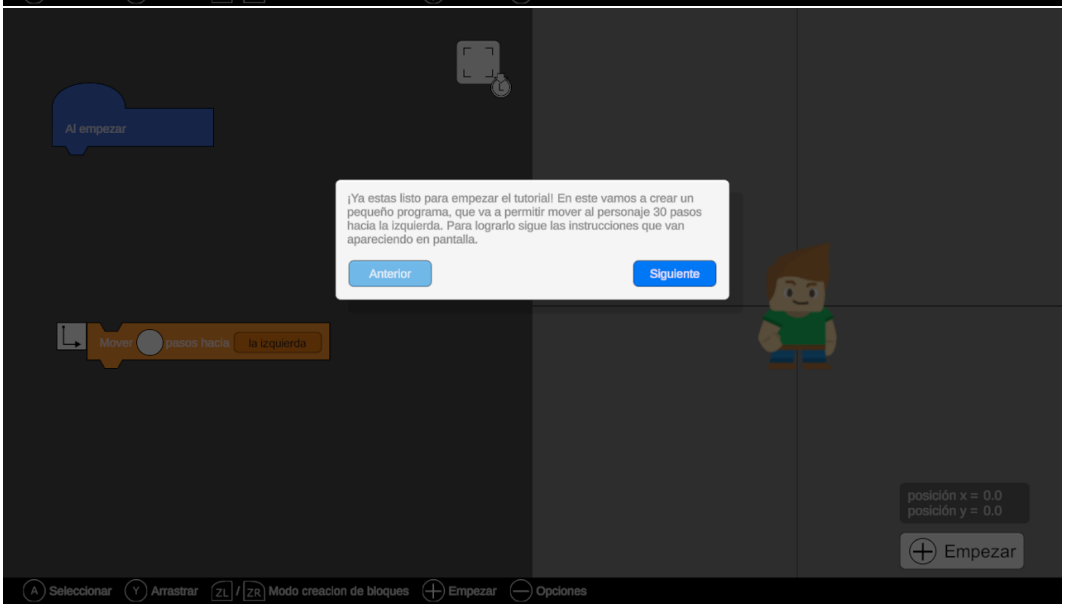

Al empezar
En esta plataforma, se tiene el siguiente **personaje**:
Anterior
Siguiente
Mover pasos hacia la izquierda
posición x = 0.0
posición y = 0.0
Empezar
A Seleccionar Y Arrastrar ZL / ZR Modo creacion de bloques Empezar Opciones

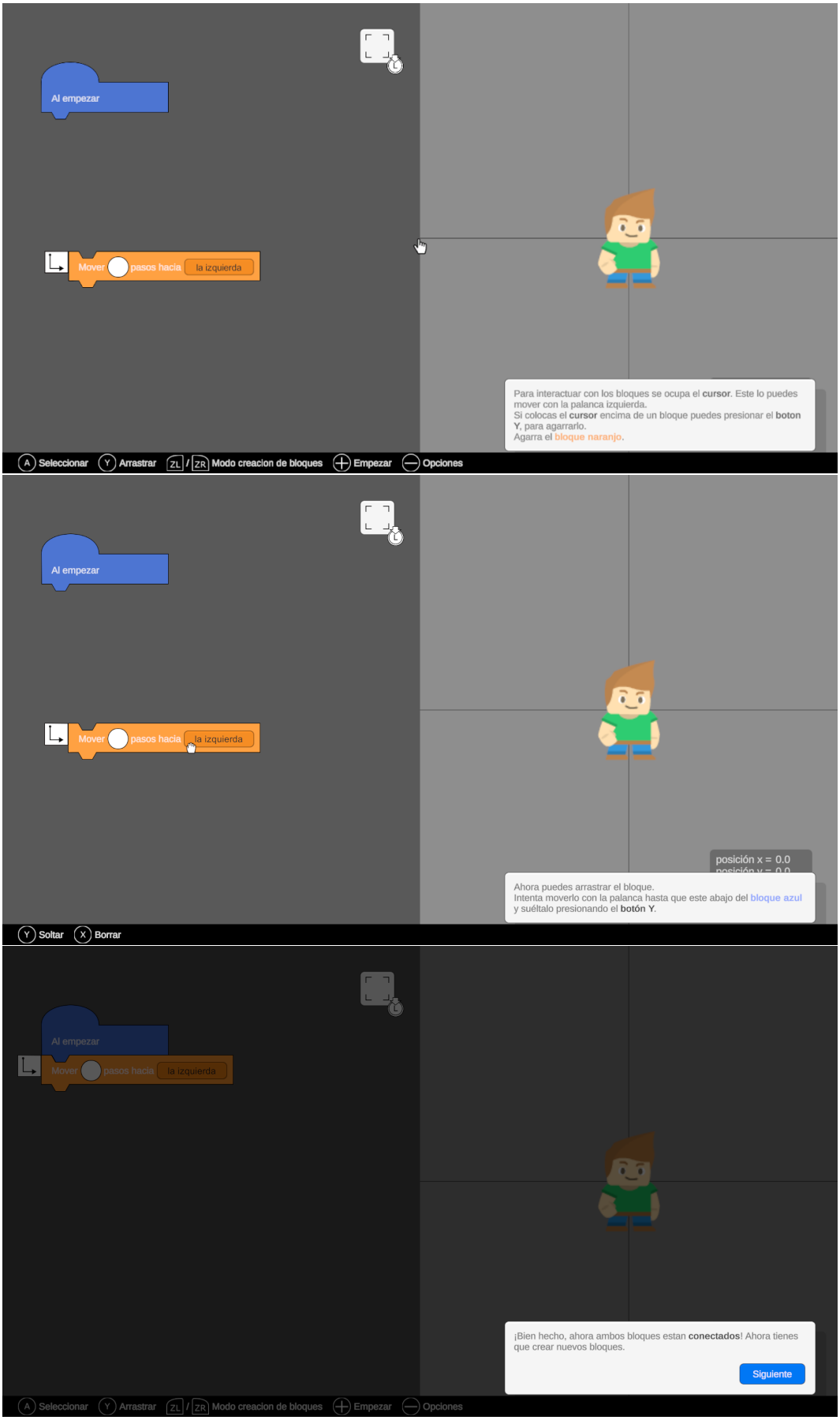

Al empezar
El personaje está a la espera de instrucciones que se pueden entregar a través de **bloques**, como el siguiente:
Mover 0 pasos hacia la izquierda
Anterior
Siguiente
Mover pasos hacia la izquierda
posición x = 0.0
posición y = 0.0
Empezar
A Seleccionar Y Arrastrar ZL / ZR Modo creacion de bloques Empezar Opciones

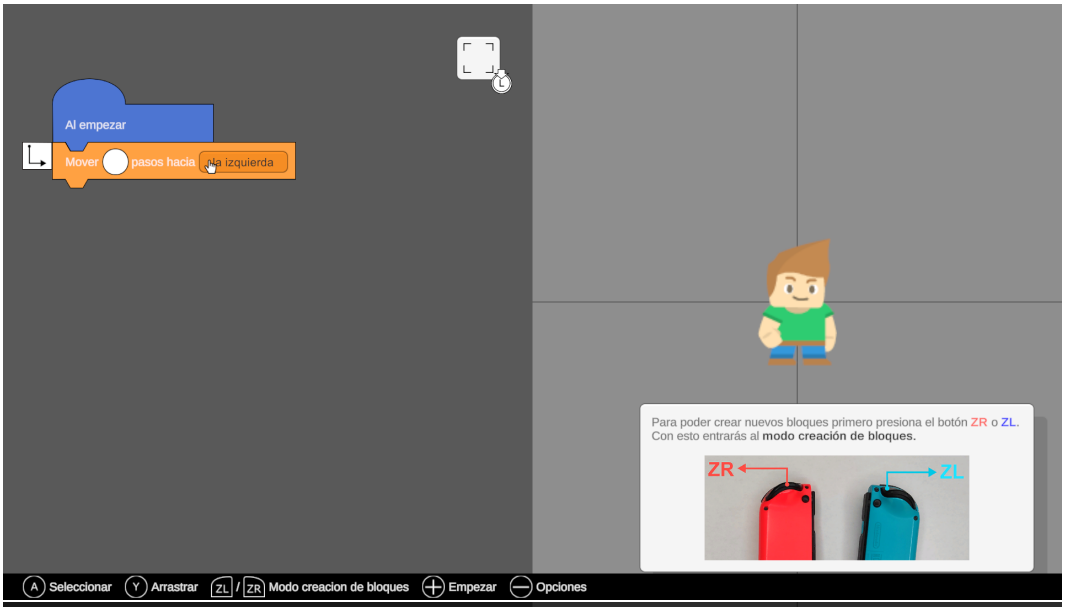

Al empezar
Hay muchos tipos de bloques y se pueden **conectar entre ellos**, formando una secuencia de instrucciones que tiene que realizar el personaje.
Al empezar
Ir a la posición x: 0 y: 0
Mover 30 pasos hacia la izquierda
Anterior
Siguiente
Mover pasos hacia la izquierda
posición x = 0.0
posición y = 0.0
Empezar
A Seleccionar Y Arrastrar ZL / ZR Modo creacion de bloques Empezar Opciones

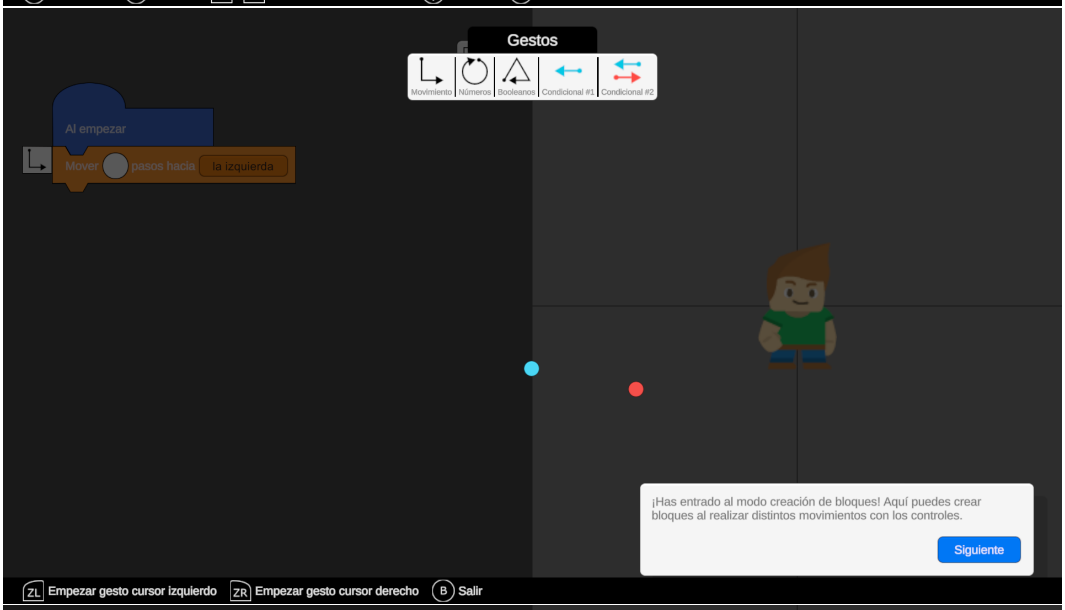

Al empezar
Las instrucciones que va a seguir el personaje son solamente las de los bloques que esten conectados al bloque "Al empezar":
Al empezar
Anterior
Siguiente
Mover pasos hacia la izquierda
posición x = 0.0
posición y = 0.0
Empezar
A Seleccionar Y Arrastrar ZL / ZR Modo creacion de bloques + Empezar − Opciones

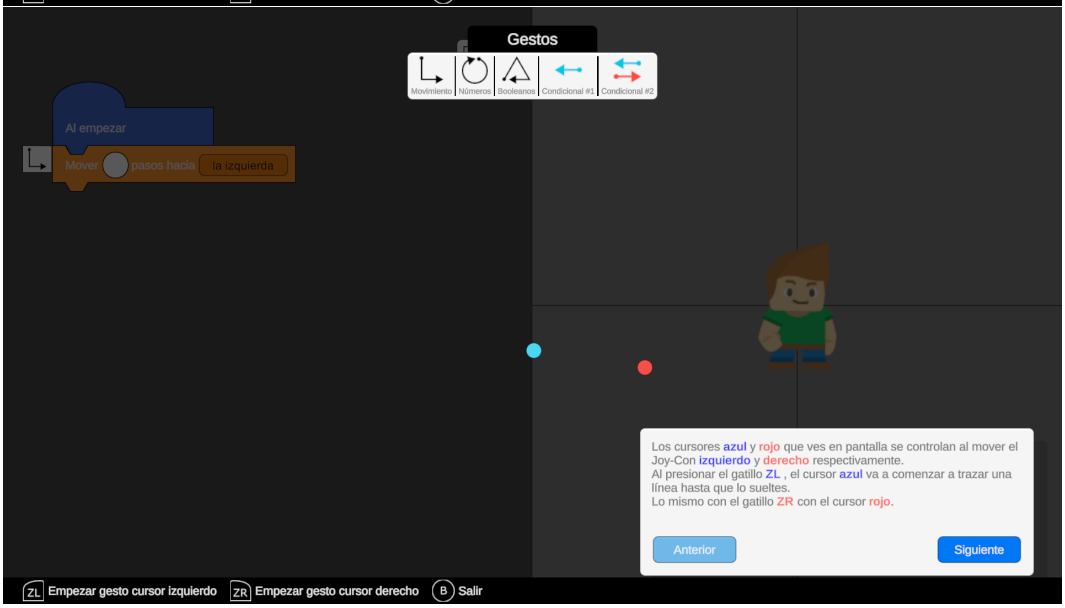

Al empezar
Finalmente, para que las instrucciones se ejecuten tienes que presionar el botón + del control.
Anterior
Siguiente
Mover pasos hacia la izquierda
posición x = 0.0
posición y = 0.0
Empezar
A Seleccionar Y Arrastrar ZL / ZR Modo creacion de bloques + Empezar − Opciones

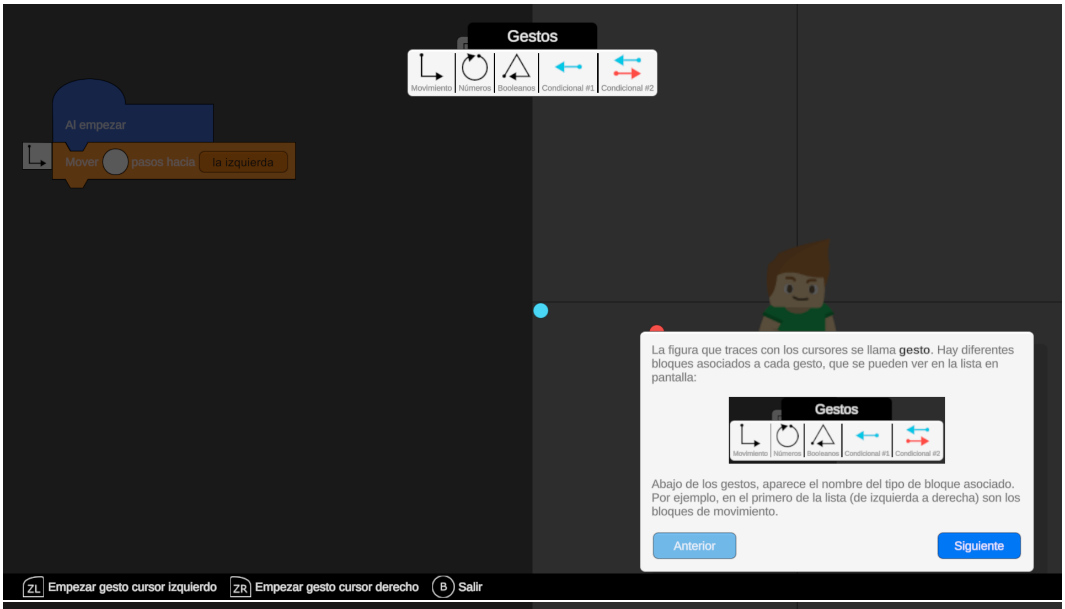

Al empezar
¡Ya estas listo para empezar el tutorial! En este vamos a crear un pequeño programa, que va a permitir mover al personaje 30 pasos hacia la izquierda. Para lograrlo sigue las instrucciones que van apareciendo en pantalla.
Anterior
Siguiente
Mover pasos hacia la izquierda
posición x = 0.0
posición y = 0.0
Empezar
A Seleccionar Y Arrastrar ZL / ZR Modo creacion de bloques + Empezar − Opciones

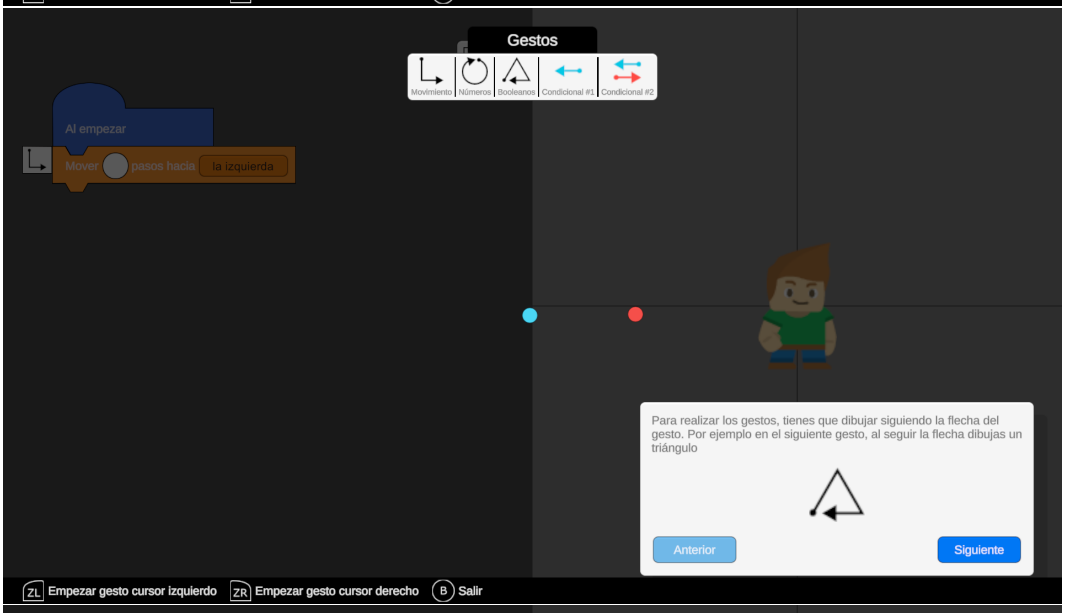

Al empezar
Mover pasos hacia la izquierda
Para interactuar con los bloques se ocupa el cursor. Este lo puedes mover con la palanca izquierda.
Si colocas el cursor encima de un bloque puedes presionar el boton Y, para agarrarlo.
Agarra el bloque naranjo.
A Seleccionar
Y Arrastrar
ZL / ZR Modo creacion de bloques
Empezar
Opciones
Al empezar
Mover pasos hacia la izquierda
posición x = 0.0
Ahora puedes arrastrar el bloque.
Intenta moverlo con la palanca hasta que este abajo del bloque azul y suéltalo presionando el botón Y.
Y Soltar
X Borrar
Al empezar
Mover pasos hacia la izquierda
¡Bien hecho, ahora ambos bloques estan conectados! Ahora tienes que crear nuevos bloques.
Siguiente
A Seleccionar
Y Arrastrar
ZL / ZR Modo creacion de bloques
Empezar
Opciones

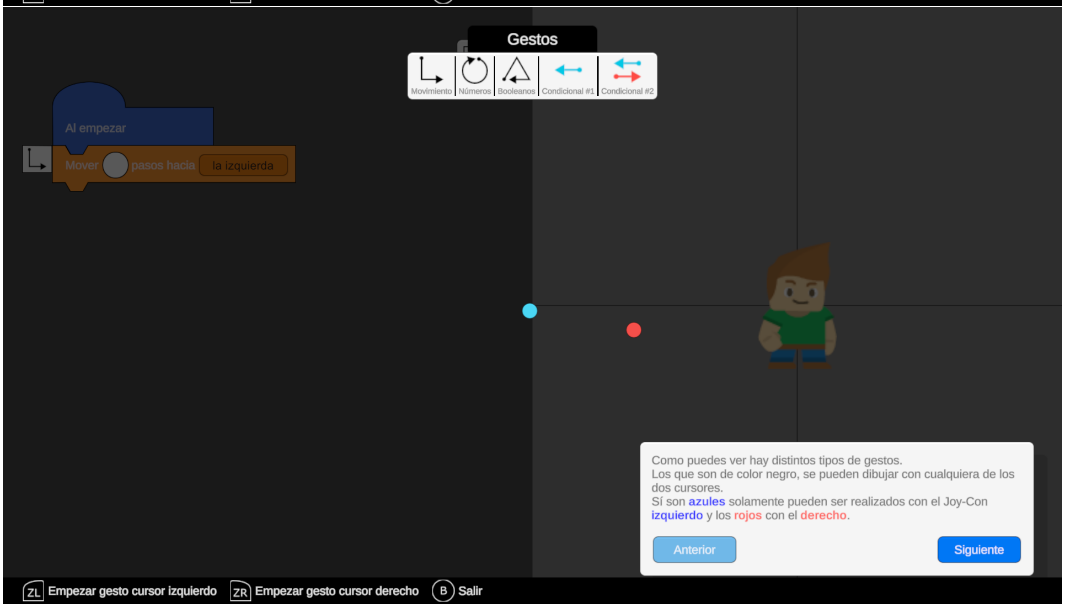

Al empezar
Mover pasos hacia la izquierda
Para poder crear nuevos bloques primero presiona el botón ZR o ZL.
Con esto entrarás al modo creación de bloques.
ZR
ZL
A Seleccionar
Y Arrastrar
ZL / ZR Modo creacion de bloques
Empezar
Opciones

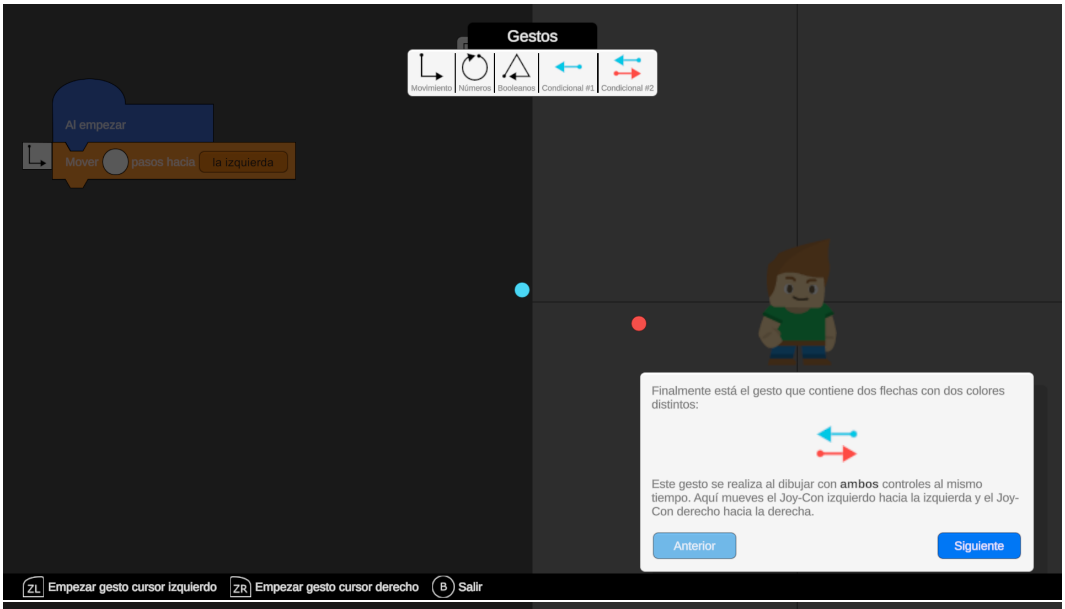

Gestos
Movimiento
Números
Booleanos
Condicional #1
Condicional #2
Al empezar
Mover pasos hacia la izquierda
¡Has entrado al modo creación de bloques! Aquí puedes crear bloques al realizar distintos movimientos con los controles.
Siguiente
ZL Empezar gesto cursor izquierdo
ZR Empezar gesto cursor derecho
B Salir

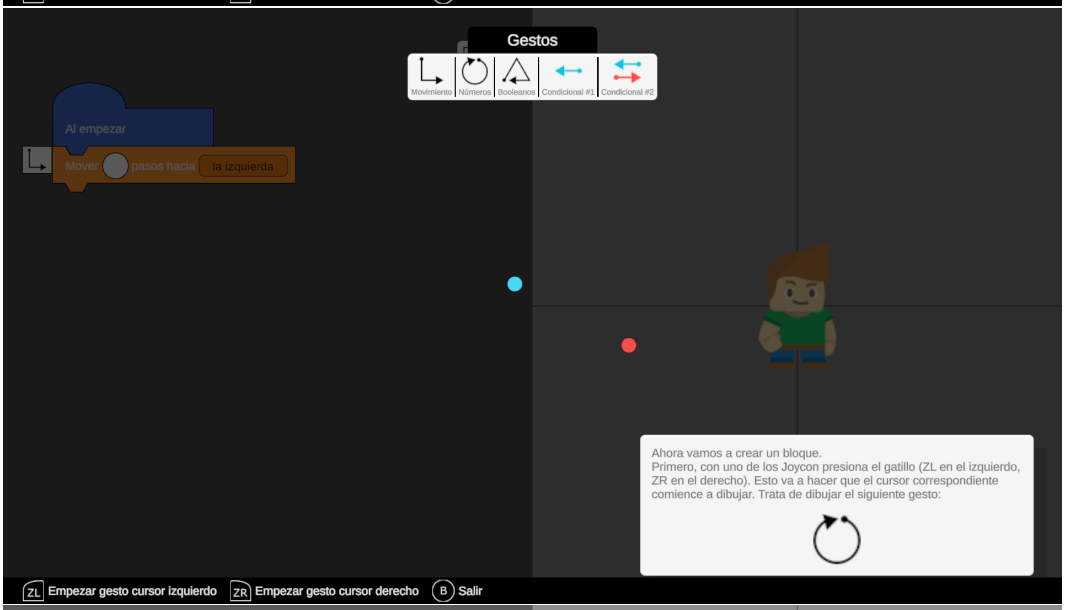

Gestos
Movimiento
Números
Booleanos
Condicional #1
Condicional #2
Al empezar
Mover pasos hacia la izquierda
Los cursores azul y rojo que ves en pantalla se controlan al mover el Joy-Con izquierdo y derecho respectivamente.
Al presionar el gatillo ZL , el cursor azul va a comenzar a trazar una línea hasta que lo sueltes.
Lo mismo con el gatillo ZR con el cursor rojo.
Anterior
Siguiente
ZL Empezar gesto cursor izquierdo
ZR Empezar gesto cursor derecho
B Salir

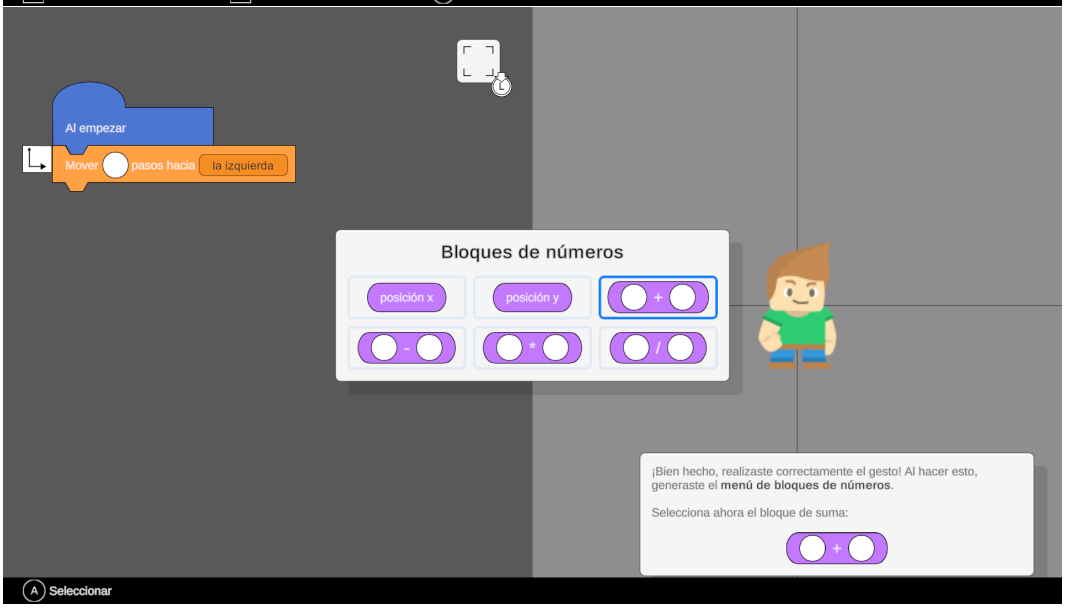

Gestos
Movimiento
Números
Booleanos
Condicional #1
Condicional #2
Al empezar
Mover pasos hacia la izquierda
La figura que traces con los cursores se llama gesto. Hay diferentes bloques asociados a cada gesto, que se pueden ver en la lista en pantalla:
Gestos
Abajo de los gestos, aparece el nombre del tipo de bloque asociado. Por ejemplo, en el primero de la lista (de izquierda a derecha) son los bloques de movimiento.
Anterior
Siguiente
ZL Empezar gesto cursor izquierdo
ZR Empezar gesto cursor derecho
B Salir

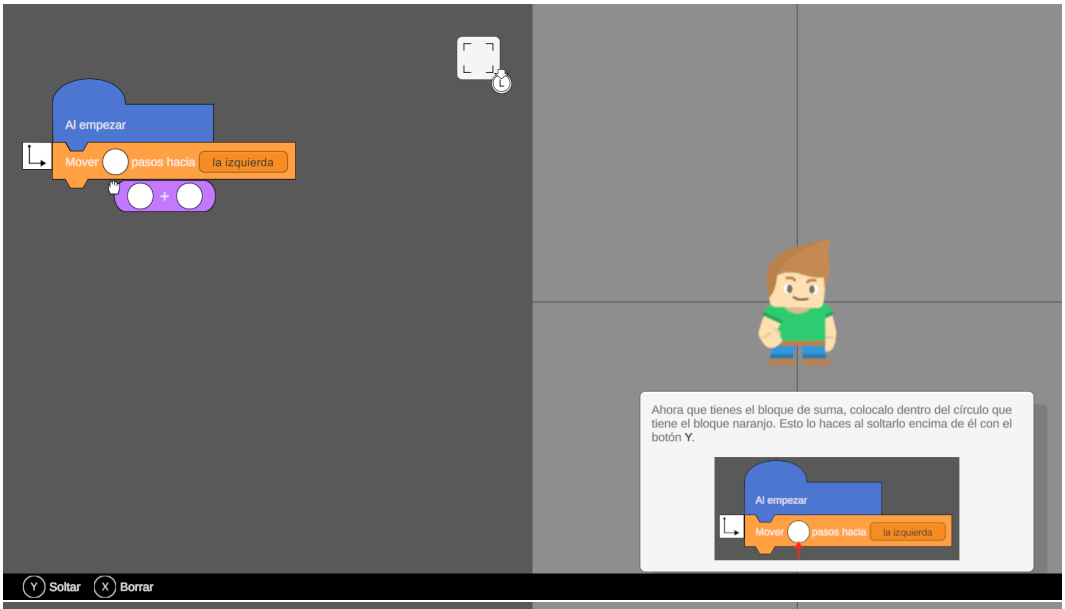

Gestos
Movimiento
Números
Booleanos
Condicional #1
Condicional #2
Al empezar
Mover pasos hacia la izquierda
Para realizar los gestos, tienes que dibujar siguiendo la flecha del gesto. Por ejemplo en el siguiente gesto, al seguir la flecha dibujas un triángulo
Anterior
Siguiente
ZL Empezar gesto cursor izquierdo
ZR Empezar gesto cursor derecho
B Salir

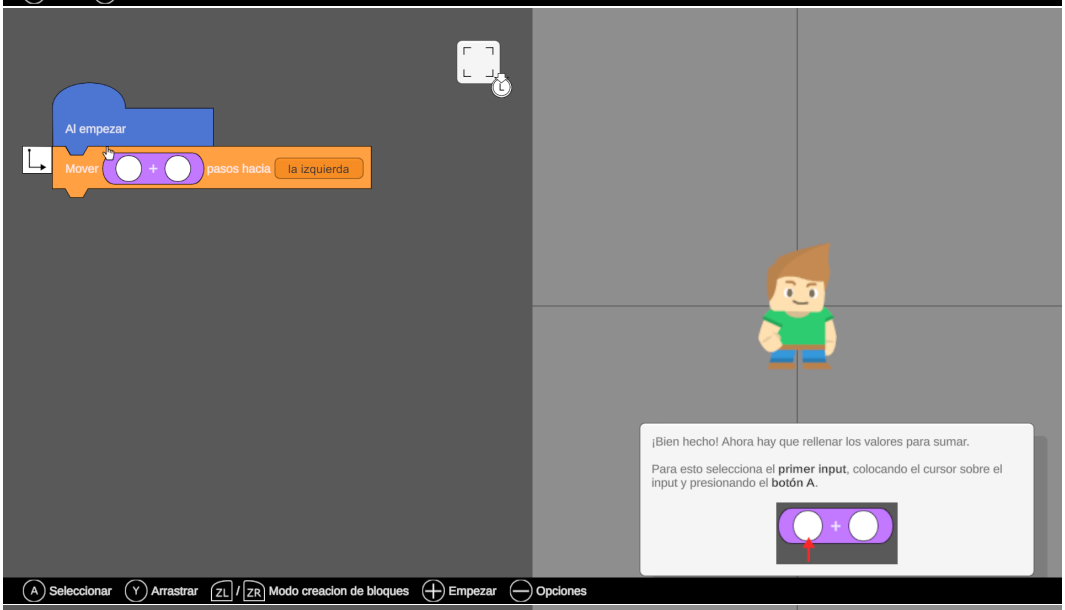

Gestos
Movimiento
Números
Booleanos
Condicional #1
Condicional #2
Al empezar
Mover pasos hacia la izquierda
Como puedes ver hay distintos tipos de gestos.
Los que son de color negro, se pueden dibujar con cualquiera de los dos cursores.
Sí son azules solamente pueden ser realizados con el Joy-Con izquierdo y los rojos con el derecho.
Anterior
Siguiente
ZL Empezar gesto cursor izquierdo
ZR Empezar gesto cursor derecho
B Salir

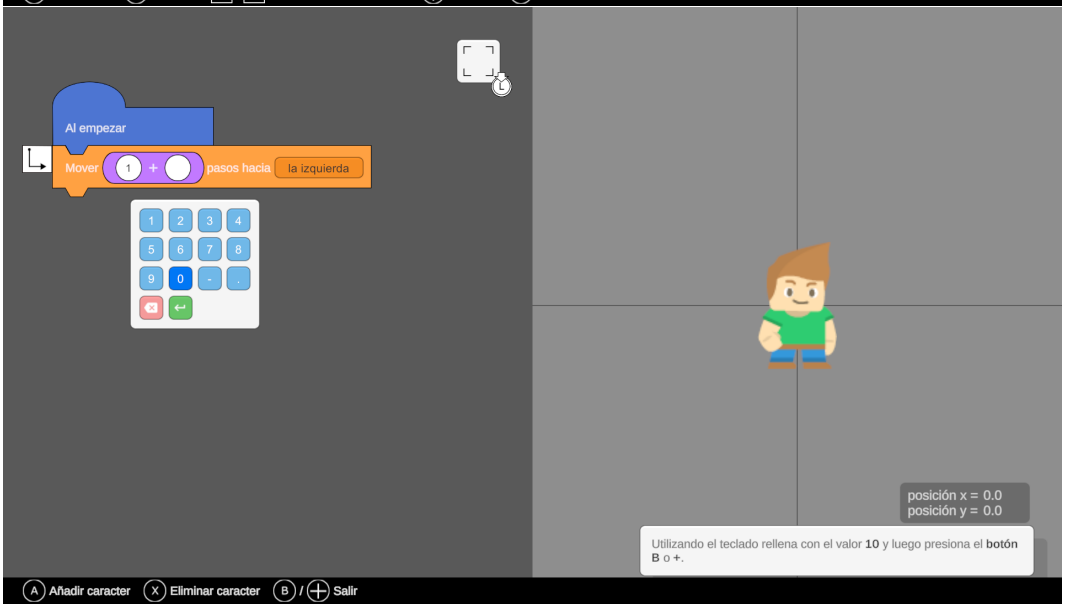

Gestos
Movimiento
Números
Booleanos
Condicional #1
Condicional #2
Al empezar
Mover pasos hacia la izquierda
Finalmente está el gesto que contiene dos flechas con dos colores distintos:
Este gesto se realiza al dibujar con ambos controles al mismo tiempo. Aquí mueves el Joy-Con izquierdo hacia la izquierda y el Joy-Con derecho hacia la derecha.
Anterior
Siguiente
ZL Empezar gesto cursor izquierdo
ZR Empezar gesto cursor derecho
B Salir

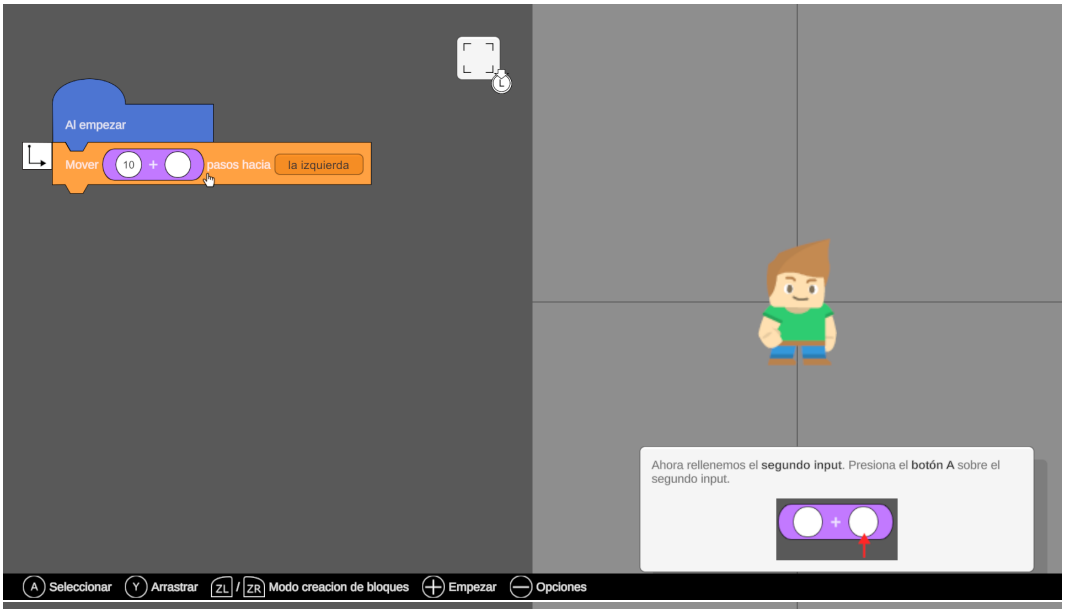

Gestos
Movimiento
Números
Booleanos
Condicional #1
Condicional #2
Al empezar
Mover pasos hacia la izquierda
Ahora vamos a crear un bloque.
Primero, con uno de los Joycon presiona el gatillo (ZL en el izquierdo, ZR en el derecho). Esto va a hacer que el cursor correspondiente comience a dibujar. Trata de dibujar el siguiente gesto:
ZL Empezar gesto cursor izquierdo
ZR Empezar gesto cursor derecho
B Salir

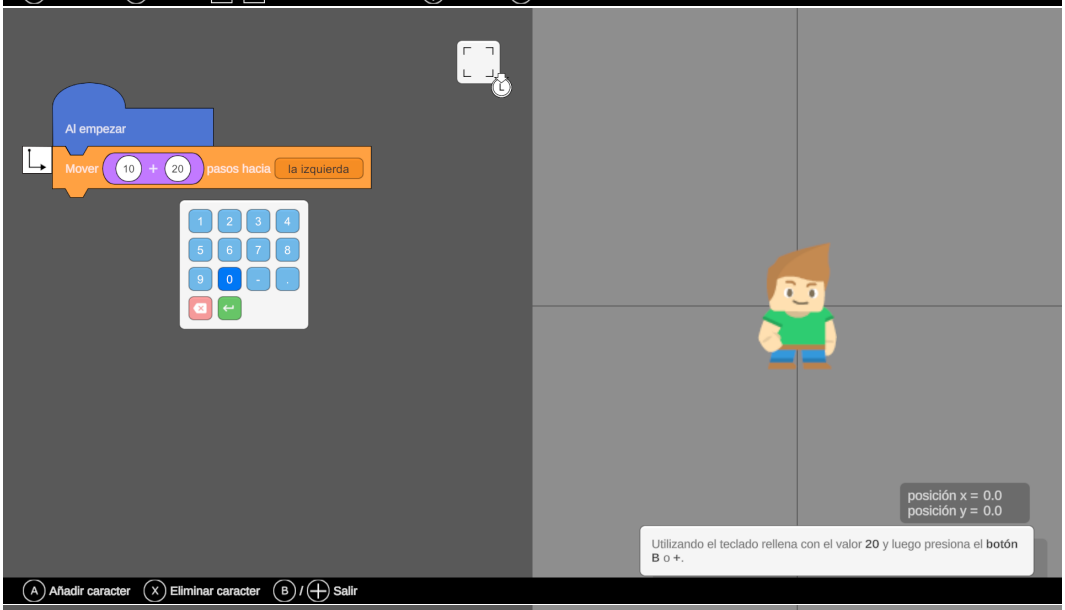

Al empezar
Mover pasos hacia la izquierda
Bloques de números
posición x
posición y
¡Bien hecho, realizaste correctamente el gesto! Al hacer esto, generaste el menú de bloques de números.
Selecciona ahora el bloque de suma:
A Seleccionar

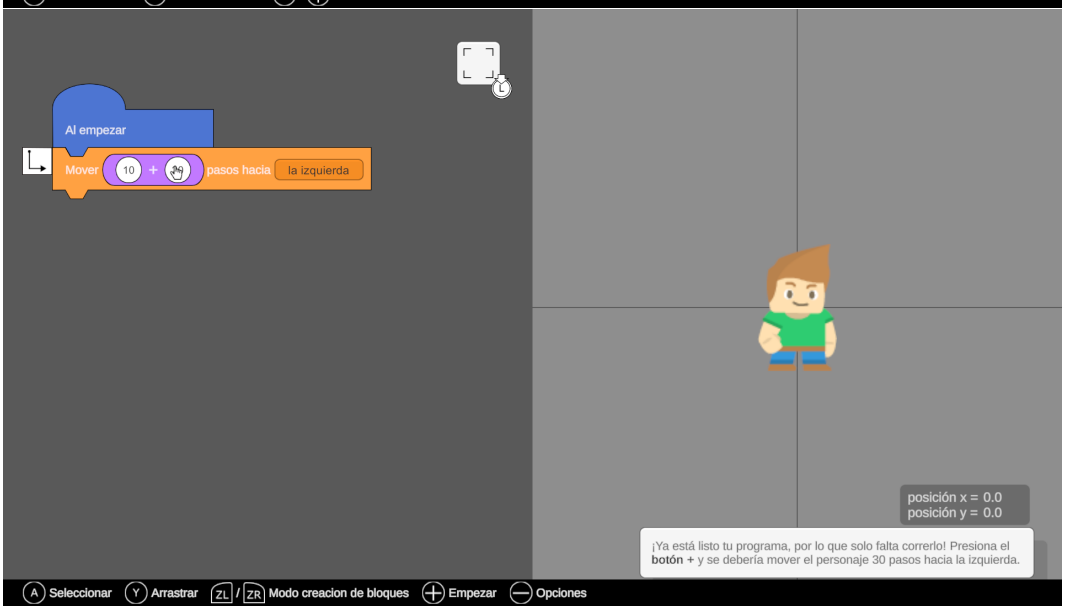

Al empezar
Mover
pasos hacia
la izquierda
Ahora que tienes el bloque de suma, colocalo dentro del círculo que tiene el bloque naranjo. Esto lo haces al soltarlo encima de él con el botón Y.
Al empezar
Mover
pasos hacia
la izquierda
Y Soltar
X Borrar

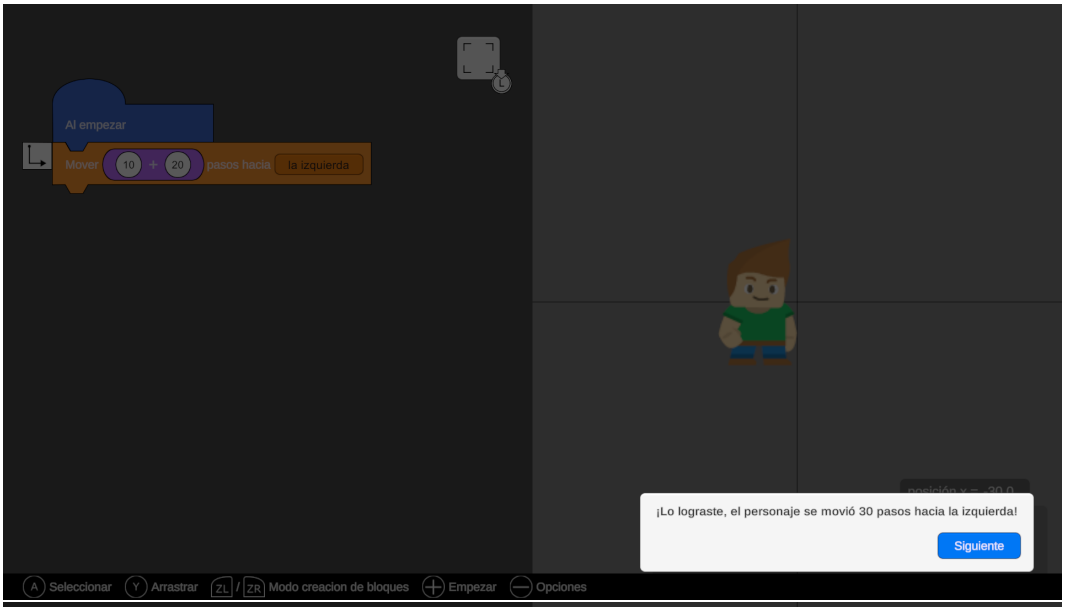

Al empezar
Mover
pasos hacia
la izquierda
¡Bien hecho! Ahora hay que rellenar los valores para sumar.
Para esto selecciona el primer input, colocando el cursor sobre el input y presionando el botón A.
A Seleccionar
Y Arrastrar
ZL / ZR Modo creacion de bloques
Empezar
Opciones

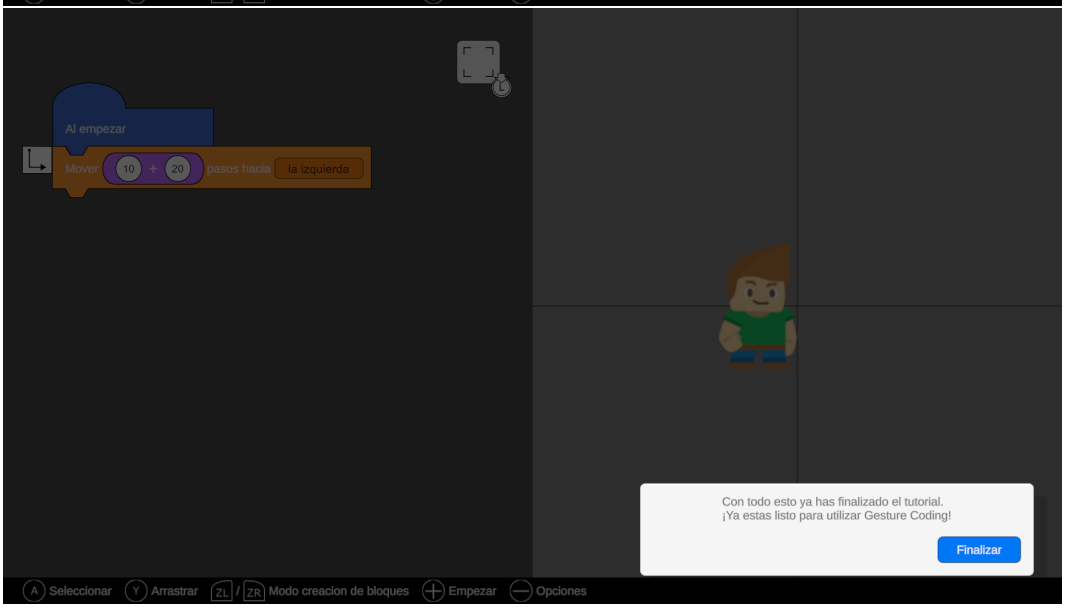

Al empezar
Mover
1
pasos hacia
la izquierda
posición x = 0.0
posición y = 0.0
Utilizando el teclado rellena con el valor 10 y luego presiona el botón B o +.
A Añadir caracter
X Eliminar caracter
B / Salir


Al empezar
Mover 10 + pasos hacia la izquierda
Ahora rellenemos el segundo input. Presiona el botón A sobre el segundo input.
A Seleccionar Y Arrastrar ZL / ZR Modo creacion de bloques Empezar Opciones


Al empezar
Mover 10 + 20 pasos hacia la izquierda
posición x = 0.0
posición y = 0.0
Utilizando el teclado rellena con el valor 20 y luego presiona el botón B o +.
A Añadir caracter X Eliminar caracter B / Salir


Al empezar
Mover 10 + 30 pasos hacia la izquierda
posición x = 0.0
posición y = 0.0
¡Ya está listo tu programa, por lo que solo falta correrlo! Presiona el botón + y se debería mover el personaje 30 pasos hacia la izquierda.
A Seleccionar Y Arrastrar ZL / ZR Modo creacion de bloques Empezar Opciones


Al empezar
Mover 10 + 20 pasos hacia la izquierda
¡Lo lograste, el personaje se movió 30 pasos hacia la izquierda!
Siguiente
A Seleccionar
Y Arrastrar
ZL / ZR Modo creacion de bloques
Empezar
Opciones


Al empezar
Mover 10 + 20 pasos hacia la izquierda
Con todo esto ya has finalizado el tutorial.
¡Ya estas listo para utilizar Gesture Coding!
Finalizar
A Seleccionar
Y Arrastrar
ZL / ZR Modo creacion de bloques
Empezar
Opciones