

UNIVERSIDAD TÉCNICA FEDERICO SANTA MARÍA
DEPARTAMENTO DE INFORMÁTICA
SANTIAGO - CHILE



“ENTORNO PARA EL APRENDIZAJE DE LA PROGRAMACIÓN POR BLOQUES PARA PERSONAS CON DISCAPACIDAD VISUAL”

RODRIGO ADOLFO CAYAZAYA MARÍN

MEMORIA PARA OPTAR AL TÍTULO DE
INGENIERO CIVIL EN INFORMÁTICA

Profesor Guía: Federico Meza
Profesora Correferente: Claudia López

Enero - 2023

DEDICATORIA

Para Hernán Castillo y sobre todo para el Instituto Antonio Vicente Mosquete
por confiar en mí.

AGRADECIMIENTOS

Agradezco a mis amigos de Iquique, a mis amigos y compañeros de universidad, a los buenos profesores, a mi jefe de carrera, a mi pareja, a mis abuelos y padres.
Sin ustedes no podría haber llegado tan lejos.

RESUMEN

Resumen— La programación por bloque, debido a su naturaleza visual, se encuentra limitada para las personas con discapacidad visual, lo cual plantea un desafío significativo. A pesar de varias propuestas, ninguna solución ha sido perfecta ni adaptada al público hispanohablante. Para abordar esta problemática, se decide crear un entorno que permita a las personas con discapacidad visual utilizar la programación por bloque. Se eligió Accessible Blockly como base, tras observar la interacción de estudiantes con discapacidad visual en un instituto y aplicar el método de observación AEIOU.

Accessible Blockly, una idea desechada de Google, se escoge como la opción principal debido a la ausencia de requerimientos específicos de hardware o experiencia previa en programación. Para la creación de un prototipo se implementan mejoras a Accessible Blockly, como la traducción al español, un tutorial básico y la capacidad de ejecutar código creado y traducirlo a Javascript. Después de crear el prototipo, se llevan a cabo pruebas solitarias y de co-desarrollo con tres usuarios, aplicando mejoras basadas en sus comentarios y realizando un test de deseabilidad.

La plataforma se percibe por los usuarios como accesible y novedosa, pero compleja. Es por esto que se añadió una ayuda visual en forma de imágenes tipo gif en tutoriales para facilitar la colaboración con personas guías. Tras análisis y ajustes basados en pruebas de usuario, se concluye que el desarrollo para personas con discapacidad visual es todo un desafío, del cual surgió una plataforma que cumple con los objetivos deseados. El proyecto completo se encuentra disponible en el repositorio: <https://github.com/RodrigoCaya/Blockly-Accessible>.

Palabras Clave— Programación por bloque, Discapacidad visual, Accesibilidad, Co-desarrollo.

ABSTRACT

Abstract— Block coding, due to its visual nature, is limited for individuals with visual disabilities, posing a significant challenge. Despite various proposals, no solution has been perfect or tailored to the Spanish-speaking audience. To address this problem, a decision was made to create an environment that enables individuals with visual disabilities to engage in block coding. Accessible Blockly was chosen as the foundation after observing the interaction of visually impaired students in an institute and applying the AEIOU observation method.

Accessible Blockly, a discarded idea from Google, is selected as the primary option due to the absence of specific hardware requirements or previous programming experience. To create a prototype, improvements are implemented in Accessible Blockly, such as Spanish translation, a basic tutorial, and the ability to execute created code and translate it to JavaScript. After creating a prototype, solo and co-development tests are conducted with three users, incorporating improvements based on their feedback, and a desirability test is performed.

The platform is perceived by users as accessible and innovative, but complex. Therefore, visual assistance in the form of gif images were added to tutorials to facilitate collaboration with sighted individuals. After analysis and adjustments based on user testing, it is concluded that development for individuals with visual disabilities is a significant challenge, resulting in a successful platform that meets the desired objectives. The complete project is available in the repository: <https://github.com/RodrigoCaya/Blockly-Accessible>.

Keywords— Block coding, Visual impairment, Accessibility, Co-development.

GLOSARIO

NVDA: *Non Visual Desktop Access*. Es un lector de pantalla para computador.

VI: *Visual Impairment*. Discapacidad visual.

Feedback: Retroalimentación.

Shortcuts: Atajos de teclado.

AudioDoom: Videojuego para personas con discapacidad visual basado en Doom.

AudioDisco: Videojuego para personas con discapacidad visual.

Clúster: Conjunto o grupo de notas que forman una idea.

Typo: Error de tipografía.

ÍNDICE DE CONTENIDOS

RESUMEN	IV
ABSTRACT	V
GLOSARIO	VI
ÍNDICE DE FIGURAS	IX
ÍNDICE DE TABLAS	X
INTRODUCCIÓN	1
CAPÍTULO 1: DEFINICIÓN DEL PROBLEMA	3
1.1 Origen del problema	3
1.2 Soluciones generales	4
1.3 Soluciones para la programación en bloque	5
1.4 Objetivos	6
1.4.1 Objetivo General	6
1.4.2 Objetivos Específicos	6
CAPÍTULO 2: MARCO CONCEPTUAL	7
2.1 Definición de conceptos generales	7
2.2 Programación por bloque	7
2.2.1 Scratch	8
2.2.2 Blockly	9
2.3 Soluciones propuestas	9
2.3.1 Ambientes creados desde cero	10
2.3.2 Ambientes basados en Blockly	11
CAPÍTULO 3: PROPUESTA DE SOLUCIÓN	13
3.1 Escogiendo una solución	13
3.1.1 Observaciones	14
3.1.2 Descubrimientos	15
3.1.3 Solución escogida	20
3.2 Utilización de Accessible Blockly	20
3.2.1 Ejecutando Accessible Blockly	20
3.3 Nacimiento de Blockly Accesible	21
3.3.1 Lo bueno, lo malo y lo feo	21
3.4 Implementando mejoras	22
3.4.1 Traducción al español	22
3.4.2 Agregando un propósito	23
3.4.3 Agregando un tutorial	24

CAPÍTULO 4: VALIDACIÓN DE LA SOLUCIÓN	26
4.1 Pruebas en solitario	26
4.1.1 Prueba con NVDA y vendado	26
4.2 Preparación pruebas de usuario	27
4.2.1 Supuestos y expectativas	27
4.2.2 Tareas	28
4.2.3 Test de deseabilidad	29
4.3 Pruebas de usuarios	29
4.3.1 Identificación de colores	29
4.3.2 Conociendo a los usuarios	30
4.3.3 Resultados	31
4.4 Ayuda visual	38
4.5 Entrevista a usuarios	39
4.5.1 Estructura	39
4.5.2 Preguntas y respuestas	40
CAPÍTULO 5: CONCLUSIONES	43
5.1 Aprendizajes	43
5.1.1 Aprendizajes test de deseabilidad	43
5.1.2 Aprendizajes del co-desarrollo	44
5.1.3 Aprendizajes durante el desarrollo	45
5.2 Conclusiones finales	45
5.2.1 Trabajo futuro	46
ANEXOS	47
REFERENCIAS BIBLIOGRÁFICAS	52

ÍNDICE DE FIGURAS

1	Ejemplo de un programa en programación por bloque.	4
2	Árbol de definición del problema.	5
3	Programa hecho en Scratch.	8
4	Programa hecho en Blockly.	9
5	Diagrama de flujo.	13
6	Colores utilizados por tema.	14
7	Descubrimientos: clúster N°1.	15
8	Descubrimientos: clúster N°2.	16
9	Descubrimientos: clúster N°3.	17
10	Descubrimientos: cluster N°4.	18
11	Descubrimientos: clúster N°5.	18
12	Descubrimientos: clúster N°6.	19
13	Última versión obtenible de Accessible Blockly.	21
14	Primer prototipo traducido al español.	23
15	Ejecución de código.	24
16	Traducción a Javascript.	24
17	Pantalla principal del tutorial 1.	25
18	Instrucciones del tutorial 1.	25
19	Colores utilizados por pruebas.	30
20	Pruebas de usuario: clúster N°1.	32
21	Pruebas de usuario: clúster N°2.	33
22	Pruebas de usuario: clúster N°3.	33
23	Pruebas de usuario: clúster N°4.	34

24	Pruebas de usuario: clúster N°5.	35
25	Pruebas de usuario: clúster N°6.	36
26	Pruebas de usuario: clúster N°7.	36
27	Pruebas de usuario: clúster N°8.	37
28	Ayuda visual en tutoriales.	38
29	Ayuda visual en tareas.	39
30	Resultados test de deseabilidad.	44

ÍNDICE DE TABLAS

1	Tabla comparativa entre las soluciones existentes.	12
2	Tabla de supuestos y expectativas.	28
3	Palabras utilizadas para el test de deseabilidad	29
4	Problemas y soluciones: prueba con NVDA.	47
5	Problemas y soluciones: prueba con NVDA y vendado.	48
6	Problemas y soluciones: primera prueba usuario 1.	48
7	Problemas y soluciones: primera prueba usuario 2.	49
8	Problemas y soluciones: segunda prueba usuario 1.	50
9	Problemas y soluciones: primera prueba usuario 3.	51
10	Problemas y soluciones: segunda prueba usuario 3.	51

INTRODUCCIÓN

La programación por bloque siempre ha sido inherentemente visual, esto es debido a la necesidad de unir bloques de código. Sin embargo, existe una parte de la población incapaz de poder utilizar este tipo de programación: las personas con discapacidad visual.

Existen diversas soluciones propuestas: Noodle [Lewis, 2014], CodeMirror-Blocks [Bahram *et al.*, 2019], Blocks4All [Milne y Ladner, 2018], Accessible Blockly [Ludi, 2015] y Pseudospacial Blocks [Koushik y Lewis, 2016]. Sin embargo, ninguna ha sido perfecta ni tampoco se han adaptado al público hispanohablante.

Es por esto que, con el fin de solucionar esta falta de solución al problema, se decide crear un entorno que permita utilizar la programación por bloque a las personas con discapacidad visual.

Para realizar este programa se decidió escoger una de las soluciones anteriormente mencionadas para tomar como punto de referencia. No obstante, primero era necesario conocer al usuario que utilizaría el programa. Para ello, se visitó un instituto para personas con discapacidad visual donde se pudo observar una clase de computación y conocer cómo los estudiantes interactúan con el computador, para lograr esto se aplicó el método de observación AEIOU [Estudio, 2023].

De esta visita se rescataron 6 descubrimientos importantes:

1. Utilizar lector de pantalla es la opción más accesible y más utilizada para interactuar con el computador.
2. NVDA es una buena opción al momento de escoger un lector de pantalla.
3. Los indicadores auditivos hacen la diferencia.
4. Se valora mucho la navegación accesible.
5. Es común la utilización de *shortcuts*.
6. La curva de aprendizaje es mayor para una persona con discapacidad visual.

Debido a que no requiere de ningún hardware especializado ni tampoco de ningún tipo de experiencia programando, se seleccionó a Accessible Blockly como la opción a utilizar.

Accessible Blockly es una idea descartada de Google (los originales dueños de Blockly) que se desarrolló hasta el año 2020, por lo que necesitaba de muchas mejoras para poder crear el prototipo deseado.

Las primeras mejoras que se introdujeron fueron la traducción al español, la creación de un tutorial básico y la posibilidad de correr el código creado y poder traducirlo a Javascript.

Con el prototipo ya creado se realizaron pruebas de manera solitaria, intentando simular la experiencia que tendría el usuario y aplicando estas mejoras al prototipo, para luego realizar pruebas de usuario utilizando el metodología de co-desarrollo, debido a que solamente se pudo conseguir a 3 usuarios dispuestos a realizar pruebas. Sin embargo, gracias al co-desarrollo se pudo realizar un total de 5 pruebas, aplicando las mejoras a partir del feedback del usuario después de cada prueba. Además se realizó un test de deseabilidad [Moran, 2016] después de realizar la primera prueba con cada usuario, de la cual se pudo rescatar que los usuarios sintieron que el proyecto era accesible y novedoso, pero también complejo. Es por esto que se creó una ayuda visual en base a imágenes tipo gif colocadas en los tutoriales para que el usuario pueda pedir ayuda a una persona guía vidente si así lo desea.

Finalmente se realizó una entrevista a uno de los usuarios donde explicó cómo fue su experiencia realizando este co-desarrollo, al igual que su experiencia con la tecnología siendo una persona con discapacidad visual.

Para concluir, es posible decir que desarrollar para personas con discapacidad visual es todo un reto y su validación tomó más del tiempo estimado, pero fue provechoso al poder crear una plataforma que cumple con los objetivos deseados, la cual es de acceso gratuito y es posible ingresar al código de este proyecto a través del siguiente repositorio: <https://github.com/RodrigoCaya/Blockly-Accesible>.

CAPÍTULO 1

DEFINICIÓN DEL PROBLEMA

Hoy en día la programación, aunque sea de manera básica, se ha convertido en una habilidad esencial en el mundo computarizado en el cual vivimos. Ya sea para aprender un poco más sobre cómo funcionan los programas que utilizamos o para desarrollar el pensamiento lógico-matemático, la programación se está convirtiendo en una habilidad necesaria independiente del área en que un individuo se especialice. Es por esto que diferentes países han tomado resoluciones para enseñar la habilidad de programar a niños desde una edad temprana, además la habilidad de programación está siendo requerida dentro de los currículos educacionales en más de 12 países de Europa [de Oliveira, 2017].

Debido a esta reforma educacional en Europa, y los distintos países que se guían por sus modelos, se puede asumir que en un futuro la mayoría de los países del mundo incluirá la programación en sus planes de estudio. Ahora si bien lo ideal es enseñar a programar desde una temprana edad, independiente de la edad que tenga el estudiante, aprender a programar puede ser un reto bastante complicado, no obstante existen lenguajes de programación más amigables para los usuarios principiantes, tal como es el caso de la programación por bloque.

Con ambientes de programación famosos como Scratch¹ o Blockly², la programación en bloque surgió como una forma de enseñar los conceptos de programación a niños desde los 5 años. Este tipo de programación utiliza piezas de puzzle para entregar pistas visuales al usuario sobre cómo y dónde los comandos deben ser utilizados [Guzdial, 2019]. Estas piezas de puzzle contienen extractos de código que pueden ser modificados, los cuales permiten al usuario despreocuparse de la sintaxis y solamente preocuparse de arrastrar y unir estas piezas de manera correcta para construir instrucciones de manera consecutivas, tal como se muestra en la Figura 1.

1.1. Origen del problema

Debido a su naturaleza simplificada e interactiva, la programación en bloque está diseñada para ser amigable, especialmente con los usuarios con poca o nula interactividad previa en la programación. Sin embargo, a causa de esta naturaleza simplificada de ver los fragmentos de códigos como puzzles, es que nace una necesidad visual debido a que es necesario observar los fragmentos de código para poder unirlos, lo que ha provocado que la adaptación de la programación por bloque para personas con discapacidad visual (VI³) sea todo un desafío, generando poca variedad al momento de decidir incursionar en el mundo de la programación

¹<https://scratch.mit.edu>

²<https://developers.google.com/blockly>

³VI = Visual Impairment.

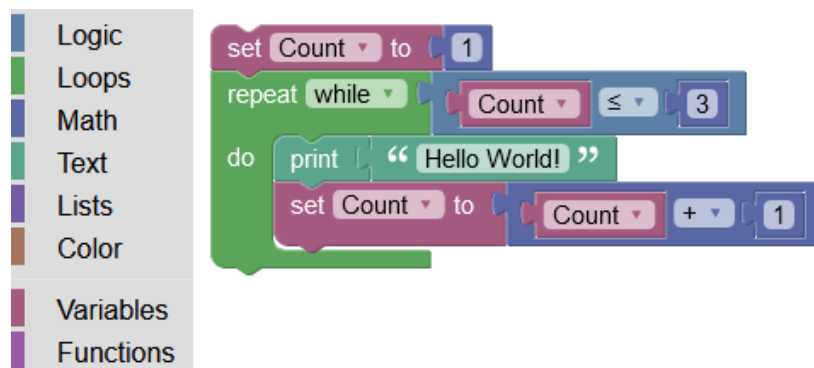


Figura 1: Ejemplo de un programa en programación por bloque.
Fuente: <https://developers.google.com/blockly>.

siendo una persona con discapacidad visual. Esto cobra mayor relevancia cuando, hasta el año 2018 existían 253 millones de personas VI en todo el mundo, por lo que las necesidades de este grupo de personas no deben ser olvidadas.

Privar a un niño de la oportunidad de aprender programación no solamente le impide obtener los beneficios de esta habilidad, ya que quizás sea una noticia nueva para algunas personas, pero existen aproximadamente 134.000 programadores VI [Larson, 2017], lo cuál demuestra que esto no es un impedimento para poder aprender la habilidad de programación. Esto refuerza la importancia de incluir y fomentar el aprendizaje de esta habilidad en niños con discapacidad visual, sobre todo en países hispanohablantes, ya que la poca investigación enfocada a este grupo proviene de fuentes angloparlantes, generando una necesidad urgente de profundizar en la investigación de este grupo segregado.

La habilidad de programación se está volviendo cada vez más importante, sobre todo enseñarla desde una temprana edad, y bajo el principio de la igualdad, es importante que sea de manera equitativa, es por esto que se han propuesto variadas soluciones.

A modo de resumen se presenta un árbol de definición del problema en la Figura 2.

1.2. Soluciones generales

Hoy en día existen diferentes soluciones que utilizan las personas VI para poder programar, tanto para lenguajes de programación basados en bloques como para lenguajes basados en texto, sin embargo la gran mayoría de estas soluciones son para este último grupo [Hadwen-Bennett *et al.*, 2018], ya que este tipo de lenguaje conlleva una menor carga visual.

Independiente del lenguaje de programación que utilice el usuario, este se debe ser capaz de comprender lo que se muestra en pantalla. Para esto se han utilizado diferentes soluciones: un lector de pantalla, un magnificador de pantalla o una línea braille.

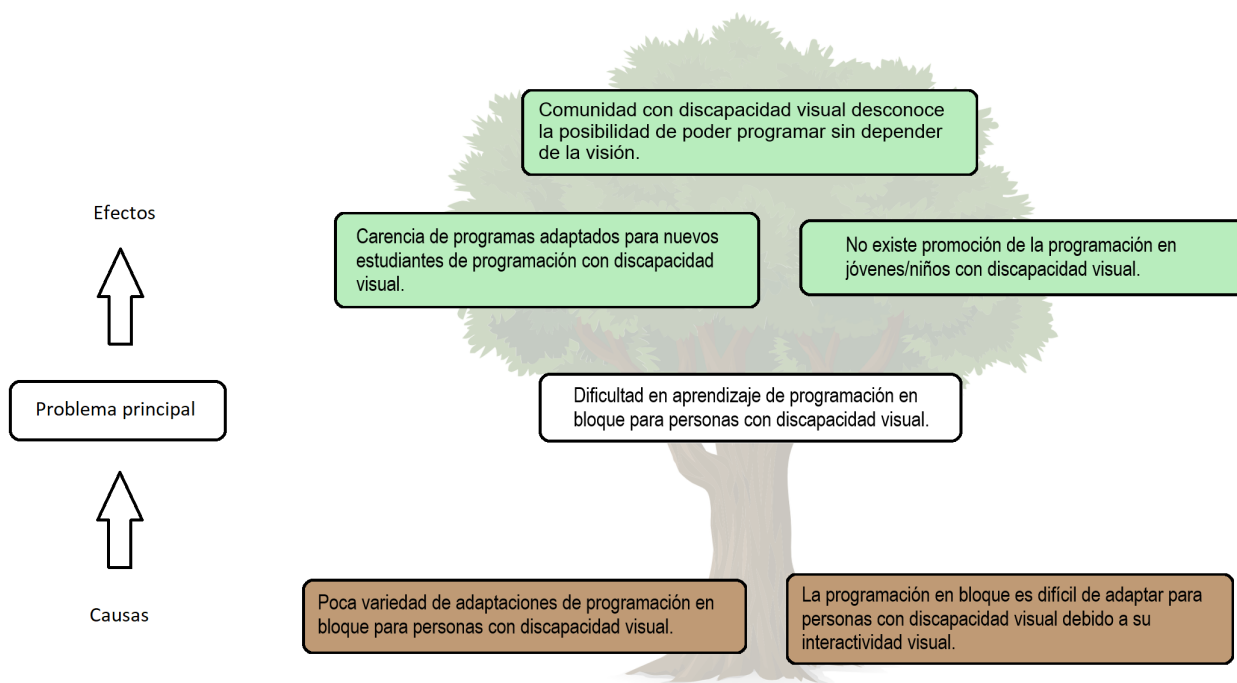


Figura 2: Árbol de definición del problema.

De entre los 3 mencionados anteriormente, el lector de pantalla es el preferido entre los programadores VI, sin embargo no son perfectos, ya que deben personalizarse para cada nuevo programa y no todos los programas son compatibles [Bigham *et al.*, 2008] es por esto que nacieron iniciativas como Javaspeak [Francioni y Smith, 2002, Smith *et al.*, 2000, Smith *et al.*, 2003], un *plug-in* del ambiente de programación Eclipse que permite al usuario programar en el lenguaje basado en texto JAVA sin ningún problema al momento de navegar entre las herencias de clases.

1.3. Soluciones para la programación en bloque

Las soluciones actuales para implementar la programación por bloque para personas VI son variadas, estos trabajos constan desde la realización de entornos de programación desde cero como Noodle [Lewis, 2014], Blocks4All [Milne y Ladner, 2018] o CodeMirror-Blocks [Bahram *et al.*, 2019], hasta la modificación de lenguajes de programación por bloques como Accessible Blockly [Ludi, 2015] y Pseudospacial Blocks [Koushik y Lewis, 2016]. Cabe destacar que cada uno de estos trabajos contienen defectos para el contexto en el cuál se requiere dar una solución.

Es de importancia resaltar que los trabajos anteriormente mencionados utilizaron algún tipo de lector de pantalla para poder comunicar la salida y lectura de lo que se encontraba.

Además están hechos para angloparlantes, lo cual no realiza un enfoque en los usuarios a los que va dirigida esta investigación.

1.4. Objetivos

1.4.1. Objetivo General

Desarrollar un entorno para el aprendizaje de la programación por bloques en personas con discapacidad visual.

1.4.2. Objetivos Específicos

- Realizar un estudio comparativo entre las soluciones actuales.
- Diseñar un prototipo funcional.
- Crear guías de trabajo para el prototipo.
- Evaluar el prototipo creado para validar su eficacia.

CAPÍTULO 2

MARCO CONCEPTUAL

2.1. Definición de conceptos generales

- **Lectores de pantalla.** Son una pieza de software que sintetiza en audio la información textual mostrada en pantalla. Esta síntesis puede llevarse a cabo de diferentes formas, al leer sentencias, palabras, o caracteres individuales [Cheong, 2010]. Existen diferentes lectores de pantalla para diferentes tipos de sistemas operativos. Los más populares son: JAWS (*Job Access With Speech*) para Windows, VoiceOver para iOS y Emacspeak para sistemas operativos variados.
- **Línea braille.** Son aparatos capaces de mostrar una línea de texto de la pantalla en formato *braille* y poder cambiarla de forma dinámica mientras el usuario navega entre el texto de la pantalla [Cheong, 2010], permitiendo al usuario poder entender que lo aparece en pantalla sin realizar una sobrecarga de audio como sucede con los lectores de pantalla.
- **Magnificador de pantalla.** Es un programar capaz de agrandar los contenidos de la pantalla al escalar los elementos en pantalla y mantener los componentes importantes en pantalla [Bigham *et al.*, 2008], siendo ZoomText uno de los magnificadores más utilizados.

2.2. Programación por bloque

La programación por bloque es como la programación de texto tradicional, pero con la diferencia de que las instrucciones vienen modularizadas en forma de piezas de puzle que permiten unirse entre sí para formar instrucciones concatenadas, generalmente el resultado de estas instrucciones puede ser visualizado a través de una pantalla. Los ambientes de la programación por bloque están orientadas a un público de entre 8 a 16 años de edad [Guzdial, 2019], esto lo logran quitándole flexibilidad al usuario al permitirle solamente modificar de manera limitada las instrucciones y prohibiendo la unión de 2 instrucciones inválidas, logrando que niños o adultos sin experiencia en la programación, puedan crear programas sin preocuparse de la sintaxis. Para que los usuarios conozcan qué instrucciones pueden ser unidas, las piezas de puzle están conceptualmente organizadas y coloreadas, de modo que sea intuitiva la unión de instrucciones. Siendo los 2 ambientes de programación más utilizados son Blockly y Scratch.

2.2.1. Scratch

Scratch es un ambiente de programación por bloques gratuito creado por el Instituto de Tecnología de Massachusetts (MIT) con el objetivo de enseñar a programar a niños y niñas de manera sencilla e interactiva. Scratch incluye una pantalla que permite a los jóvenes crear historias digitales, juegos y animaciones, tal como se observa en la Figura 3. Para incentivar la pedagogía en la programación, Scratch cuenta con tutoriales y guías de iniciación creadas por el MIT, además de una comunidad de estudiantes donde pueden publicar sus proyectos y comentar sobre otros proyectos.

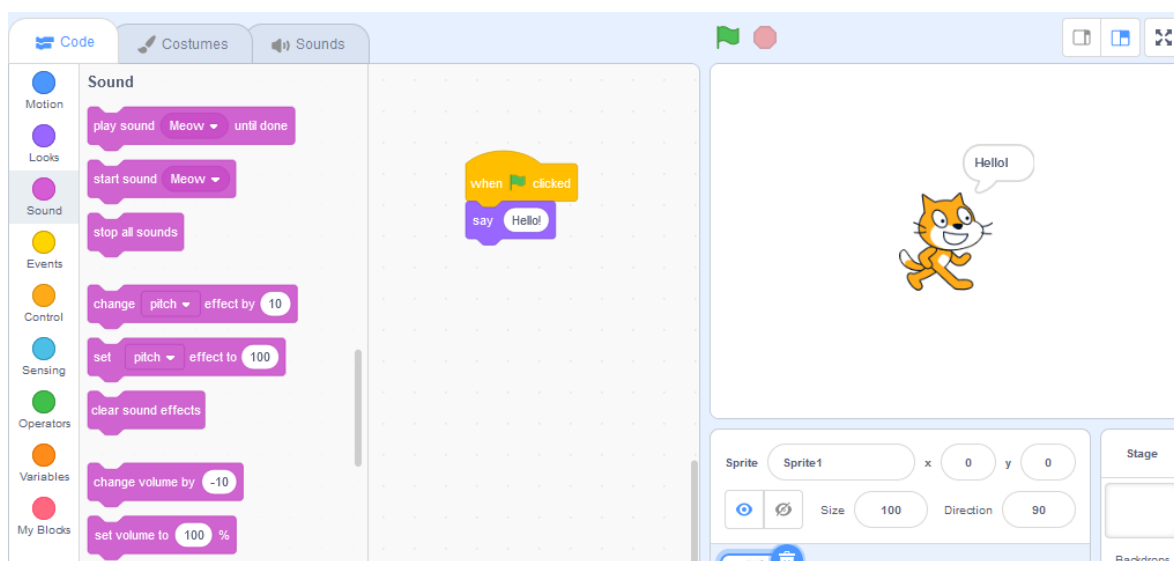


Figura 3: Programa hecho en Scratch.

Fuente: <https://scratch.mit.edu>.

Actualmente, la última versión de Scratch es la versión 3.0, la cual es de código abierto y funciona directamente desde el navegador, permitiendo a los usuarios ingresar desde cualquier dispositivo que tenga conexión a internet. Además, es posible descargar una versión de ordenador llamada “Scratch Desktop”.

Existen diferentes variaciones de Scratch, tal como es el caso de ScratchJr⁴, un ambiente de programación por bloques creada por los mismos creadores de Scratch, pero enfocado en niños de entre 5 a 7 años permitiendo introducir las habilidades de la programación a niños que aún no son capaces de leer bien. Snap!⁵ es otro ambiente derivado de Scratch, pero con más y diferentes bloques, además de la capacidad de poder crear bloques de manera más personalizada y compleja.

⁴<https://www.scratchjr.org/>

⁵<https://snap.berkeley.edu/about>

2.2.2. Blockly

Blockly es un ambiente de programación por bloques de código abierto creado por Google similar a Scratch, pero con características agregadas que no solamente llaman la atención de niños, sino también de desarrolladores. Una de estas características es que existen bloques más complejos y con instrucciones muy similares a las de un lenguaje de programación basado en texto, tanta es esta similitud que permite observar la representación en tiempo real del código creado en Blockly en lenguajes como Javascript, Lua, Dart, Python o PHP y permite la exportación en estos lenguajes, tal como se observa en la Figura 4.

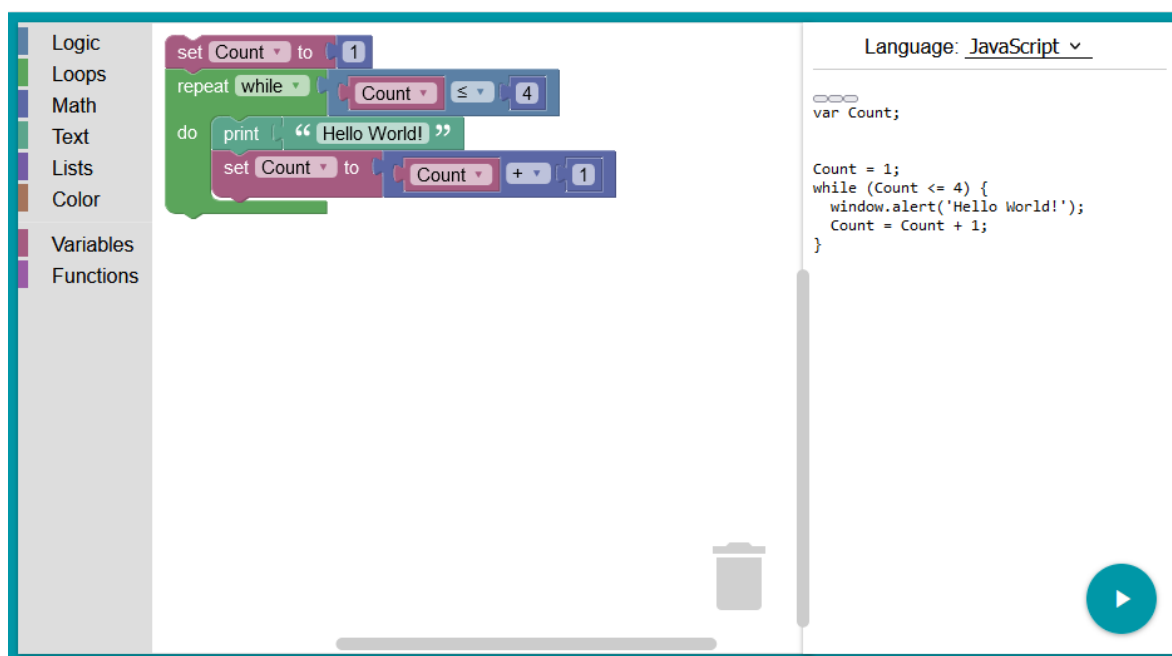


Figura 4: Programa hecho en Blockly.
Fuente: <https://developers.google.com/blockly>.

A pesar de que existen diferencias entre Blockly y Scratch, existe un ambiente en desarrollo por los creadores de Scratch llamado Scratch Blocks⁶ que utiliza el código base de Blockly, el cual ofrece un modelo simplificado ideal para los estudiantes jóvenes.

2.3. Soluciones propuestas

Dentro de las soluciones propuestas se puede observar una segregación de 2 categorías:

- Las soluciones que tomaron inspiración de ambientes de programación por bloques

⁶<https://scratch.mit.edu/developers>

famosos, pero se implementaron completamente desde cero, generando un producto nuevo y personalizado.

- Las soluciones que se implementaron a partir del ambiente de programación por bloques Blockly, donde este fue modificado para brindar la accesibilidad necesaria.

2.3.1. Ambientes creados desde cero

Dentro de la categoría de ambientes de programación creados desde cero encontramos:

- **Noodle** [Lewis, 2014] es un entorno de programación implementado en JavaScript como una aplicación web, la cual permite programar distintos sonidos, concatenarlos, modificar su frecuencia y longitud, esto lo realiza utilizando las flechas del teclado para realizar las acciones de navegación y operaciones para modificar el estado del programa. La navegación depende de los arcos que se vayan creando entre los bloques, formando así un camino por el cual pasan los datos. A pesar de utilizar bloques para representar los fragmentos de códigos, esta aplicación web está pensada para usuarios con un conocimiento mínimo de programación, lo cual da mayor flexibilidad y un mayor alcance al momento de programar, pero aumenta la edad mínima para utilizar este programa escapándose del usuario objetivo de esta investigación.
- **CodeMirror-Blocks (CMB)** [Bahram *et al.*, 2019] es un entorno de programación que se puede adaptar a todos los lenguajes ejecutados en página web, incluyendo lenguajes basados en texto y en bloques. Esto lo realiza creando una división del código en bloques, concatenándolos si existen ambientes diferentes (como por ejemplo código dentro de una función o dentro de un *loop*). A pesar de poder adaptar lenguajes de programación basados en bloques, las pruebas existentes y realizadas fueron en un lenguaje basado en texto, siendo el enfoque en usuarios con experiencia previa en la programación.
- **Blocks4All** [Milne y Ladner, 2018] es un ambiente de programación creado para utilizarse por pantallas táctiles y creado como una aplicación en IOS, debido a que los dispositivos que tienen este sistema operativo vienen con un lector de pantalla integrado. Para que el usuario se pueda guiar dentro de la aplicación, al tocar un bloque este es descrito por audio, al volver a tocar el bloque este es seleccionado. Además, cuenta con diferentes métodos para poder mover un bloque y para indicar la posición espacial de la estructura. Sin embargo, las instrucciones que contienen los bloques son utilizadas exclusivamente para poder controlar un robot, quitándole versatilidad a la programación.

2.3.2. Ambientes basados en Blockly

La otra categoría son los ambientes de programación basado en el ambiente Blockly, estos son:

- **Accessible Blockly** [Ludi, 2015] es una implementación que permite acceder a la colección de bloques, a los componentes dentro de los bloques y mover los bloques seleccionados, solamente utilizando el teclado. Otra característica que posee Accessible Blockly es la capacidad de transmitir de manera visual o auditiva a los usuarios si la unión de los bloques es válido o no, además permite organizar los comentarios que cree el usuario en una estructura de árbol para que el lector de pantalla pueda leerlos sin problema. Todas estas modificaciones permiten que pueda ser utilizados por usuario VI como por usuarios sin problemas visuales. Sin embargo, esta aproximación conlleva una gran carga auditiva, lo cual puede causar molestia en los usuarios VI.
- **Pseudospacial Blocks (PB)** [Koushik y Lewis, 2016], también es un ambiente que utiliza el teclado para navegar por los bloques, permitiendo al usuario programar instrucciones aritméticas, control de estructuras y generación de sonidos capaces de producir secuencias de bloques y por consiguiente melodías. PB está adaptado para usuarios sin conocimiento de programación y esto lo realiza permitiendo al usuario solamente escoger bloques compatibles, así el usuario no puede equivocarse al momento de unir los bloques como piezas de puzle. Sin embargo, esta implementación elimina un aspecto inherente en la programación: aprender a través de los errores.

Es posible visualizar a través de la Tabla 1 que no existe una solución perfecta, es por esto que el objetivo de esta investigación es la creación de un ambiente de programación capaz de recopilar la mayor cantidad de ventajas posibles de las soluciones anteriormente propuestas y aterrizarlo en un contexto donde los usuarios sean niños o adultos con entusiasmo de aprender, que el español sea su idioma principal y nunca hayan programado o se encuentren empezando a aprender a programar.

Solución	Ventajas	Desventajas
Noodle [Lewis, 2014]	Permite programar distintos sonidos, concatenarlos, modificar su frecuencia y longitud.	El usuario requiere un conocimiento mínimo de programación.
CodeMirror-Blocks [Bahram <i>et al.</i> , 2019]	Flexibilidad al poder adaptarse a todos los lenguajes ejecutados en página web	El usuario requiere experiencia previa en la programación.
Blocks4All [Milne y Ladner, 2018]	Utiliza dispositivos que contengan pantallas táctiles, las cuáles son más conocidas por niños.	Exclusivo para interactuar con un robot.
Accessible Blockly [Ludi, 2015]	Permite ser utilizado tanto por usuario VI como por usuarios sin problemas visuales, además indica de manera auditiva si no es posible realizar una conexión de bloques.	Muchas referencias por audio puede generar una carga auditiva.
Pseudospacial Blocks [Koushik y Lewis, 2016]	Permite al usuario programar instrucciones aritméticas, control de estructuras y generar melodías.	Es una solución que se encuentra en desarrollo y no ha habido más actualizaciones desde entonces.

Tabla 1: Tabla comparativa entre las soluciones existentes.

CAPÍTULO 3

PROPUESTA DE SOLUCIÓN

Durante la propuesta de solución se realizaron diferentes métodos de observación para descubrir cuál es la mejor solución existente para este proyecto. Luego, durante la validación de la solución, se realizó un conjunto de preparativos para la realización de diferentes pruebas de usuario, las cuales culminaron en una entrevista final con uno de los participantes. Estas etapas se muestran más a detalle en el diagrama de flujo (Figura 5).

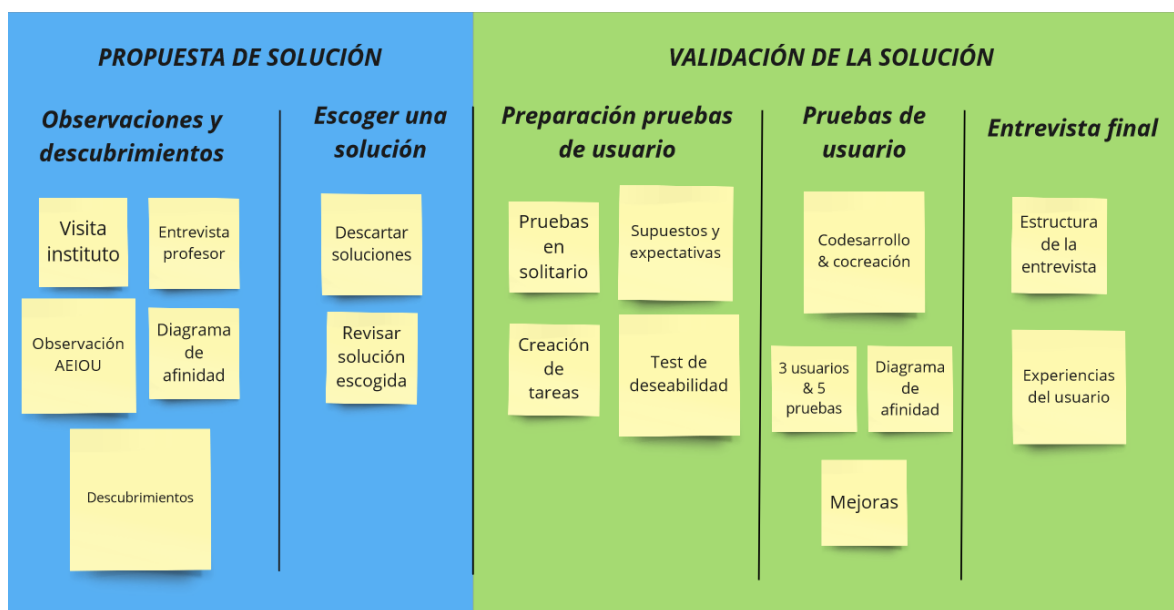


Figura 5: Diagrama de flujo.

3.1. Escogiendo una solución

Con tantas soluciones disponibles, es necesario escoger la que mejor se adapte al objetivo de esta investigación. Sin embargo, se espera que haya usuarios interesados en la solución que se va a proponer, ya que hasta ahora solamente se ha trabajado con supuestos y no es posible saber si realmente existen personas VI que se sientan entusiasmadas por la programación o por la tecnología, quizás esto sea un mundo ajeno para ellas. Además surgen las siguientes interrogantes: ¿cómo utilizan el computador? ¿utilizan el ratón? ¿tienen computadores especiales? Para responder estas dudas es necesario conocer al usuario final que utilizará la plataforma.

3.1.1. Observaciones

Para conocer las interacciones que tienen las personas VI con un computador, se realizó una visita a un instituto educacional para personas con discapacidad visual⁷.

Dentro del instituto se realizó una breve entrevista al profesor de historia y computación del establecimiento, al que se referirá como *P1*, quien tiene 51 años y ha sido profesor por 15 años. Luego, se participó en una clase de computación como oyente, realizando una observación AEIOU [Estudio, 2023], la cual consiste en observar y anotar las actividades, el entorno, las interacciones, los objetos y a los usuarios. En la clase se encontraban 2 estudiantes a las que se referirá como *E1* y *E2*, teniendo 16 y 45 años respectivamente. Además, dentro de esta observación se pudo analizar la interacción de *E1* y su computador al jugar los videojuegos de AudioDoom [SANSARRICQ, 2008] y AudioDisco [Mairena, 2022]. Para terminar, se realizó nuevamente una breve entrevista al profesor para despejar las dudas surgidas durante la observación.

La recolección de datos se tradujo en un conjunto de notas, las cuales se diferenciaron por colores como lo muestra la Figura 6. De estas notas nació un diagrama de afinidad [Asana, 2022] para identificar *clústeres*, los cuales representan los descubrimientos del análisis realizado.

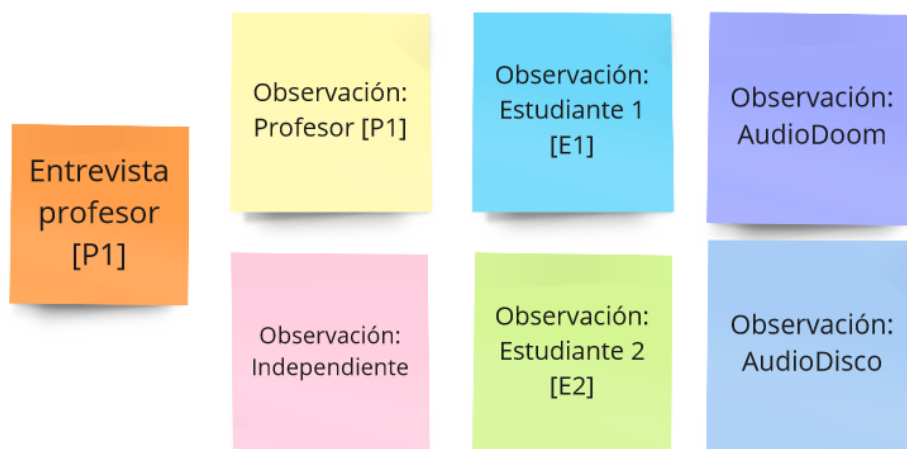


Figura 6: Colores utilizados por tema.

⁷No se mencionará ningún nombre de las personas involucradas.

3.1.2. Descubrimientos

Descubrimiento N°1: Utilizar lector de pantalla es la opción más accesible y más utilizada para interactuar con el computador.

Antes de participar en la clase computación para personas VI, estaba la expectativa de poder presenciar un computador adaptado a las necesidades de estos usuarios, con hardware y software especializado, tal como es el caso de la línea braille [Mora, 2006]. Fue una gran sorpresa ver computadores sin ningún hardware adicional, ni tampoco algún sistema operativo diferente.

A simple vista las computadoras se diferenciaban a las tradicionales por 2 motivos: No tenían ratón y tenían el monitor apagado. Mayor fue la sorpresa cuando llegaron a la clase las estudiantes E1 y E2 con sus respectivas laptops.

En materia de software, a pesar de contar con el sistema operativo Windows, sí tenían un programa corriendo desde el momento en que prendieron sus computadores: el lector de pantalla. Inmediatamente se llegó a la conclusión que este software era lo que marcaba la diferencia en las personas VI entre poder o no poder utilizar un computador.

El lector de pantalla no solamente se usa en los computadores, ya que, durante la entrevista con el profesor P1, se observó como él lo utilizaba en su teléfono celular para poder ingresar fácilmente a sus aplicaciones. Sin embargo, el lector de pantalla para el celular es muy distinto que el utilizado en el computador. Para este último, P1 comentó que anteriormente en sus clases utilizaban el lector JAWS⁸, hasta que conocieron un lector de pantalla mucho mejor.

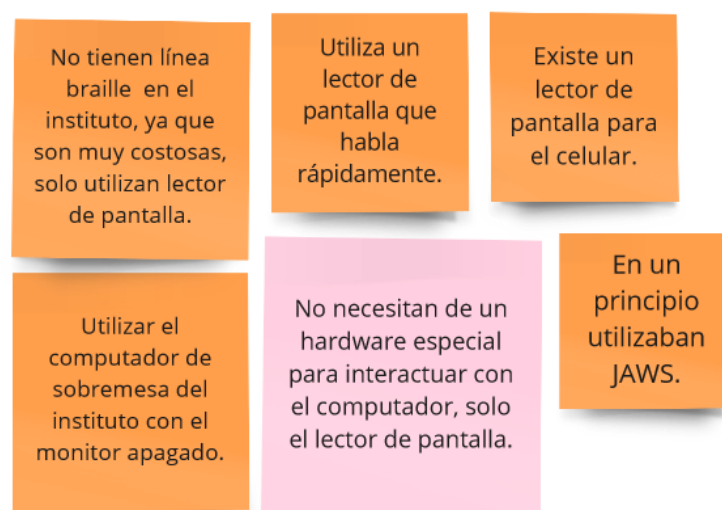


Figura 7: Descubrimientos: clúster N°1.

⁸<https://www.freedomscientific.com/products/software/jaws/>

Descubrimiento N°2: NVDA es una buena opción al momento de escoger un lector de pantalla.

Anteriormente, *P1* utilizaba el programa JAWS el cuál funcionaba bien, sin embargo al ser un programa pagado, a pesar de los esfuerzos de *P1* y el instituto, no pudieron costear la continuidad del programa, además que los equipos de computación, al ser antiguos, no ejecutan el programa JAWS correctamente.

La falta de un programa que ayudará a la clase a continuar con el aprendizaje en computación, llevó a *P1* a buscar otros softwares, hasta encontrar al lector de pantalla NVDA⁹, un programa gratuito de código abierto, que incorpora diferentes tipos de voces, además de utilizar pocos recursos del ordenador.

Este software es utilizado actualmente por *P1*, al igual que *E1* y *E2* en su día a día, además de ser el software utilizado en la clase de computación.

A pesar de que los lectores de pantalla son indispensables para la utilización de un computador, también existen otros tipos de ayuda para guiar al usuario.

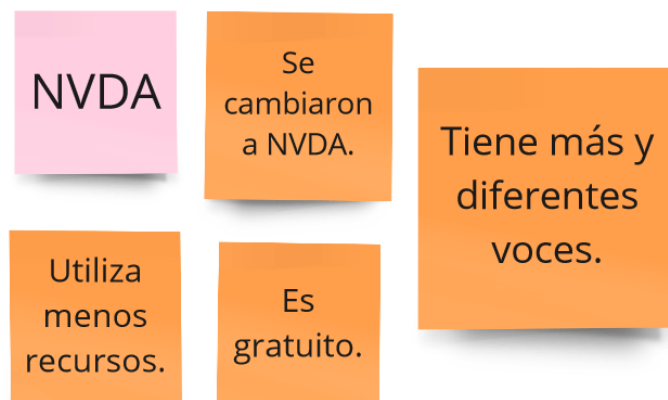


Figura 8: Descubrimientos: clúster N°2.

Descubrimiento N°3: Los indicadores auditivos hacen la diferencia.

Los indicadores auditivos no son una característica única para las personas no videntes, sino que es usado transversalmente por el computador: desde el sonido de inicio que realizaban las versiones antiguas de Windows, hasta el sonido que aparece para avisarte de un error. Estas herramientas sirven para disminuir la carga visual o auditiva que puedan presentar los usuarios.

Por el lado de una persona vidente, estos sonidos no son completamente relevantes, ya que siempre es posible disminuir la carga visual con algún símbolo, como por ejemplo una X roja

⁹<https://nvda.es/>

cuando ocurre un error o un *ticket* verde cuando todo sale bien. Sin embargo, esta ayuda visual no tiene importancia para una persona no vidente, por lo que si no existe hardware especializado, los indicadores auditivos son una de las pocas formas de aliviar la carga auditiva de estos usuarios al disminuir la cantidad de texto que debe interpretar el lector de pantalla.

Pero, ¿cómo es posible que un sonido alivie la carga auditiva? Tomando el ejemplo del error que muestra Windows cuando se realiza una acción indebida, la carga auditiva será mucho menor si el usuario escucha solamente una campana a que si el usuario escucha “acción indebida” cada vez que comete un error.

Muchos videojuegos para personas no videntes utilizan este tipo de técnica para entregarle información de manera rápida al usuario, tal como se observó con E1 al jugar AudioDoom y AudioDisco, ya que estos juegos utilizaban solamente indicadores auditivos para su jugabilidad.

Sin embargo existe una curva de aprendizaje de estos indicadores auditivos, que, aunque sea pequeña, es necesario conocerla. Ya que, a menos que exista un tutorial, el usuario tendrá que descubrir qué significa cada sonido, tal como sucedió con E2, ya que desconocía que significaba el sonido de error de Windows. Una vez aprendido el sonido, podrá relacionarlo con su significado y será utilizado para su propósito, una ayuda.

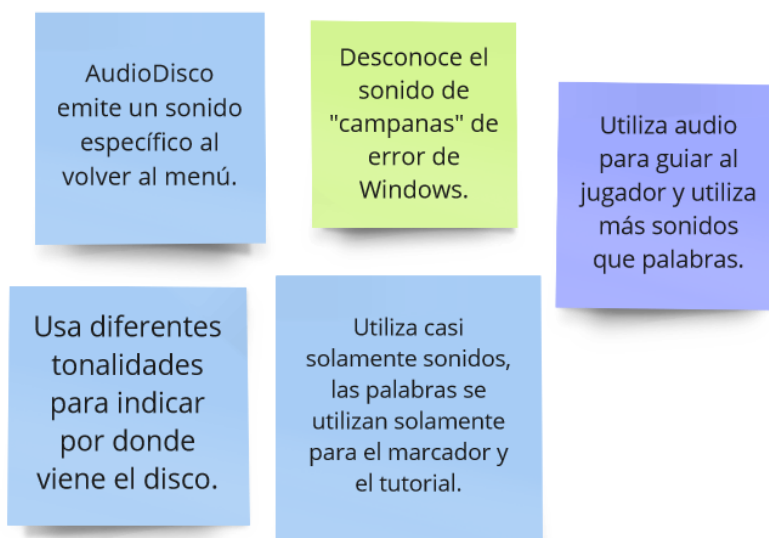


Figura 9: Descubrimientos: clúster N°3.

Descubrimiento N°4: Se valora mucho la navegación accesible.

A pesar de todos los esfuerzos que pueda realizar el usuario para poder utilizar un computador, todo puede ser en vano si la plataforma que está utilizando no está pensada para ser

accesible, ya que el usuario terminará perdiéndose en la navegación, generando una frustración. Esto es un problema común que le sucedió tanto a E1 como a E2.

Existen varias formas de desarrollar teniendo en cuenta la accesibilidad en mente, pero la más común es agregar una descripción a un objeto que el lector de pantalla no pueda leer. Por ejemplo, al desarrollar una página web es posible utilizar el atributo de HTML *aria-label* para agregar una descripción a una imagen que es interpretada únicamente por el lector para conocer su contenido, el cual no es accesible por un usuario vidente.

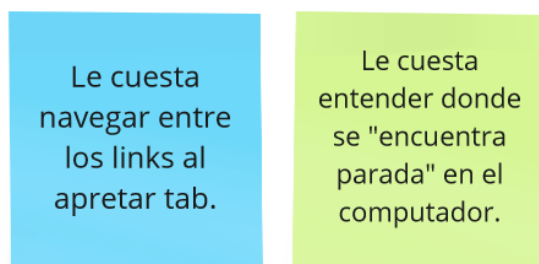


Figura 10: Descubrimientos: cluster N°4.

Descubrimiento N°5: Es común la utilización de *shortcuts*.

Una de las funciones más utilizadas por las personas VI son los atajos o *shortcuts*, los cuales se entienden como la combinación de teclas para ejecutar una acción. Las personas con discapacidad visual utilizan estos *shortcuts* una vez entendido el programa y para compensar una navegación que puede ser confusa.

Agregar *shortcuts* en el desarrollo de un programa es una forma fácil de mejorar la navegación accesible y, además, puede ser algo bueno tanto para las personas videntes como para las no videntes, ya que todos los usuarios se pueden beneficiar de estos atajos.

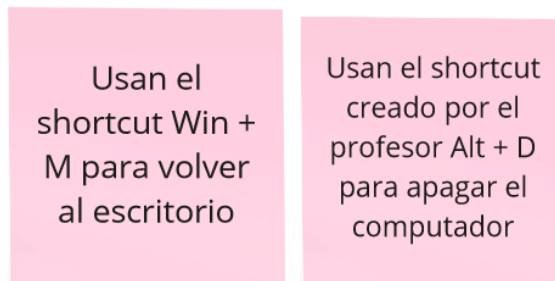


Figura 11: Descubrimientos: clúster N°5.

Descubrimiento N°6: La curva de aprendizaje es mayor para una persona con discapacidad visual.

Este último descubrimiento puede ser algo obvio, pero es el más importante a tener en cuenta al momento de desarrollar pensando en personas VI. Es el motivo por el cual todos los demás descubrimientos existen, ya que las mayorías de los programas no están hechos con la accesibilidad en mente y, por lo tanto, dificulta el aprendizaje del uso del computador.

Durante la clase fue sorprendente ver como las tareas que tenían que realizar, eran acciones mundanas que, para la mayoría de las personas videntes, pueden ser algo sencillo y rápido de realizar, pero para una persona con discapacidad visual requería de más esfuerzo, esto habla de lo resilientes que son al no quedarse atrás a pesar de que estas tecnologías no están pensadas para ellos.

Es por esto que este último descubrimiento es el más importante, ya que debe ser la guía a seguir al momento de desarrollar la plataforma deseada.

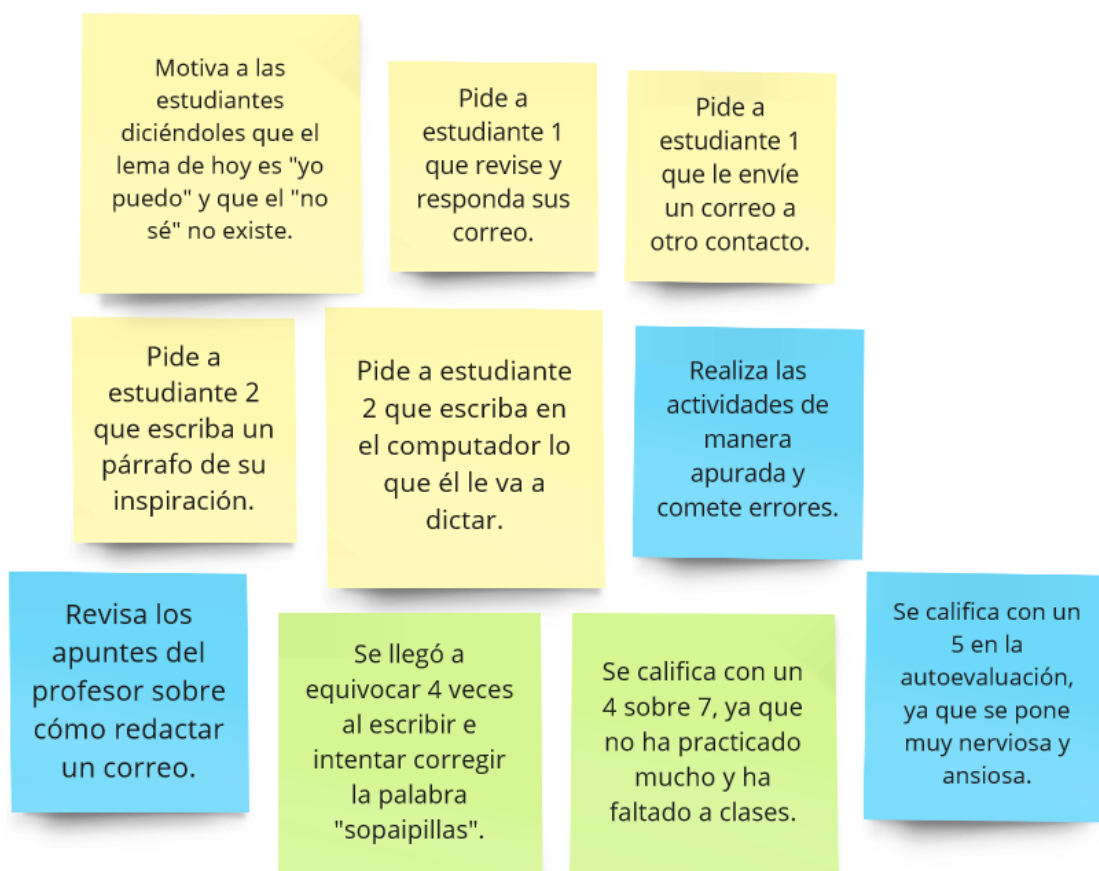


Figura 12: Descubrimientos: clúster N°6.

3.1.3. Solución escogida

Teniendo una pauta de descubrimientos, es posible realizar un descarte de soluciones para escoger la que mejor se adapte a este proyecto. Dentro de las 5 soluciones propuestas en la tabla 1 se descartan las opciones que necesiten de algún artefacto/programa distinto a un lector de pantalla, por lo que se descarta Blocks4All. También se descartan las soluciones que requieren de un poco de experiencia del usuario, ya que este programa va dirigido a principiantes, por lo que se descarta Noodle y CodeMirror-Blocks. Finalmente, las únicas 2 opciones son: Accessible Blockly y Pseudospacial Blocks, sin embargo este último se encuentra en desarrollo. Por lo que la solución escogida es **Accessible Blockly**.

Si bien su desventaja es que utiliza muchas referencias por audio y que, dada las observaciones realizadas, se concluye que esto no es realmente un impedimento para los usuarios VI, ya que si se aplica de buena manera puede generar una ayuda para disminuir la carga auditiva.

3.2. Utilización de Accessible Blockly

Accessible Blockly es una idea descartada por Google de hacer Blockly utilizable solamente por el teclado y con guías de sonido para las personas con discapacidad visual.

Google realizó diferentes versiones de Blockly: Un Blockly para poder visualizar funciones en un gráfico, una versión de Blockly para el teléfono móvil, una versión de Blockly para crear bloques personalizados, etc, etc. sin embargo, muchas de estas ideas fueron abandonadas, tal como es el caso de Accessible Blockly que fue desarrollada hasta el año 2020 y movida a un repositorio exclusivo¹⁰, para que esta pueda ser añadida al código principal de Blockly.

A pesar de ser de código abierto, poder correr Blockly fue todo un problema.

3.2.1. Ejecutando Accessible Blockly

El primer problema fue que Accessible Blockly utiliza una versión de Blockly antigua y muy diferente a la actual. Por lo que no basta con descargar Blockly y agregarle Accessible Blockly, sino que es necesario encontrar una versión compatible. La cual se encontró en la versión "2.20190722.0" de Blockly.

El segundo problema es la necesidad de utilizar una librería de Google llamada Closure-Library, la cual se discontinuó el año 2024 y que su versión actual tampoco es compatible con Accessible Blockly, por lo que se necesitó encontrar y utilizar una versión compatible, como lo es la versión "v20190121".

¹⁰<https://github.com/google/blockly-experimental>

Finalmente, después de agregar imports faltantes, copiar archivos de Accessible Blockly dentro de Blockly y ejecutar un código de Python, es posible ejecutar Accessible Blockly.

Al ejecutar Accessible Blockly, es posible encontrar una demo de un producto no terminado, pero con una idea clara y una base de código sólida, perfecta para poder trabajar.

[Blockly](#) > [Demos](#) > Accessible Blockly

This is a demo of a version of Blockly designed for screen readers, optimized for NVDA on Firefox. It allows users to create programs in a workspace by manipulating groups of blocks.

- To explore a group of blocks, use the arrow keys.
- To navigate between groups, use Tab or Shift-Tab.
- To add new blocks, use the buttons in the menu on the right.
- To delete or add links to existing blocks, press Enter while you're on that block.

Workspace

There are no blocks in the workspace. [Create new block group...](#)

[Attach new block to link...](#)

[Create new block group...](#)

[Erase Workspace](#)

[Add Variable](#)

Figura 13: Última versión obtenible de Accessible Blockly.

Fuente: <https://github.com/google/blockly-experimental>.

3.3. Nacimiento de Blockly Accesible

En honor a todo el trabajo que realizó el equipo de Google con Accessible Blockly, se decidió cambiar el nombre de este proyecto a una traducción en español directa: **Blockly Accesible**. Sin embargo, aún quedaba mucho por desarrollar para que este sea un producto terminado.

3.3.1. Lo bueno, lo malo y lo feo

Accessible Blockly entregó una base sólida, pero con muchos lugares dónde mejorar. Muchas cosas buenas, otras malas y otras feas.

Lo bueno

- Es posible crear bloques y navegar entre ellos utilizando solamente el teclado.
- Tiene indicadores auditivos para guiar al usuario.
- Tiene diferentes tipos de bloques.

Lo malo

- Está en inglés.
- Existen bloques que no funcionan.
- Es difícil de entender hasta para una persona con visibilidad.
- No tiene un tutorial, solamente instrucciones de navegación.
- A pesar de tener diferentes bloques, es una cantidad limitada a todos los bloques que permite Blockly.

Lo feo

- La única forma de vincular bloques es haciendo los siguientes pasos: 1) seleccionar el bloque que quieres que pueda vincularse. 2) Apretar el botón para vincular bloques. 3) Escoger los bloques disponibles dentro de una lista. Este proceso es demasiado enredado para una persona vidente, por lo que se asume que será aún más para una persona VI.
- Utiliza una versión antigua de Angular, específicamente Angular 2.

3.4. Implementando mejoras

Las mejoras que se implementaron se pueden dividir en 3 categorías:

1. Realizar una traducción a español.
2. Agregar un motivo por el cuál una persona VI quiera utilizar este programa.
3. Agregar un tutorial para que el usuario pueda entender cómo funciona.

3.4.1. Traducción al español

Se realizó fue una traducción de lenguaje, de inglés a español. Para esto Blockly tiene una opción para configurar el lenguaje deseado solamente exportando el lenguaje que se quiera utilizar, sin embargo, existen nuevos bloques y nuevas funcionalidades que son inherentes de Accessible Blockly, las cuales no son traducidas de manera automática con la importación de lenguaje, por lo que se tuvo que realizar una traducción manual para estos casos, sin contar la traducción de la bienvenida y las instrucciones iniciales.

Este fue el único cambio de diseño que se realizó. Es importante notar que cuando se hablan de cambios de diseño, en este caso no se refiere a un cambio de diseño estético, ya que,

como se pudo comprobar en las observaciones, la estética es lo último que se debe pensar al momento de desarrollar para personas VI, sino que el foco debe estar en cómo el usuario va a poder recibir la información a través del lector de pantalla.

Bienvenido a Blockly Accesible

Esta es una versión de Blockly diseñada para lectores de pantalla, optimizado para NVDA en Firefox. Permite a los usuarios crear programas en un espacio de trabajo manipulando grupos de bloques.

Instrucciones de navegación:

- Para explorar un grupo de bloques, use las teclas de flecha del teclado.
- Para navegar entre grupos, use Tab o Shift-Tab.
- Para agregar nuevos bloques, use los botones del menú que puede encontrar navegando con Tab.
- Para eliminar o agregar enlaces a bloques existentes, presione Enter mientras está en ese bloque.

Para comenzar, prueba el tutorial: [Aprende a utilizar Blockly Accesible](#)

Espacio de trabajo

[Mostrar código JavaScript](#) [Ejecutar código](#)

No hay bloques en el espacio de trabajo. [Crear nuevo grupo de bloques...](#)



[Vincular un nuevo bloque a la expresión...](#)

[Crear nuevo grupo de bloques...](#)

[Borrar espacio de trabajo.](#)

[Add Variable](#)

Figura 14: Primer prototipo traducido al español.

3.4.2. Agregando un propósito

Para que el usuario final se interese por un producto como este, es necesario agregarle un propósito, el cuál también debe alinearse con el objetivo general de este proyecto: Desarrollar un entorno para el aprendizaje de la programación por bloques en personas con discapacidad visual.

Lo primero que llama la atención al visualizar Blockly Accesible es la falta de un botón para ejecutar el código creado por el usuario. Es por esto que lo primero que se decidió agregar fue esta implementación así el usuario tendrá un motivo por el cual quiera programar.

También se decidió agregar una traducción del código creado por bloques a un código creado en un lenguaje de programación tradicional. Así el usuario podría conocer cómo sería su código creado por bloques, pero en otros lenguajes realmente utilizados por la industria de la programación.

En este caso se escogió utilizar una traducción a Javascript, ya que es un lenguaje muy utilizado en la programación web [Krithika, 2023] y además ya se utilizaba esta traducción para realizar la ejecución del código en bloques.

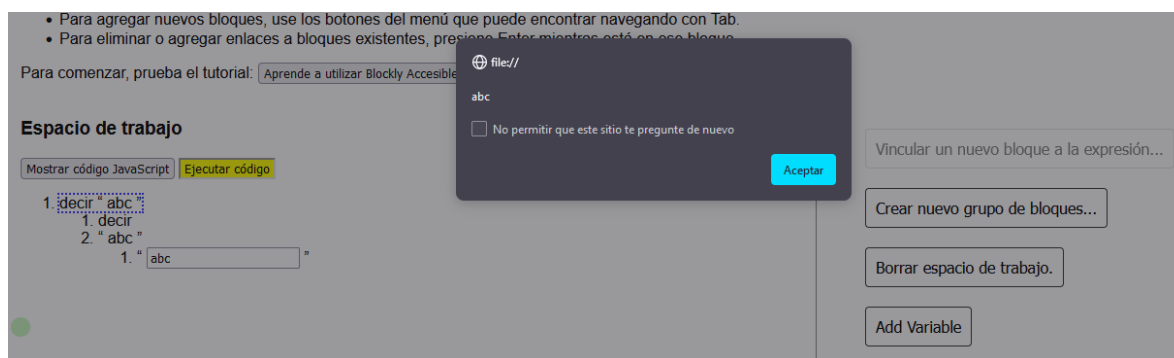


Figura 15: Ejecución de código.

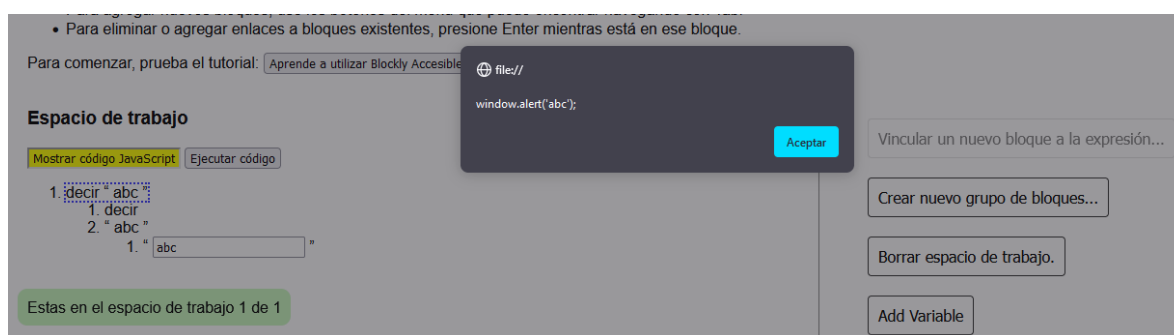


Figura 16: Traducción a Javascript.

3.4.3. Agregando un tutorial

Otra implementación importante que se decidió realizar basándose en el objetivo de “crear guías de trabajo para el prototipo”, fue la creación de un tutorial que pueda ayudar a que el usuario pueda entender cómo funciona Blockly Accesible y también pueda aprender a programar.

Estos tutoriales se realizaron con una dificultad escalonada, para que el usuario empiece aprendiendo lo básico con instrucciones simples y poco a poco pueda ir aprendiendo y creando programas más difíciles.

En un inicio se creó el primer tutorial llamado: Tutorial 1.1: Hola Mundo. Donde el usuario tendrá que crear un programa que imprima un texto en pantalla, osea un print. Para esto se le limitó la cantidad de bloques que puede escoger a solamente 1: el bloque de print. También se creó una serie de pasos que debe seguir el usuario para poder completar el tutorial.

De la misma manera se creó un segundo tutorial llamado: Tutorial 1.2: Revés, donde el usuario podrá escuchar un texto al revés, y un tercer tutorial llamado: Tutorial 1.3: Matemáticas, donde el usuario podrá verificar si un número cumple con alguna propiedad matemática.

A pesar de sonar simples, estos tutoriales tienen la finalidad de enseñar al usuario a utilizar

el programa, a unir bloques y enseñar lógica básica de programación.

Se tiene pensado realizar tutoriales por temática de programación, siendo un total de 5, donde cada temática sería:

1. Tutorial 1: Enseñará cómo crear un print para diferentes tipos de valores.
2. Tutorial 2: Enseñará cómo crear y cómo funcionan los condicionales.
3. Tutorial 3: Enseñará cómo crear y cómo funcionan las variables.
4. Tutorial 4: Enseñará cómo crear y cómo funcionan los ciclos.

Tutorial 1.1: Hola Mundo

Esta es una demostración de una versión de Blockly diseñada para lectores de pantalla, optimizado para NVDA en Firefox. Permite a los usuarios crear programas en un espacio de trabajo manipulando grupos de bloques.

- Para explorar un grupo de bloques, use las teclas de flecha.
- Para navegar entre grupos, use Tab o Shift-Tab.
- Para agregar nuevos bloques, use los botones del menú a la derecha.
- Para eliminar o agregar enlaces a bloques existentes, presione Enter mientras está en ese bloque.

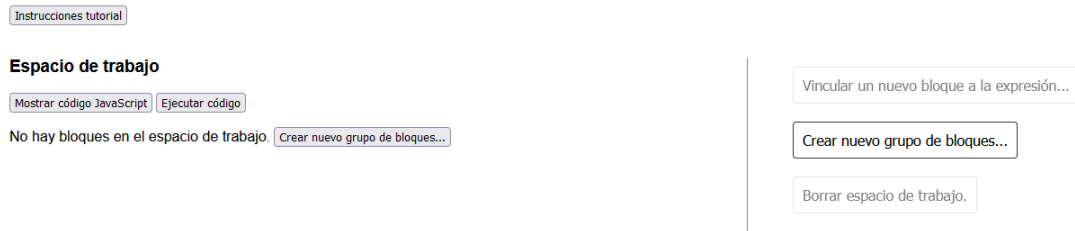


Figura 17: Pantalla principal del tutorial 1.

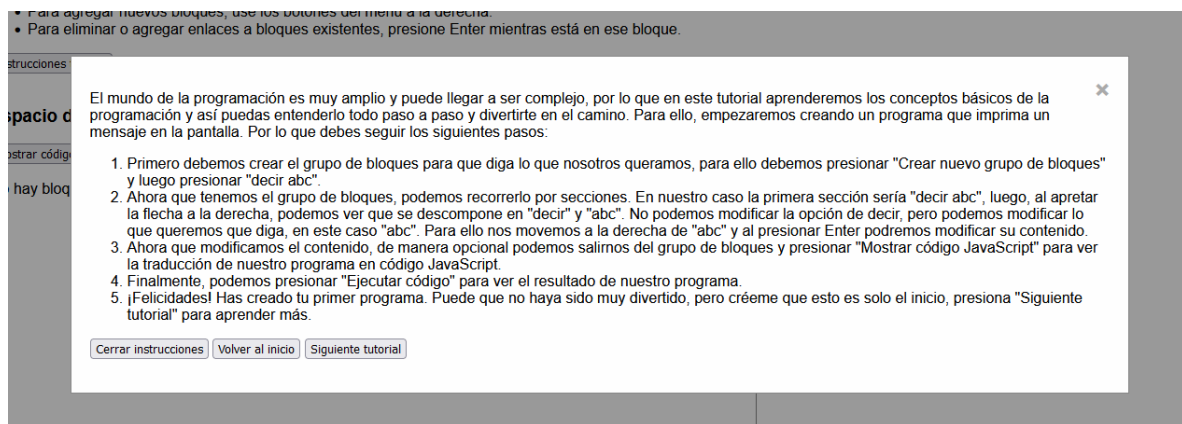


Figura 18: Instrucciones del tutorial 1.

CAPÍTULO 4

VALIDACIÓN DE LA SOLUCIÓN

4.1. Pruebas en solitario

Una vez obtenido el prototipo, es necesaria la validación de un usuario para ir mejorando y puliendo el programa. Esto cobra mayor relevancia cuando existe una gran diferencia del uso y el conocimiento de la tecnología entre el desarrollador del programa y su usuario final. Es entendible “pasar de largo” algunos detalles que, para una persona vidente, pueden ser obvios, pero para una persona con discapacidad visual no lo son.

Es por esto que se realizaron diferentes tipos de pruebas de manera solitaria para ponerse en el lugar del usuario final y entender su perspectiva. De esta forma se abarcaron algunos problemas obvios del programa que pudieron ser solucionados antes de las pruebas con los usuarios, logrando así que estas pruebas sean más enriquecedoras.

4.1.1. Prueba con NVDA y vendado

Se realizaron 2 pruebas de manera individual, la primera se realizó utilizando únicamente el lector de pantalla que se utilizaría como punto de referencia: el lector NVDA. El cuál fue seleccionado debido al “descubrimiento N°2” la sección 3.1.2. La segunda prueba se realizó con los ojos vendados y guiándose únicamente gracias al lector de pantalla NVDA, así intentar recrear de cierto modo la experiencia del usuario final y poder encontrar errores que se pasaron por alto al utilizar la vista.

Las pruebas consistieron en realizar los 3 primeros tutoriales y así poder escuchar y seguir las instrucciones por las cuales atravesaría el usuario final.

En base a estas 2 pruebas se encontraron 4 categorías de problemas:

Categoría N°1: Problemas de typo

En esta categoría se encuentran los problemas como los errores ortográficos y las palabras que permanecían sin traducción al español.

Categoría N°2: Problemas de navegación accesible

Se encontraron errores de navegación que pueden dificultar a personas VI, tales como la falta de títulos o párrafos que se repiten al inicio de distintas secciones de la plataforma, pudiendo hacer que el usuario se sienta perdido al navegar entre secciones. Además, se encontraron problemas con las instrucciones, ya que no lograban ser claras o eran muy extensas.

Categoría N°3: Errores de lectura

Durante las pruebas se reconoció un problema de lectura producido por NVDA al momento de leer algunas opciones de una lista desplegable, ya que estas opciones eran omitidas por el lector.

Categoría N°4: Problemas sin solución

Se encontraron diferentes instancias donde el lector de pantalla leía de formas no esperadas, las cuales pueden llamar la atención a un usuario principiante en lectores de pantalla, pero se desconoce si tienen el mismo efecto en usuarios acostumbrados a los lectores. Es por esto que se anotaron estas observaciones, pero no se aplicó ninguna mejora.

Los problemas de las categorías 1, 2 y 3 fueron solucionados acordemente, es posible ver el detalle de la primera y segunda prueba en las Tablas 4 y 5 respectivamente.

4.2. Preparación pruebas de usuario

Una vez realizadas las pruebas en solitario fue necesario realizar las preparaciones para las pruebas con los usuarios.

Siempre es importante no abusar del tiempo que brindan los usuarios para realizar las pruebas, por lo que estas preparaciones sirven para sacar el máximo provecho a estas instancias.

4.2.1. Supuestos y expectativas

La primera preparación que se realizó fue una tabla de supuestos y expectativas (Tabla 2), donde se colocaron los posibles problemas que podrían surgir durante las pruebas, sus posibles soluciones y las modificaciones al programa que se podrían realizar.

Posibles problemas	Posibles soluciones	Posibles modificaciones
Los usuarios no van a ser capaces de recordar las instrucciones.	Buscar la manera para que siempre pueda acceder a las instrucciones sin importar dónde se encuentren en el programa.	Combinación de botones para ingresar a las instrucciones.
Los usuarios no van a saber manejarse dentro de los menús.	Simplificar ciertas interacciones.	Que los usuarios no puedan modificar bloques, que solo puedan unirlos.
Los usuarios no podrán entender el código traducido a Javascript.	Hacer el código fácil de leer.	Cambiar de Javascript a un lenguaje con una sintaxis más entendible, como Python.

Tabla 2: Tabla de supuestos y expectativas.

4.2.2. Tareas

La segunda preparación consistió en la lista de tareas que realizaron los usuarios durante sus pruebas. Esto fue necesario para medir el desempeño y tener un flujo de usabilidad.

Las tareas fueron las siguientes:

1. Leer las instrucciones.
2. Ingresar al tutorial 1.1: Hola Mundo.
3. Encontrar las instrucciones del tutorial.
4. Seguir las instrucciones del tutorial 1.1.
5. Avanzar al siguiente tutorial.
6. Hacer el tutorial 1.2.
7. Hacer el tutorial 1.3.
8. Volver al inicio.
9. ¿Qué mejorarías?

Debido a que cada usuario avanza a su ritmo, se dedicó un rango de tiempo de entre 60 a 90 minutos para realizar cada prueba. Esto significa que, si el usuario no logra terminar los 3 tutoriales, quedarían pendientes para una siguiente ocasión.

4.2.3. Test de deseabilidad

Finalmente, la tercera preparación consistió en un test de deseabilidad [Moran, 2016] que se realizó con cada usuario después de realizar su primera prueba con el objetivo de conocer la primera impresión que les causó el programa. Este test consistió en seleccionar 5 de 25 palabras que mejor representen la sensación que tuvo el usuario al utilizar la plataforma. Originalmente el test consta de 118 palabras, pero si se consideran demasiadas, se recomienda escoger al menos 25 de estas palabras.

Las palabras fueron las siguientes:

Accesible	Atractivo	Complejo	Cómodo	Confuso
Conveniente	Aburrido	Fácil de usar	Eficiente	Amigable
Frustrante	Difícil de usar	Útil	Intimidante	Intuitivo
Organizado	Abrumador	Profesional	Novedoso	Rígido
Simple	Lento	Estable	Consume mucho tiempo	Confiable

Tabla 3: Palabras utilizadas para el test de deseabilidad

4.3. Pruebas de usuarios

Una vez realizadas las pruebas de manera solitaria y los preparativos, se realizaron las pruebas de usuarios.

En primera instancia solamente se pudo realizar pruebas con 2 usuarios, sin embargo, al final se agregó un tercer usuario para tener una mayor representatividad del usuario final.

Aún así, 3 usuarios resultan en una baja muestra para trabajar, por lo que se decidió realizar un co-desarrollo [Chathuranga, 2022] con ellos, donde después de cada prueba, se tomaría en cuenta el feedback dejado por el usuario, se desarrollarían mejoras y se volvería a realizar una prueba de usuario, haciendo que el proceso de prueba se realizara de manera iterativa. Estas mejoras fueron realizándose de manera continua.

Con respecto a la cantidad de pruebas, se realizaron no más de 2 por usuarios para disminuir la posibilidad de terminar desarrollando un programa para un usuario en específico y no para el usuario final.

4.3.1. Identificación de colores

Al igual que en la sección 3.1.2, se realizó un diagrama de afinidad en donde se le asignó un color para cada prueba, tal como se muestra en la Figura 19. Siendo los usuarios definidos

por la letra *U* y las pruebas por la letra *P*, por ejemplo “*P1 U2*” hace referencia a la prueba 1 con el usuario 2.

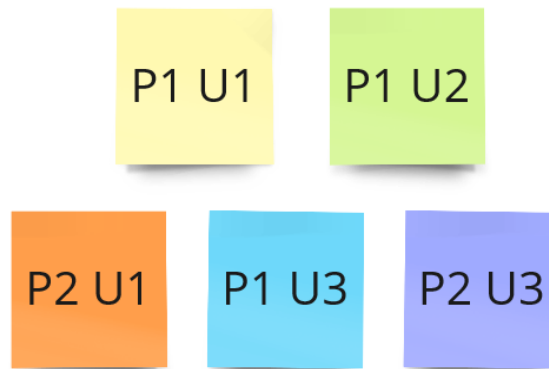


Figura 19: Colores utilizados por pruebas.

4.3.2. Conociendo a los usuarios

Antes de explicar el diagrama de afinidad y las mejoras que se realizaron en el programa, es necesario conocer a los usuarios con los que se desarrollaron las pruebas, quienes poseen un grado de discapacidad visual completa.

Usuario 1 [*U1*]:

- Tiene 35 años.
- Fue estudiante del profesor nombrado en la sección 3.1.1.
- Practica deportes como el fútbol y el goalball.
- Se maneja en la computación de manera que puede revisar correos y escuchar videos en youtube.
- Le interesa la programación y le gustaría poder trabajar de eso en un futuro.
- No sabe utilizar Excel muy bien.

Usuario 2 [*U2*]:

- Tiene 59 años.
- Fue estudiante del profesor nombrado en la sección 3.1.1.

- Anteriormente había aprendido lenguajes de programación antiguos como Cobol, Pascal, Basic, Clipper y Assembly.
- No se encuentra muy interesado en aprender un lenguaje intermedio como la programación por bloque, sino que le gustaría aprender directamente Python.
- Utiliza una versión antigua de NVDA ya que esa versión tiene diferentes voces para utilizar. Además, en su computador tiene configurado NVDA a su gusto.

Usuario 3 [U3]:

- Es el profesor nombrado en la sección 3.1.1.
- Tiene 51 años.
- Es profesor de historia y computación.
- Es Perfeccionista.

Lamentablemente, durante la realización de la prueba con *U2* fue posible notar que el usuario no representaba al usuario final, esto es debido a que el usuario ya tuvo diferentes experiencias con la programación, logrando programar en diferentes lenguajes “antiguos”, por lo que no estaba dispuesto a aprender un lenguaje intermedio como lo es la programación por bloque. Es por esto que se realizó solamente 1 prueba con este usuario.

4.3.3. Resultados

Al igual que en la sección 3.1.1, se tomaron notas de las interacciones que iban realizando los usuarios durante las pruebas, específicamente en los problemas que estos iban teniendo al interactuar con el programa.

Las notas se evaluaron de manera individual para descubrir si correspondían a la realización de una mejora en el programa o no. En esta sección se agruparán las notas en 8 categorías, las cuales se explicarán a continuación (el detalle de cada prueba se puede observar en las Tablas 6, 7, 8, 9 y 10).

Categoría N°1: Textos incorrectos

En esta categoría se encuentran los problemas que se pudieron solucionar solamente modificando texto, ya sea cambiando ciertos términos de inglés a español, disminuyendo la cantidad de instrucciones o modificando la forma en la que NVDA lee algunos textos.

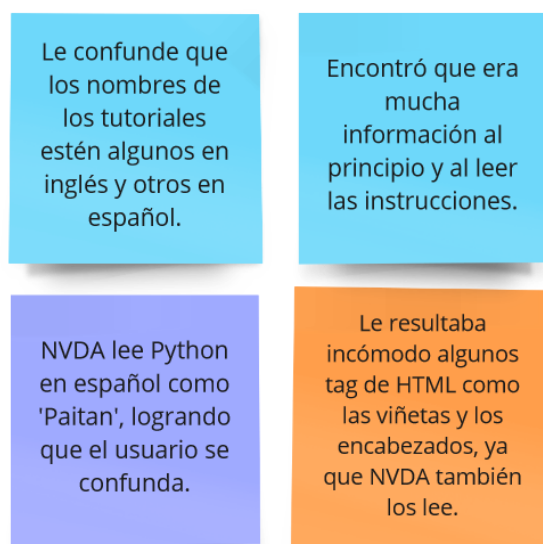


Figura 20: Pruebas de usuario: clúster N°1.

Categoría N°2: Explicaciones poco claras

Durante las pruebas fue posible notar que los usuarios tendían a perderse o no darse cuenta de lo que podían y no podían hacer. Es por esto que se agregaron explicaciones más detalladas sobre algunos temas para que el usuario sepa cuál es su alcance y donde se encontraban “parados”.

Hubo un caso particular donde el usuario no entendió el código traducido a Javascript (tal como se planteó en la tabla de supuestos 2), debido a que no se podía modificar la traducción, es por esto que se agregó la traducción a un lenguaje más amigable como Python.

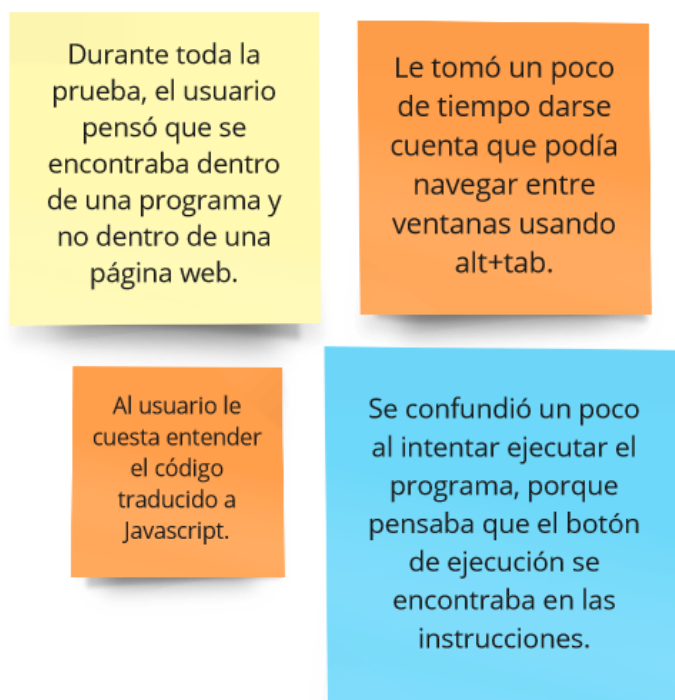


Figura 21: Pruebas de usuario: clúster N°2.

Categoría N°3: Interfaz confusa

Estos problemas son muy parecidos a los de la categoría anterior, sin embargo, su resolución no se dio al generar una mejor explicación, sino que se tuvo que cambiar la forma en la que estaba diseñada la interfaz, debido a que eran problemas ocasionados por una mala interfaz que no respetó el **descubrimiento N°4** de la sección 3.1.2.

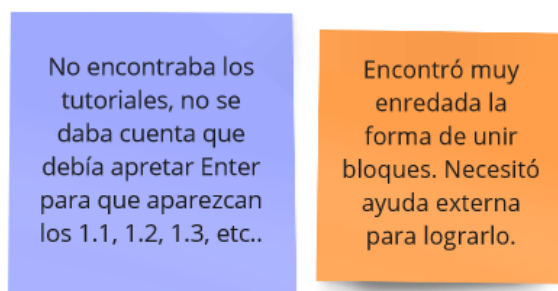


Figura 22: Pruebas de usuario: clúster N°3.

Categoría N°4: Memoria frágil

Durante el desarrollo del programa se tuvo en cuenta la creencia popular de que las personas con discapacidad visual tienen una mejor memoria que las personas videntes, sin embargo, durante las pruebas se descubrió que quizás no es así para todas las personas VI, ya que fue una sorpresa ver que los usuarios tenían el mismo tipo de memoria frágil que una persona vidente, por lo que fue necesario realizar cambios para que el usuario pudiera acceder de forma sencilla a las instrucciones y a los resultados que iba obteniendo.

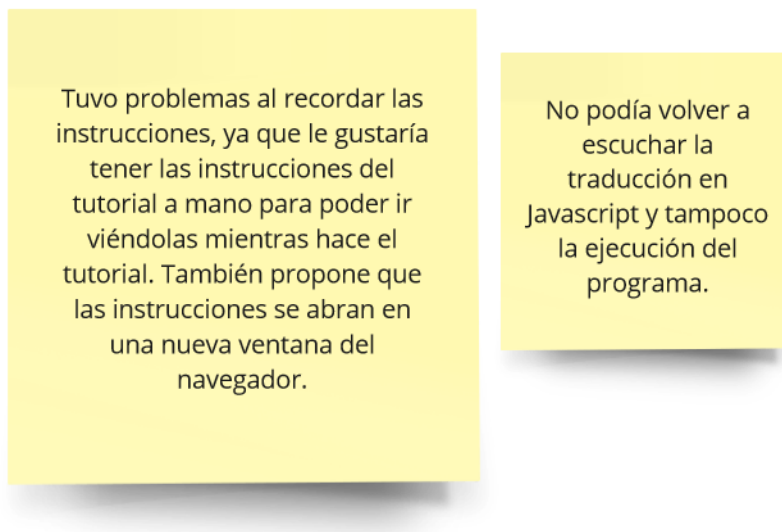


Figura 23: Pruebas de usuario: clúster N°4.

Categoría N°5: Modals

En la programación web existe un elemento llamado “modal”, que es parecido a los famosos “pop ups” que aparecen al navegar en internet. Este elemento, a diferencia del “pop up” aparece como un recuadro dentro de la página y se utilizó para mostrar las instrucciones, el manual de uso y la navegación entre tutoriales. Esto fue un error debido a que hacían pensar al usuario que se encontraban fuera de la página y, como se comentó en la categoría anterior, no eran de fácil acceso, produciendo problemas al tener que memorizar su contenido.

Debido a esto, al final todos los “modals” fueron reemplazados por una ventana nueva en el navegador.

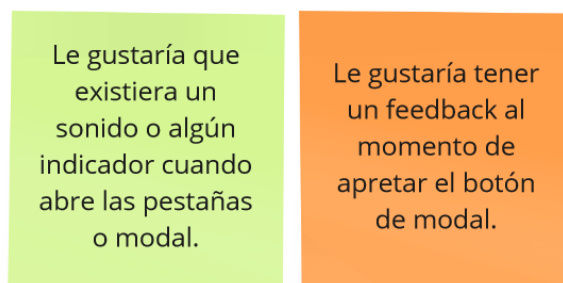


Figura 24: Pruebas de usuario: clúster N°5.

Categoría N°6: Navegación de botones

Al utilizar NVDA existen 2 formas de navegar en una página web, una es utilizando las flechas para navegar por el contenido de la página y la otra forma es utilizando las teclas “tab” y “shift+tab” para navegar entre los botones. El problema con esta última forma, es que cuando se habla de botones se refiere a todos los botones, incluyendo los del navegador. Esto produce una confusión, ya que al terminar de navegar entre los botones del programa, se empiezan a seleccionar los botones del navegador sin previo aviso.

Este problema se arregló agregaron 3 soluciones:

1. Se agregó el sufijo “último botón” al último botón del programa.
2. No se permitió acceder a los botones del navegador, generando la navegación de botones en un formato cíclico, osea después del último botón se selecciona el primero y viceversa.
3. Al pasar del primero botón al último o viceversa, se ejecuta un audio que simula el sonido al pasar una hoja de un libro, generando un indicador auditivo descrito en el **descubrimiento N°3** de la sección 3.1.2.

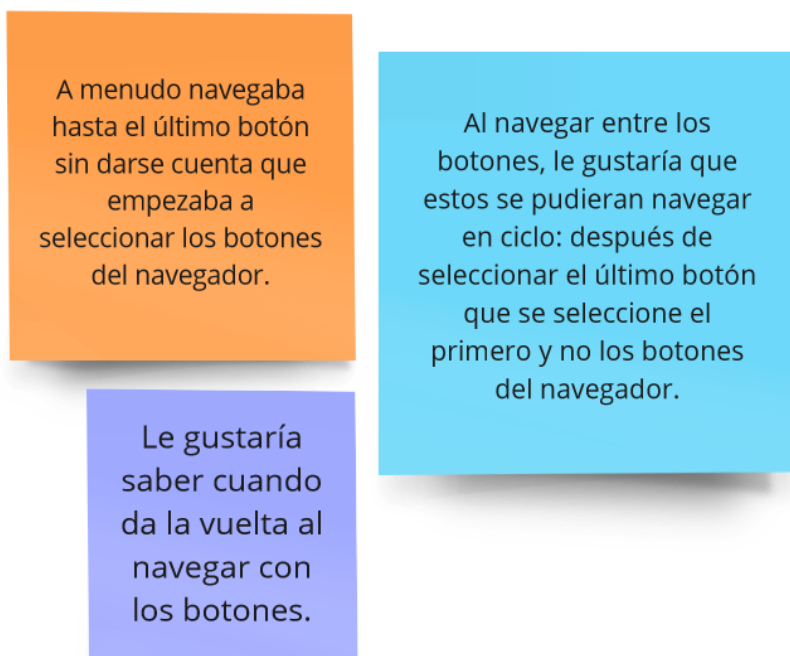


Figura 25: Pruebas de usuario: clúster N°6.

Categoría N°7: Problemas de navegación

En esta categoría entran 2 problemas muy diferentes de navegación, uno causado por un problema de Accessible Blockly al ingresar un texto el programa pedía realizar una acción ya ejecutada, y el otro causado por la falta de *shortcuts* que un usuario hizo notar.

Se corrigió el primer problema y el segundo se solucionó generando *shortcuts* para los botones principales: ejecutar código, traducción de código a Javascript, traducción a código Python e instrucciones del tutorial.

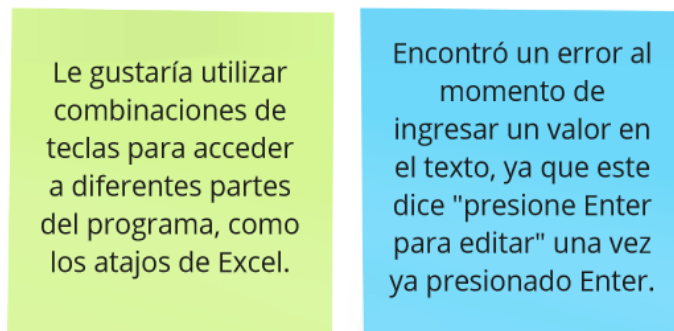


Figura 26: Pruebas de usuario: clúster N°7.

Categoría N°8: No solucionados

Por último, aquí se encuentran los problemas que no se solucionaron, ya sea porque no era un problema del programa o porque era un problema únicamente del usuario.

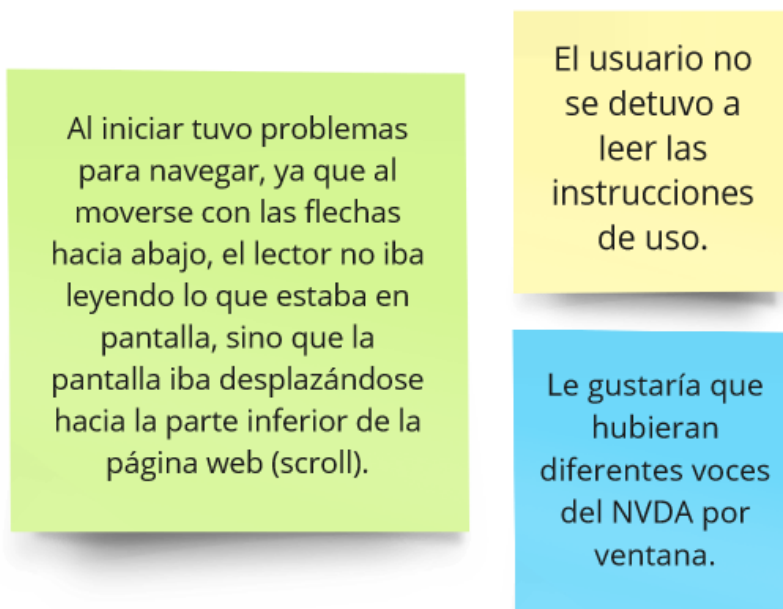


Figura 27: Pruebas de usuario: clúster N°8.

Observar a los usuarios interactuar con el programa ayudó a entender que es necesaria la ayuda de una persona vidente que guíe en los tutoriales al usuario final, ya que existen varios conceptos que pueden no entenderse de la mejor manera si son escritos como una instrucción. Es por esto que se decidió agregar una ayuda visual para guiar a la persona vidente y que esta pueda ayudar al usuario final.

Cabe destacar que esto no significa que el programa no pueda utilizarse sin una persona vidente, sino que, para que el usuario final pueda entender bien los tutoriales y, por ende, como funciona el programa, puede llegar a ser necesaria la ayuda de una persona vidente. En este caso la ayuda visual corresponde a una imagen tipo gif que muestra cómo se hace paso a paso el tutorial que debe realizar el usuario final. La cual se explica a continuación.

4.4. Ayuda visual

Debido a la complejidad que conlleva explicar los tutoriales paso a paso, se decidió utilizar una ayuda visual para que una persona vidente pueda guiar al usuario si este se encuentra “atrapado” en alguna parte del tutorial.

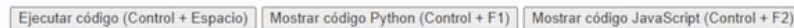
Esta ayuda visual consiste en varias imágenes tipo gif que muestran el paso a paso necesario para desarrollar el tutorial, tal como se muestra en la Figura 28. Todos los gif llevan la descripción “Este es un video para el guía.” la cual solo puede ser leído por el lector de pantalla.

3. Ahora solo nos queda modificar el contenido del texto que queremos que diga nuestro programa. Para ello debemos ingresar con la flecha derecha al bloque interno "abc", y luego presionar Enter para ingresar al modo de edición de texto y escribir lo que queramos y presionar Enter para salir.

Tutorial 1.1: Hola Mundo



Espacio de trabajo



```
1. decir "Hola mundo. "  
  1. decir  
  2. "Hola mundo. "  
    1. "Hola mundo"
```

4. Ahora que terminamos de modificar el contenido, podemos escuchar cómo sería la traducción de nuestro programa en el lenguaje Python o JavaScript presionando el botón: "Mostrar código Python" o "Mostrar código JavaScript" fuera del espacio de trabajo. Este botón lo seleccionamos con el Tab o Shift + Tab.

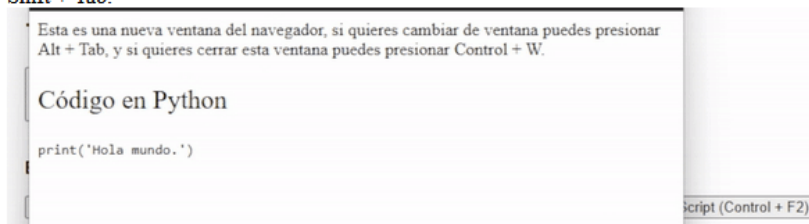


Figura 28: Ayuda visual en tutoriales.

Además, utilizando esta herramienta, se realizó una tarea por tutorial, siendo un total de 4 tareas a modo de desafío, donde el usuario no tiene ninguna indicación sobre cómo realizarlo, pero sí existen los indicadores visuales por si necesita ayuda, tal como se muestra en la Figura 29.

La tarea consiste en hacer un programa que diga la longitud de un texto.

Pista: El bloque final debe quedar como "decir longitud de 'abc'".

Tutorial 1.5: Tarea 1



Espacio de trabajo



No hay bloques en el espacio de trabajo.

Crear nuevo grupo de bloques

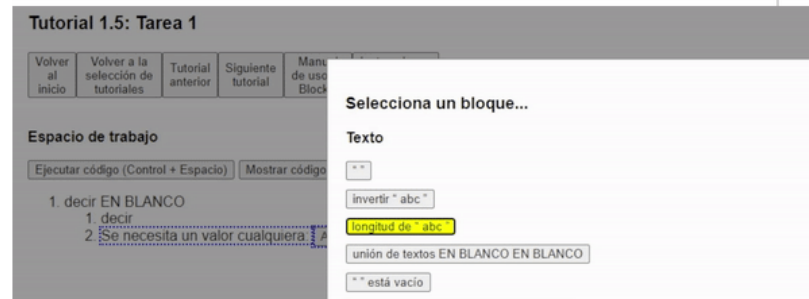


Figura 29: Ayuda visual en tareas.

4.5. Entrevista a usuarios

Una vez realizadas las pruebas de usuario, se intentó contactar al usuario 1 y 3 para realizar una entrevista sobre sus experiencias al realizar este co-desarrollo y sus experiencias con la tecnología en general. Sin embargo, solo fue posible realizar la entrevista con el usuario 3, no obstante, sus respuestas fueron bastante interesantes.

4.5.1. Estructura

La estructura de la entrevista se dividió en 3 partes:

1. Introducción: Se da un contexto sobre el motivo de la entrevista y se realizan preguntas personales para entrar en confianza con el usuario.

2. Preguntas principales: Se realizan preguntas sobre el tema en concreto, en este caso se le preguntó sobre su experiencia con el co-desarrollo y la tecnología.
3. Cierre: Se realiza la última pregunta con el objetivo de que tenga una respuesta más amplia y menos estructurada.

4.5.2. Preguntas y respuestas

A continuación, se presentan las preguntas que se le hizo al usuario y un resumen de sus respuestas:

Introducción

Ya que hemos realizado en conjunto una gran cantidad de mejoras al programa, quiero hacerle unas preguntas para conocer cómo ha sido su experiencia con este y otros programas relacionados con la tecnología. Si alguna pregunta le incomoda, puede simplemente no contestar y pasamos a la siguiente pregunta.

Pregunta 1: Su edad y a que se dedica.

Respuesta: Soy profesor de historia y geografía, tengo 51 años y me desempeño en el instituto [...] de la ciudad [...]. Si bien soy profesor de historia y geografía, la mayoría de mis horas de trabajo las imparto haciendo clases de tecnología, y para nosotros (las personas no videntes) la tecnología implica acceso al uso del celular y del computador a través de determinadas aplicaciones como los lectores de pantalla. Teniendo dominio del lector de pantalla ya puedes acceder a otras aplicaciones, porque ya tienes la base.

Pregunta 2: ¿Desde cuándo ha tenido una discapacidad visual?

Respuesta: Desde que era pequeño.

Preguntas principales

Ahora voy a empezar con preguntas relacionadas a la tecnología.

Pregunta 3: ¿Cómo ha sido su experiencia con la tecnología? ¿Se le ha dificultado?

Respuesta: De manera general, la tecnología te ayuda y te simplifica muchas cosas, pero a medida que me fui incorporando en esto (tecnología) y que se fue ampliando mi mundo profesional, me fui dando cuenta que lamentablemente no es accesible para todos, muchas cosas que quisiera hacer o que me recomiendan utilizar, resulta que no es accesible para usarse con el lector de pantalla. Por ejemplo: el microondas, el horno eléctrico, el televisor, la lavadora, el cajero automático. La mayoría de estos aparatos tienes que aprendértelos de memoria o necesitas que alguien te ayude.

Pregunta 4: ¿Usted cree que con el tiempo esto ha ido mejorando?

Respuesta: Sin duda, pero no al nivel que uno quisiera. La brecha es muy grande todavía y no veo que disminuya mucho.

Pregunta 5: ¿Cuál sería el aparato electrónico que más utiliza? ¿por qué?

Respuesta: El celular y computador, por un tema laboral.

Pregunta 6: ¿Cuál le costó más aprender a utilizarlo?

Respuesta: Me costó mucho el paso del celular analógico al teléfono inteligente. Aunque yo sabía que en algún momento tenía que migrar y sabía cómo funcionaba el otro (el inteligente), sabía que había una pantalla, que era a través de gestos, etc, pero el hecho de dar ese paso me costó mucho. Aunque hoy existen, en el mercado internacional, teléfonos inteligentes con teclas. Pero ya estamos hablando de algo más específico que requiere de un mayor costo.

Este problema lo hemos visto acá (en el instituto) con los estudiantes, sobre todo con los adultos mayores que les cuesta mucho manejar el teléfono inteligente para acceder a funciones más complejas. Utilizamos el asistente de Google para indicarle las instrucciones, pero esta función es muy limitada y ahí es donde debemos recurrir al aprendizaje del lector de pantalla.

Pregunta 7: ¿Qué otra tecnología le gustaría poder utilizar? ¿por qué?

Respuesta: Yo siempre quise haber tenido una grabadora digital, porque antes se utilizaba la grabadora con cassette, pero con el tiempo fueron reemplazadas por las grabadoras digitales, pero existía el mismo problema, porque la grabadora digital tenía una pantalla y se debía interactuar a través de los botones.

También, como te comentaba, me gustaría poder utilizar los electrodomésticos que están en la casa libremente, sin la ayuda de otra persona. También al aprenderlo a utilizar de memoria, sucede que al cambiarlo por otra marca o modelo significa hacer todo el proceso de aprendizaje de nuevo.

Con respecto al programa que fuimos realizando de manera conjunta:

Pregunta 8: ¿Cómo ha sido esta experiencia? ¿había hecho algo similar anteriormente?

Respuesta: Bastante provechoso, porque muchas veces acá (en el instituto) llegan alumnos con tu misma situación de crear proyectos, pero buscamos ideas e iniciamos un desarrollo, pero después no terminan en nada, los proyectos quedan a medias.

Pregunta 9: ¿Cree que este tipo de programa tiene el potencial para ayudar a las personas con discapacidad visual para aprender a programar?

Respuesta: Sí, sin duda. Acá hay muchos estudiantes que les gusta el área de la computación y quieren seguir estudiando en el área de la programación, entonces por ahí tendría su potencial.

Pregunta 10: ¿Y usted se lo recomendaría a estos estudiantes?

Respuesta: Sin duda, para que tengan la experiencia y sepan más o menos de que se trata (la programación), porque muchas veces uno debe pensar que la programación es algo muy simple, que uno le dice al programa las instrucciones y listo. En cambio con esto puedo decirle a los estudiantes “te comparto esta experiencia para que te des cuenta lo que vas a hacer cuando te quieras desarrollar profesionalmente”.

Cierre Ya estamos terminando con la entrevista, pero antes me gustaría preguntarle:

Pregunta 11: Cuando nos conocimos ya tenía mi proyecto ya empezado y con un enfoque ya establecido: “adaptar la programación en bloque a personas con discapacidad visual para facilitar el aprendizaje de la programación”, pero si fuera por usted, ¿cuál habría sido el enfoque de este proyecto?

Respuesta: Sinceramente, cuando me comentaste sobre la programación por bloque, yo pensé que era algo más concreto, como algún aparato conectado al computador, como algo parecido a la robótica.

Respondiendo tu pregunta, me parece que esta bien y me parece bien que tu hayas tomado elementos que ya existían y que no te hayas puesto a fabricar algo de la nada. Me refiero a que usaste un lector de pantalla y trabajaste en base al navegador.

Esto concluye la entrevista, quería darle las gracias por tomarse el tiempo y por el apoyo para hacer este proyecto realidad.

CAPÍTULO 5

CONCLUSIONES

5.1. Aprendizajes

Durante el transcurso de este proyecto, se realizaron diversos métodos o técnicas para llevar a cabo el resultado obtenido. De estos procedimientos es posible analizar y aprender de sus resultados.

5.1.1. Aprendizajes test de deseabilidad

Tal como se comentó en la sección 4.2.3, este test se realizó solamente durante la primera prueba de cada usuario, teniendo un total de 3 muestras. Se decidió hacerlo de esta manera para conocer cuál es la primera impresión que tienen los usuarios del software y así conocer cuáles serían las primeras impresiones del usuario final.

Como se puede apreciar en la Figura 30, las palabras más seleccionadas fueron “accesible” y “novedoso”, siendo seleccionadas por el 100 % de los participantes. Esto significa que los usuarios sintieron los esfuerzos realizados para que sea una plataforma accesible para personas VI y los esfuerzos por crear un producto nuevo que pocas veces se ha visto hasta el momento.

Sin embargo, no todas las palabras escogidas fueron favorables, ya que el 66 % de los usuarios también criticaron la plataforma por ser compleja, lo que se explica debido a la curva de aprendizaje que conlleva utilizar el programar, sumándole las complejidades que pueden significar adaptarlo para personas VI. Esto refuerza la necesidad de crear guías de trabajo, lo cual se logró al crear un tutorial y de crear una ayuda visual para estos tutoriales.

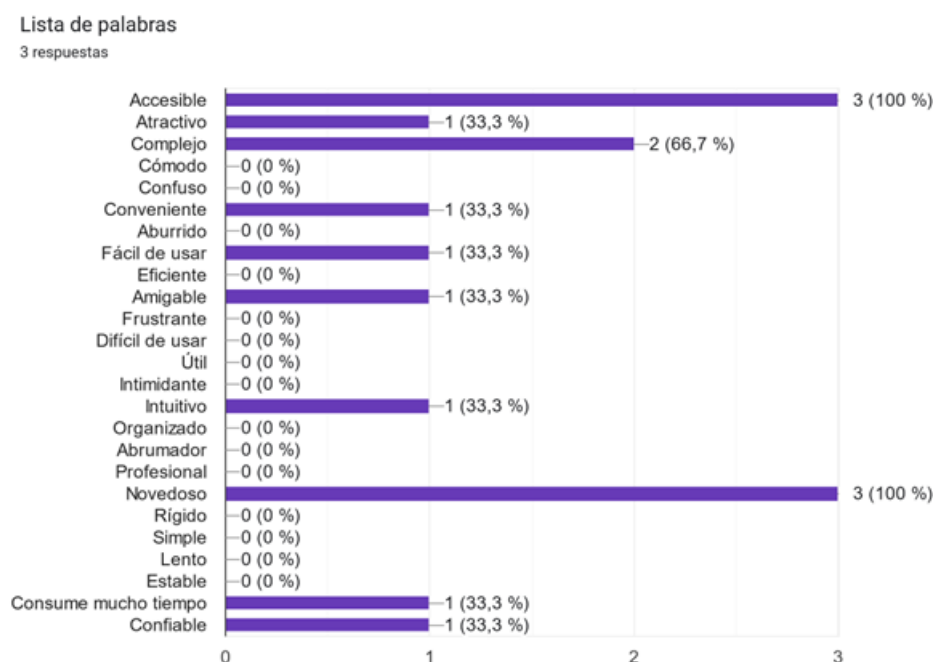


Figura 30: Resultados test de deseabilidad.

5.1.2. Aprendizajes del co-desarrollo

Debido a lo difícil que es conseguir un usuario final que contenga las características deseadas para este proyecto, si se hubieran desarrollado las pruebas de usuario sin la técnica de co-desarrollo, entonces se tendría un total de 3 pruebas, las cuales según Nielsen y Landauer no son suficientes [Nielsen, 2000], a diferencia de las 5 pruebas mínimas que se recomiendan realizar. Además, realizar un co-desarrollo ayudó a los usuarios a sentirse más involucrados en el proyecto, debido a que podían percibir y utilizar las mejoras que habían propuesto en pruebas anteriores.

Sin embargo, no todo fue beneficioso al momento de realizar co-desarrollo, ya que después de aplicar las mejoras de la última prueba no se tuvo un feedback de este proceso. Además, debido a que los usuarios no son desarrolladores, estos proponían mejoras que eran imposibles de realizar, como cambiar el tipo de voz que utiliza NVDA dependiendo de la pestaña en la que se encuentre el usuario.

De todas maneras, realizar co-desarrollo fue una buena idea, ya que tanto los usuarios como el desarrollador quedaron conformes con el resultado de la plataforma.

5.1.3. Aprendizajes durante el desarrollo

Durante el desarrollo del programa a nivel de código, ocurrieron ciertos detalles que no se tenían en cuenta al momento de programar pensando en usuarios VI y es posible que le ocurran a otros desarrolladores que quieran crear contenido accesible.

Uno de estos detalles ocurrió al conocer que NVDA lee los encabezados de manera explícita, generando disconformidad al usuario, por lo que para eliminarlos se utilizó CSS usando un “font-size” y un “font-weight” similar al encabezado que se quiere colocar. NVDA también lee los saltos de línea realizados con el tag de HTML
, por lo que tampoco se podía utilizar.

De igual forma, se intentó colocar las instrucciones como botones para que el usuario pudiera seleccionarlos individualmente con “tab” y “shift+tab”, pero NVDA los reconocía como botones sin importar las modificaciones que se realizaran.

Por último, siempre es necesario escuchar cómo va quedando el programa con el lector que se desee utilizar, ya que no siempre se lee de la manera que uno espera. Además, el lector puede interactuar de diferente manera dependiendo del navegador.

5.2. Conclusiones finales

Esta plataforma se desarrolló con la idea inicial de realizar un programa para personas con discapacidad visual que permita acercarlas al mundo de la programación a través de la programación en bloque, utilizando un programa sencillo y entretenido, utilizando Blockly como base y que pueda ser probado por el mismo tipo de usuario que utilizará el programa. Gracias a las observaciones realizadas en la clase de computación se pudo guiar el desarrollo a lo que realmente necesita y utiliza un usuario VI.

Sin embargo, al realizar las pruebas, no estaba estimada la cantidad de esfuerzo que tomaría realizar esta validación, logrando tener hasta 5 versiones distintas del producto (una por cada prueba). Este esfuerzo rindió frutos al crear una plataforma que, a pesar de no ser perfecta, cumple con todos los objetivos deseados y la aprobación de los usuarios.

Cabe destacar que esta plataforma se desarrolló con la idea de poder llegar a la mayor cantidad de personas posibles, es por esto que es completamente gratis y accesible desde cualquier navegador, aunque solamente está probado con Google Chrome. Además, el código está disponible para que cualquiera pueda descargarlo, la url es: <https://github.com/RodrigoCaya/Blockly-Accesible>.

Dentro del repositorio se encuentra la url para poder ingresar y probar la plataforma.

5.2.1. Trabajo futuro

Durante el desarrollo surgieron varias ideas que no pudieron implementarse debido al tiempo que tomó el proyecto. Sin embargo, se explicarán a continuación.

- Al abrir una ventana nueva, ya sea la de ejecutar código, traducción a otro lenguaje o la de las instrucciones del tutorial, el navegador solamente permite una ventana por tipo, por ejemplo, si se quiere ejecutar el código, pero todavía se encuentra abierta la ventana de un código anterior, el navegador no lo permitirá hasta que se cierre la primera ventana. Lo ideal sería que, al abrir una nueva ventana, la otra se cierre automáticamente o que permita más de una ventana abierta al mismo tiempo.
- Al realizar las tareas propuestas por el tutorial, no existe una comprobación de que se llegó al resultado esperado, por lo que sería ideal verificar si el resultado fue correcto o no, e informarlo al usuario.
- Al crear una variable, existen solamente 2 opciones para aplicar operaciones matemáticas en la variable, lo cual es muy reducido si vemos la cantidad de operaciones que pueden realizar los lenguajes como Python y Javascript.
- Existe una forma de comprimir Accessible Blockly para que no sea tan pesado para el navegador del usuario final, pero no se pudo aplicar esta compresión.

ANEXOS

Notas/Problemas/Errores	Soluciones/Modificaciones
Se encontraron diferentes palabras que seguían estando en inglés, tales como: “Deleted, now on workspace”, “add variable” y “selected option”.	Se tradujeron las palabras a español.
Se encontraron algunos errores ortográficos.	Se corrigieron los errores ortográficos.
Faltan links desde el inicio para el tutorial 1.2 y 1.3.	Se agregaron los links necesarios.
Algunos pasos en las instrucciones son muy largos.	Se acortaron pasos sin perder el contexto de lo que se quiere decir.
Algunas instrucciones no quedan muy claras.	Se escribieron instrucciones más explícitas para evitar confusión.
Existen ciertos casos muy específicos donde el lector NVDA no lee algunas opciones de una lista desplegable.	Se arreglaron las opciones que el lector no lograba leer.
La ejecución del código y la traducción a Javascript empiezan diciendo el nombre de la página, luego dicen “diálogo” y finalmente lo que se quiere mostrar.	No se aplicaron mejoras, ya que se desconoce si el usuario final se encuentra acostumbrado a esta lectura que da NVDA.
NVDA lee el texto hasta ciertas cantidades de palabras, luego se debe presionar la flecha hacia abajo para que siga leyendo, haciendo que algunas frases se encuentren interrumpidas.	No se aplicaron mejoras, ya que se desconoce si el usuario final se encuentra acostumbrado a esta lectura que da NVDA.

Tabla 4: Problemas y soluciones: prueba con NVDA.

Notas/Problemas/Errores	Soluciones/Modificaciones
El manual de uso, que se encuentran al inicio de los tutoriales, puede desorientar al usuario, haciéndolos pensar que siguen en la página inicial y no dentro del tutorial, ya que esta página igual contiene el manual de uso.	Se colocó el manual de uso en una estructura llamada modal, la cual solo puede ser activada presionando un botón.
Es necesario algún indicador que le diga al usuario que se encuentra dentro de las instrucciones del tutorial.	Se agregó un título para indicar que se encuentra dentro del modal de instrucciones del tutorial.
NVDA lee ciertas palabras muy juntas y pueden no entenderse.	No se aplicaron mejoras, ya que no existía certeza si estos fueron realmente problemas que podría tener el usuario final.
El lector confunde palabras en español como si estuvieran en inglés, como por ejemplo “explores” lo confunde y lo dice como la palabra en inglés “explore”.	No se aplicaron mejoras, ya que no existía certeza si estos fueron realmente problemas que podría tener el usuario final.

Tabla 5: Problemas y soluciones: prueba con NVDA y vendado.

Notas/Problemas/Errores	Soluciones/Modificaciones
Tal como se planteó en la tabla de supuestos y expectativas (tabla 2), el usuario tuvo problemas al recordar las instrucciones, ya que le gustaría tener las instrucciones del tutorial a mano para poder ir viéndolas mientras hace el tutorial. También propone que las instrucciones se abran en una nueva ventana del navegador.	Se tomó en cuenta la sugerencia del usuario y se cambia la forma de ingresar a las instrucciones, ahora se abren desde una nueva ventana del navegador, permitiendo que las instrucciones se encuentren al alcance del usuario en cualquier momento y de una manera conocida por el usuario.
Durante toda la prueba, el usuario pensó que se encontraba dentro de una aplicación de escritorio y no dentro de una página web.	Se agregó en las instrucciones de manera explícita que el usuario se encontraba dentro de una página web.
El usuario no se detuvo a leer las instrucciones de uso.	No se aplicaron mejoras, ya que el usuario pudo no haber leído las instrucciones de uso, debido a que esto no era parte del listado de tareas.
No podía volver a escuchar la traducción en Javascript y tampoco la ejecución del programa.	Se cambia la forma de ingresar a la traducción y ejecución del código, Anteriormente se visualizaba a través de un alert y ahora desde una nueva ventana del navegador, permitiendo escucharse las veces que el usuario quiera.

Tabla 6: Problemas y soluciones: primera prueba usuario 1.

Notas/Problemas/Errores	Soluciones/Modificaciones
Tal como se planteó en la tabla de supuestos y expectativas (tabla 2), al usuario le gustaría utilizar combinaciones de teclas para acceder a diferentes partes del programa, como los atajos de Excel. El usuario comenta que las personas con discapacidad visual se aprenden esas combinaciones en vez de navegar por los menús.	Se agregó la opción de ingresar a las instrucciones, la ejecución del programa y la traducción del código a Javascript, a través de combinaciones de botones (<i>shortcuts</i>).
Le gustaría que existiera un sonido o algún indicador cuando abre las pestañas o modal.	No se aplicaron mejoras, ya que no se pudo reconocer si estos eran problemas recurrentes o si sucedían solo con este usuario. De todas maneras se tomaron en cuenta por si volvían a suceder.
Al iniciar tuvo problemas para navegar, ya que al moverse con las flechas hacia abajo, el lector no iba leyendo lo que estaba en pantalla, sino que la pantalla iba desplazándose hacia la parte inferior de la página web (scroll).	No se aplicaron mejoras, ya que no se pudo reconocer si estos eran problemas recurrentes o si sucedían solo con este usuario. De todas maneras se tomaron en cuenta por si volvían a suceder.

Tabla 7: Problemas y soluciones: primera prueba usuario 2.

Notas/Problemas/Errores	Soluciones/Modificaciones
Al usuario le cuesta entender el código traducido a Javascript.	Junto con Javascript, se agregó Python como uno de los lenguajes para traducir el código debido a su simplicidad.
Le gustaría tener un feedback al momento de apretar el botón de modal.	Este punto también fue nombrado en la prueba anterior, por lo que se eliminaron todos los modals restantes y se cambiaron a una nueva ventana, ya que el lector de pantalla lee el título de la nueva ventana, dando feedback de donde se encuentra el usuario.
Le resultaba incómodo algunos tag de HTML como las viñetas y los encabezados, ya que NVDA también los lee.	Se cambiaron las viñetas por enumeración y se reemplazaron los tags de encabezados por encabezados realizados de forma manual con CSS, logrando el mismo estilo visual, pero sin que el lector de pantalla los lea.
A menudo navegaba hasta el último botón sin darse cuenta que empezaba a seleccionar los botones del navegador.	Se agregó el prefijo de “Último botón:...” al último botón del programa, de esta manera el usuario puede saber cuál es el último botón.
Le tomó un poco de tiempo darse cuenta que podía navegar entre ventanas usando alt+tab.	Se agregó de manera explícita que es posible navegar entre ventanas con alt+tab.
Tal como se comentó en sección 3.3.1, el usuario encontró muy enredada la forma de unir bloques. Necesitó ayuda externa para lograrlo.	Se eliminó la necesidad de presionar el botón extra para confirmar la vinculación y, además, se crearon bloques ya armados en las primeras etapas del tutorial, de esta manera solo debe conectar bloques en etapas más avanzadas del tutorial cuando el usuario ya maneje la estructura básica del programa.

Tabla 8: Problemas y soluciones: segunda prueba usuario 1.

Notas/Problemas/Errores	Soluciones/Modificaciones
Al navegar entre los botones, le gustaría que estos se pudieran navegar en ciclo: después de seleccionar el último botón que se seleccione el primero y no los botones del navegador.	Se consideró la sugerencia, logrando que el usuario no se pierda al momento de navegar entre los botones.
Le confunde que los nombres de los tutoriales estén algunos en inglés y otros en español.	Se arregló cambiando el nombre de la primera categoría de los tutoriales, pasó de llamarse "Tutorial 1: Print" a "Tutorial 1: Hablar".
Encontró un error al momento de ingresar un valor en el texto, ya que este dice "presione Enter para editar" una vez ya presionado Enter.	Se corrigió el error.
Se confundió un poco al intentar ejecutar el programa, porque pensaba que el botón de ejecución se encontraba en las instrucciones.	Se agregó explícitamente que se debe ejecutar el programa en una ventana diferente.
Le gustaría que hubieran diferentes voces del NVDA por ventana.	No es posible generar esta mejora, ya que el programa no interfiere con NVDA.
Encontró que era mucha información al principio y al leer las instrucciones.	Se acortó la cantidad de información y de las instrucciones.

Tabla 9: Problemas y soluciones: primera prueba usuario 3.

Notas/Problemas/Errores	Soluciones/Modificaciones
No encontraba los tutoriales, no se daba cuenta que debía apretar Enter para que aparezcan los tutoriales 1.1, 1.2, 1.3, etc..	Se cambió el formato de selección de tutoriales, ya no es a través de botones, sino que ahora es una lista desplegable (dropdown). Además se eliminó el nombre de la categoría de tutoriales, ahora se llaman "Tutorial X" a secas.
NVDA lee Python en español como 'Paitan', logrando que el usuario se confunda.	Se cambió la forma en que NVDA lee las palabras en inglés como Python y Javascript, sin cambiar como se ven escritas.
Le gustaría saber cuando da la vuelta al navegar con los botones.	Se agregó un sonido que indica cuando el usuario navega por los botones y vuelve al botón inicial.

Tabla 10: Problemas y soluciones: segunda prueba usuario 3.

REFERENCIAS BIBLIOGRÁFICAS

- [Asana, 2022] Asana, T. (2022). Cómo usar un diagrama de afinidad para organizar la información. <https://asana.com/es/resources/affinity-diagram>. [En línea; fecha de consulta 07/01/2024].
- [Bahram et al., 2019] Bahram, S., Krishnamurthi, S., y Schanzer, E. (2019). Accessible ast-based programming for visually-impaired programmers. *Association of Computing Machinery*, pp. 773–779.
- [Bigham et al., 2008] Bigham, J. P., Aller, M. B., Brudvik, J. T., Leung, J. O., Yazzolino, L. A., y Ladner, R. E. (2008). Inspiring blind high school students to pursue computer science with instant messaging chatbots. *SIGCSE Bull.*, 40:449–453.
- [Chathuranga, 2022] Chathuranga, K. (2022). Co-development: an it perspective. <https://www.amplifyn.com/post/co-development-an-it-perspective>. [En línea; fecha de consulta 05/01/2024].
- [Cheong, 2010] Cheong, C. (2010). Coding without sight: Teaching object-oriented java programming to a blind student. p. 12. Hawaii International Conference on Education.
- [de Oliveira, 2017] de Oliveira, C. C. (2017). Designing educational programming tools for the blind: mitigating the inequality of coding in schools.
- [Estudio, 2023] Estudio, T. (2023). Marco de observación aeiou. <https://torresburriel.com/weblog/marco-de-observacion-aeiou/>. [En línea; fecha de consulta 04/01/2024].
- [Francioni y Smith, 2002] Francioni, J. M. y Smith, A. C. (2002). Computer science accessibility for students with visual disabilities. *SIGCSE Bull.*, 34:91–95.
- [Guzdial, 2019] Guzdial, M. (2019). Block-based programming in computer science education. *Communications of the ACM*, pp. 22–25.
- [Hadwen-Bennett et al., 2018] Hadwen-Bennett, A., Sentance, S., y Morrison, C. (2018). Making programming accessible to learners with visual impairments: A literature review. *International Journal of Computer Science Education in Schools*.
- [Koushik y Lewis, 2016] Koushik, V. y Lewis, C. (2016). An accessible blocks language: Work in progress. *Association of Computing Machinery*, pp. 317–318.
- [Krithika, 2023] Krithika, K. L. (2023). Top 10 programming languages on github in 2023. <https://analyticsindiamag.com/top-10-programming-languages-on-github-in-2023/>. [En línea; fecha de consulta 07/01/2024].

- [Larson, 2017] Larson, Q. (2017). How blind people code. <https://www.freecodecamp.org/news/how-blind-people-code-fdb64e3bf5c/>. [En línea; fecha de consulta 07/01/2024].
- [Lewis, 2014] Lewis, C. (2014). Work in progress report: Nonvisual visual programming. *PPIG, University of Sussex*.
- [Ludi, 2015] Ludi, S. (2015). Position paper: Towards making block-based programming accessible for blind users. *Rochester Institute of Technology*, pp. 67–69.
- [Mairena, 2022] Mairena, J. (2022). Audiodisco, un pong accesible para ciegos. <http://www.videojuegosaccesibles.es/2022/04/audiodisco-un-pong-accesible-para-ciegos.html>. [En línea; fecha de consulta 04/01/2024].
- [Milne y Ladner, 2018] Milne, L. R. y Ladner, R. E. (2018). Blocks4all: Overcoming accessibility barriers to blocks programming for children with visual impairments. En *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, CHI '18, p. 1–10, New York, NY, USA. Association for Computing Machinery.
- [Mora, 2006] Mora, S. L. (2006). ¿qué es una línea o teclado braille? <https://accesibilidadenlaweb.blogspot.com/2006/12/qu-es-una-lnea-o-teclado-braille.html>. [En línea; fecha de consulta 08/01/2024].
- [Moran, 2016] Moran, K. (2016). Using the microsoft desirability toolkit to test visual appeal. *Nielsen Norman Group*. [En línea; fecha de consulta 04/01/2024].
- [Nielsen, 2000] Nielsen, J. (2000). Why you only need to test with 5 users. *Nielsen Norman Group*. [En línea; fecha de consulta 07/01/2024].
- [SANSARRICQ, 2008] SANSARRICQ, J. F. V. (2008). Movilidad 3d en usuarios finales no videntes.
- [Smith et al., 2003] Smith, A. C., Cook, J. S., Francioni, J. M., Hossain, A., Anwar, M., y Rahman, M. F. (2003). Nonvisual tool for navigating hierarchical structures. *SIGACCESS Access. Comput.*, pp. 133–139.
- [Smith et al., 2000] Smith, A. C., Francioni, J. M., y Matzek, S. D. (2000). A java programming tool for students with visual disabilities. pp. 142–148. Association for Computing Machinery.