(19) OFICINA ESPAÑOLA DE PATENTES Y MARCAS

ESPAÑA

(11) Número de publicación: **3 024 683**

(51) Int. CI.:

G06F 8/34 (2008.01)
G06F 8/33 (2008.01)

(12) TRADUCCIÓN DE PATENTE EUROPEA T3

(86) Fecha de presentación y número de la solicitud internacional: **20.03.2018 PCT/US2018/023261**

(87) Fecha y número de publicación internacional: **11.10.2018 WO18187029**

(96) Fecha de presentación y número de la solicitud europea: **20.03.2018 E 18781048 (6)**

(97) Fecha y número de publicación de la concesión europea: **07.05.2025 EP 3607431**

(54) Título: **Programación de modo mixto**

(30) Prioridad:

03.04.2017 US 201762480800 P
23.05.2017 US 201762509819 P

(45) Fecha de publicación y mención en BOPI de la traducción de la patente:
05.06.2025

(73) Titular/es:

INNOVATION FIRST, INC. (100.00%)
6725 W. FM 1570
Greenville, TX 75402, US

(72) Inventor/es:

MIMLITCH, III, ROBERT H.;
MCKENNA, JASON R.;
POPE, LEVI K.;
PEARMAN, JAMES B. y
FRIEZ, TIMOTHY S.

(74) Agente/Representante:

PONS ARIÑO, Ángel

DESCRIPCIÓN

Programación de modo mixto

Antecedentes de la invención

La programación basada en bloques es programación dentro de un entorno en donde las instrucciones se representan principalmente como bloques. La programación basada en bloques también se denomina codificación basada en bloques, modalidad de bloques, programación gráfica y lenguaje de programación visual. Los ejemplos de lenguajes de programación basados en bloques incluyen Scratch, Modkit, ROBOTC Graphical y muchos otros. Los bloques consisten normalmente en una o más declaraciones y sentencias. En la mayoría de los lenguajes de programación basados en bloques, los bloques pueden anidarse dentro de otros bloques. Los bloques son fundamentales para la programación estructurada, en la que se crean estructuras de control a partir de bloques.

El fin de los bloques en la programación es permitir que grupos de sentencias se traten como si fueran una sentencia, representada por un bloque, y reducir el ámbito léxico de variables, procedimientos y funciones declaradas dentro de un bloque. Adicionalmente, la interfaz de lenguajes de programación basados en bloques permite que se escriban programas arrastrando y soltando los bloques en un editor de una forma que representa la construcción de un rompecabezas. El objetivo de estas características es reducir la barrera de entrada para los programadores novatos a medida que comienzan a aprender ingeniería informática.

Si bien la programación basada en bloques se creó teniendo en cuenta a los programadores novatos, los profesionales de hoy en día todavía usan un lenguaje de programación basado en texto. A diferencia de arrastrar y soltar bloques, la programación basada en texto es el proceso de teclear letras, números y símbolos. La programación basada en texto, también denominada modalidad de texto, usada en lenguaje de programación tal como C, JavaScript y Python, requiere que los programadores utilicen sintaxis formal para compilar un programa.

¿Por qué es importante estudiar la modalidad de bloques? La estructuración se define por la relación entre la infraestructura de representación usada dentro de un dominio de conocimiento y el conocimiento y entendimiento que posibilita y promueve la infraestructura. Esta relación no es estática, por lo que es importante que haya un entendimiento claro de qué modalidad de bloques de aprendizaje está habilitando y promoviendo. Esta es la razón por la que estudiar la modalidad basada en bloques es tan importante.

¿Por qué la modalidad de bloques es tan popular? En un estudio de la técnica anterior en el que los estudiantes podían elegir su modalidad, los estudiantes usaron abrumadoramente la modalidad basada en bloques. La transición de bloque a texto no es un cambio en un solo sentido, sino que los estudiantes se mueven hacia delante y hacia atrás a lo largo del tiempo. Cada vez que los estudiantes querían usar un nuevo comando o un comando más complejo, usaron bloques.

La modalidad de bloques anima a los usuarios a crear, compilar y entonces probar rápidamente sus programas. La investigación se refiere a la modalidad basada en bloques como que tiene un ciclo de edición-ejecución ajustado. Esto significa recurrir más a procesos de programación productivos, debido a que se da a los estudiantes una realimentación inmediata.

En este sentido, puede imaginarse que hay bastantes entornos típicos para ayudar a los usuarios en la programación de bloques. La figura 1 de la técnica anterior muestra en la técnica anterior un ejemplo de un entorno de programación de bloques 10 típico con unos bloques 12 en la barra de herramientas izquierda 14 y un área de trabajo 16 en el lado derecho del entorno 10. Los bloques 12 en la barra de herramientas izquierda pueden arrastrarse al espacio de trabajo a la derecha. Los nuevos bloques 12" cuando se colocan normalmente se ajustan de forma forzada a los bloques 12' existentes cercanos. Una vez en el área de trabajo 16, los bloques de código forman un programa informático. La figura 1 es un ejemplo del editor Modkit para VEX que se basa en Scratch del MIT.

En otro ejemplo, la figura 2 de la técnica anterior muestra otro entorno de la técnica anterior 20, el editor y lenguaje BBC Micro:Bit que se basa en Blockly. En otro ejemplo más, la figura 3 de la técnica anterior muestra el entorno conocido 30 Blockly de Google, que incorpora una barra de herramientas 32 en el lado izquierdo del entorno 34. Blockly de Google tiene un panel intermedio 36 que contiene el área de trabajo de bloques y tiene un panel derecho 38 que contiene el código final completo que es equivalente al bloque. Por último, la figura 4 de la técnica anterior es un ejemplo del entorno de editor de texto Sublime 40 que muestra código C++.

En este contexto, Jens Mönig: "*Exploring Tables with Snap!*", 24 de febrero de 2016, recuperado de Internet: https://github.com/jmoenig/Snap-Build-Your-Own-Blocks/files/144554/TablesInSnap.pdf divulga Snap!, un entorno de programación híbrido que permite usar bloques de programación gráficos y textuales que han de usarse juntos para desarrollar un programa. Además, el documento US 2017/052767 A1 divulga diversas realizaciones para generar código de programación. En una realización, se proporciona un método, por un procesador, para generar código de programación. Una pluralidad de formas se visualiza en un dispositivo de visualización. Cada una de la pluralidad de formas está asociada con una funcionalidad respectiva de código de programación. Se recibe una entrada de usuario.

Basándose en la entrada de usuario, la pluralidad de formas se dispone en el dispositivo de visualización. Basándose en dicha disposición de la pluralidad de formas, se genera código de programación. Una funcionalidad total del código de programación generado incluye la funcionalidad respectiva del código de programación asociado con cada una de la pluralidad de formas.

Además, el documento US 7 370 315 B1 divulga un sistema de desarrollo de software que tiene un editor de código fuente sincronizado y una herramienta de modelado visual. La divulgación se dirige a un entorno de desarrollo integrado en donde hay un acoplamiento estrecho entre una superficie de diseño que proporciona una representación visual de las diversas entidades físicas y lógicas en un modelo de software y las estructuras de código subyacentes que soportan las entidades. El modelo es una representación gráfica del código real. Esto permite una actualización bidireccional, es decir, el modelo se actualiza cuando el programador cambia el código y viceversa.

Sin embargo, existen desafíos en el enfoque de programación anterior, especialmente al realizar una transición entre la modalidad de bloques y la modalidad de texto. Algunos de estos desafíos son:

- Legibilidad -- Los estudiantes encuentran mucho más fácil leer y entender un bloque de código cuando se escribe en modalidad de bloques, un lenguaje más "natural".
- Memorización de comandos -- en la modalidad de bloques, los novatos pueden explorar todos los comandos disponibles, pero en la modalidad de texto, los estudiantes a menudo tienen que memorizar comandos. Adicionalmente, hay normalmente una biblioteca más grande de comandos de texto.
- Memorización de Sintaxis -- en la modalidad de texto, los estudiantes tienen que memorizar la sintaxis.
- Flexibilidad de Entrada -- se requiere teclear/deletrear en la modalidad de texto, pero no en la modalidad de bloques.
- Prototipos frente a definición -- en la modalidad de bloques, cuando un usuario arrastra un comando a un bucle, usa una función (myblocks en Scratch o Line Following en ROBOTC Graphical) o hace algo más avanzado, como radiodifundir un mensaje, el usuario solo tiene que manipular los parámetros dentro de ese bloque. En la modalidad basada en texto, los programadores tienen que construir toda esta sintaxis por su cuenta. Esto incluye el orden de los comandos, qué sintaxis usar con esos comandos, tipos de variables coincidentes, etc.
- Identificadores coincidentes -- las modalidades de bloques manejan esto para los usuarios, normalmente con menús desplegables, mientras que en la modalidad de texto, el usuario tiene que establecer la coincidencia por su cuenta.
- Definir el ámbito -- que falten o que sobren paréntesis en lenguajes que definen el ámbito con símbolos explícitos es uno de los errores más comunes con los programadores principiantes. La investigación dice que saber cómo controlar el ámbito no es suficiente; la mecánica de organizar y mantener el ámbito sigue siendo un desafío.
- Expresiones de escritura -- se proporcionan expresiones para estudiantes en modalidad de bloques. Tienen que escribirse, y escribirse correctamente, en modalidad de texto.
- Tipos de datos -- los estudiantes no tienen que usar o entender tipos de datos con modalidad de bloques.
- Mensajes de error -- con la modalidad de bloques, los estudiantes obtienen poca práctica con la interpretación de mensajes de error.
- Formateo -- con la modalidad de bloques, los estudiantes no tienen que usar administrar su distribución con cosas como espacios en blanco y sangría.
- Enfoque de Programación Mejorado -- la modalidad de bloques fomenta un enfoque de programación de abajo arriba (identificar herramientas de nivel inferior que se pueden componer para volverse un programa más grande). A los estudiantes les va peor en las preguntas basadas en lógica en el examen de ingeniería informática AP que en cualquier otro tipo de pregunta.
- Entendimiento mejorado -- los estudiantes no tienen un entendimiento profundo de cómo funcionan los bucles, qué son las variables y qué hacen en un contexto de programación después de programar en modalidad de bloques. Los bloques ayudan a los estudiantes a entender qué hace un comando, no a cómo transferir ese conocimiento a diferentes aplicaciones.
- Sintaxis -- los estudiantes aún tienen que lidiar con la sintaxis cuando experimentan la modalidad de texto por primera vez. El uso de la modalidad de bloques no resuelve este problema de transición.

Además, existen varios problemas cuando se usan variables, expresiones y bucles que se han informado previamente. Específicamente, los estudiantes tienen problemas con la asignación de variables que necesitan adoptar múltiples valores al mismo tiempo. Distinguir entre lo que va dentro de un bucle y lo que precede o sigue a un bucle, y que una expresión que implica la variable de control de un bucle puede tener diferentes valores en cada ciclo del bucle provoca problemas.

Sumario de la invención

Como se ilustra mediante los dibujos y se proporciona mediante la divulgación, de acuerdo con una o más de las realizaciones de la invención, hay un sistema como se define en la reivindicación 1.

De acuerdo con este sistema, las instrucciones de programa, que están configuradas para crear el entorno de codificación gráfica, definen adicionalmente un conjunto de instrucciones, almacenadas en la memoria, para definir un modo de edición de variables. Además, uno o más de los bloques de programación gráfica, de la pluralidad de bloques

de programación gráfica, está configurado para incluir un elemento variable establecido por un usuario que activa el modo de edición de variables.

En otro aspecto de este sistema, las instrucciones de programa configuradas para crear el entorno de codificación gráfica definen adicionalmente un conjunto de instrucciones, almacenadas en la memoria, para codificar por color dos o más bloques de programación gráfica con diferentes colores predefinidos.

En otro aspecto de este sistema, el conjunto de instrucciones de código de colores está configurado además para codificar por colores el lenguaje de programación textual en la ventana de visualización adyacente a los uno o más bloques de programación gráfica de tal modo que el color del lenguaje de programación textual coincide con el color del bloque de programación gráfica.

En otro aspecto de este sistema, las instrucciones de programa configuradas para crear el entorno de codificación gráfica definen adicionalmente un conjunto de instrucciones, almacenadas en la memoria, para definir un modo de conversión. El modo de conversión está configurado para que un usuario seleccione uno o más bloques de programación gráfica. Tras la activación, el conjunto de instrucciones para el modo de conversión está configurado para: convertir los uno o más bloques de programación gráfica seleccionados en un lenguaje de programación textual convencional, y crear uno o más bloques de programación de codificación equivalentes a los uno o más bloques de programación gráfica seleccionados, y en donde los uno o más bloques de programación de codificación son accesibles para editar con lenguaje de codificación textual convencional.

En otro aspecto de este sistema, las instrucciones de programa configuradas para crear el entorno de codificación gráfica definen adicionalmente un conjunto de instrucciones, almacenadas en la memoria, para definir un bloque de argumento de programación gráfica para su uso en la creación del programa gráfico. El bloque de argumento de programación gráfica está configurado como un bloque gráfico con un segmento de argumento embebido dentro del bloque gráfico, y el conjunto de instrucciones configurado adicionalmente para aceptar lenguaje de codificación textual convencional en el segmento de argumento.

En otro aspecto de este sistema, las instrucciones de programa configuradas para crear el entorno de codificación gráfica definen adicionalmente un conjunto de instrucciones, almacenadas en la memoria, para convertir automáticamente un bloque gráfico a lenguaje de codificación textual convencional, e insertar el lenguaje de codificación textual convencional en el bloque de programación de codificación definido en el programa gráfico, en una posición definida por un usuario y dentro del bloque de programación de codificación, cuando el bloque gráfico es seleccionado por un usuario y el usuario define dicha posición para la inserción.

En otro aspecto de este sistema, las instrucciones de programa configuradas para crear el entorno de codificación gráfica definen adicionalmente un conjunto de instrucciones, almacenadas en la memoria, para fusionar bloques. Estando configurado el conjunto de instrucciones para fusionar bloques para añadir automáticamente un segundo bloque de programación de codificación a un primer bloque de programación de codificación, definiendo un bloque de programación de codificación fusionado que comprende: tanto un segundo lenguaje de codificación textual convencional definido por el segundo bloque de programación de codificación; como un primer lenguaje de codificación textual convencional definido por el primer bloque de programación de codificación. En otro aspecto de este sistema, el conjunto de instrucciones para fusionar bloques está configurado adicionalmente para insertar el segundo lenguaje de codificación textual convencional en una posición dentro del primer lenguaje de codificación textual convencional seleccionado por un usuario.

En otro aspecto de este sistema, las instrucciones de programa configuradas para crear el entorno de codificación gráfica definen adicionalmente un conjunto de instrucciones, almacenadas en la memoria, para identificar errores en el bloque de programación de codificación. Estando configurado el conjunto de instrucciones para identificar errores para comprobar un lenguaje de codificación textual convencional definido por usuario dentro del bloque de programación de codificación para determinar si el programa de salida puede ejecutarse apropiadamente para controlar o bien el objeto virtual o bien un objeto físico de acuerdo con la funcionalidad definida por usuario. Además, el conjunto de instrucciones para identificar errores cambia automáticamente el color del lenguaje de codificación textual convencional definido por usuario cuando el programa de salida no logra ejecutarse apropiadamente. En otro aspecto de este sistema, el conjunto de instrucciones para identificar errores está configurado adicionalmente para cambiar el color del lenguaje de codificación textual convencional definido por usuario antes de que se genere el programa de salida.

En otro aspecto de este sistema, el entorno de codificación gráfica incluye además un conjunto de instrucciones para definir un modo de programación mixto. El modo de programación mixto está configurado para crear un entorno de programación de texto dentro del entorno de codificación gráfica, en donde el entorno de programación de texto define líneas de texto de codificación, recibe bloques de programación gráfica y recibe lenguaje de codificación textual convencional en las líneas de texto de codificación. Además, el modo de programación mixto se configura adicionalmente para crear el programa gráfico en respuesta a una entrada de usuario, en donde el programa gráfico comprende, en respuesta a la entrada de usuario, al menos un bloque de programación gráfica y un lenguaje de codificación textual convencional interconectados en el entorno de codificación gráfica que indica visualmente una

funcionalidad del programa gráfico de acuerdo con la entrada de usuario.

Breve descripción de los dibujos

Para un entendimiento más completo de la naturaleza de la presente invención, se debe hacer referencia a la siguiente descripción detallada tomada junto con los dibujos adjuntos, en los que:

la figura 1 es una ilustración de la técnica anterior de un ejemplo de un entorno de programación de bloques desarrollado por el editor Modkit basándose en la programación de Scratch por el Instituto de Tecnología de Massachusetts;
la figura 2 es otra ilustración de la técnica anterior de un ejemplo de editor y lenguaje BBC Micro:Bit basándose en Blockly;
la figura 3 es otra ilustración más de la técnica anterior de un ejemplo de Blockly de Google;
la figura 4 es un ejemplo de ilustración de la técnica anterior del editor de texto Sublime que muestra codificación en C++;
la figura 5A es un programa de bloques hecho con Bloques Gráficos;
la figura 5B muestra un programa de acuerdo con una realización de la invención que es equivalente al de la figura 5A;
la figura 6A muestra otro programa hecho con Bloques Gráficos;
la figura 6B muestra un programa de acuerdo con una realización de la invención que es equivalente al de los Bloques Gráficos de la figura 6A;
la figura 7 muestra un programa compuesto por Bloques Gráficos y el comienzo de un Bloque de Código sin ningún código introducido;
la figura 8A muestra otro programa hecho con Bloques Gráficos;
la figura 8B muestra un programa de acuerdo con una realización de la invención, que incluye Bloques Gráficos y un Bloque de Codificación que es equivalente al de Bloques Gráficos específicos de la figura 8A;
la figura 8C muestra un programa de acuerdo con una realización de la invención, que incluye un Bloque Gráfico y un Bloque de Codificación que es equivalente al de Bloques Gráficos específicos de la figura 8A;
la figura 8D muestra un programa de acuerdo con una realización de la invención, que incluye un Bloque de Codificación que es equivalente al de todos los Bloques Gráficos de la figura 8A;
las figuras 9A - 9E ilustran una Función de Vistazo que permite que un usuario vea rápidamente un Bloque Gráfico como un Bloque de Código;
las figuras 10A - 10D ilustran una Función de Conversión de Bloque que convierte un Bloque Gráfico en un Bloque de Código dentro del programa de Bloques Gráficos;
las figuras 11A - 11C ilustran una plantilla de pantalla que permite la adición de nuevos Bloques de Código en Blanco en un desde el Área de Caja de Herramientas en un Área de Trabajo;
las figuras 12A - 12B muestran una Característica de Editor en un Bloque de Código con una Función de Colocar Cursor Sobre para que los usuarios entiendan mejor las sentencias en C en Bloques de Codificación;
la figura 13 ilustra un Bloque de Código que incluye una ventana de argumento para introducir texto, para permitir que los usuarios expresen fórmulas en un formato de código;
las figuras 14A y 14B ilustran en una realización de la invención la capacidad de cambiar el Área de Barra de Herramientas izquierda de bloques a fragmentos de texto;
las figuras 15A - 15D ilustran una función para añadir Bloques Gráficos en un Bloque de Código Existente convirtiendo automáticamente el Bloque Gráfico en texto durante la inserción;
la figura 16 ilustra tipos de datos;
la figura 17 ilustra la conversión de Bloque Gráfico a Bloque de Código;
la figura 18 muestra la fusión de dos Bloques de Código en un único Bloque de Código resultante;
las figuras 19A - 19F en una o más realizaciones de la presente invención indicadores de error para permitir la corrección de codificación en el programa desarrollado;
las figuras 20A-20B muestran una función de avance de acuerdo con una o más de las realizaciones de la invención;
la figura 21 ilustra diversos lenguajes de codificación que el usuario puede utilizar en la programación de aplicaciones;
las figuras 22A - 22C ilustran la capacidad de contraer y expandir bloques de código en el entorno de programación;
las figuras 23A - 23B muestran un editor de programación visual de acuerdo con una realización de la invención
la figura 24A muestra código como un ejemplo de un programa completamente textual;
la figura 24B ilustra una realización alternativa de la programación de modo mixto usando un entorno de programación textual mixto que incluye un programa que es idéntico al programa completamente textual de la figura 24A;
la figura 24C muestra el mismo programa que en la figura 24B, excepto que el usuario ha hecho clic en el texto para insertar una nueva línea vacía;
la figura 24D muestra al usuario borrando una línea de texto;
la figura 24E muestra un nuevo programa que se está desarrollando en el entorno de programación textual mixto;
la figura 25 ilustra un sistema informático configurado para ejecutar un entorno de programación gráfica de modo mixto de acuerdo con una o más realizaciones de la presente invención;
la figura 26 ilustra un sistema de red que comprende dos o más sistemas informáticos que pueden implementar un entorno de programación gráfica de modo mixto de acuerdo con una o más realizaciones de la presente invención;
la figura 27 es un diagrama de bloques de alto nivel de un sistema ilustrativo que puede ejecutar o utilizar un entorno de programación gráfica de modo mixto de acuerdo con una o más realizaciones de la presente invención;

la figura 28 ilustra un sistema ilustrativo que puede realizar funciones de control y/o simulación utilizando un entorno de programación gráfica de modo mixto de acuerdo con una o más realizaciones de la presente invención; y
la figura 29 es un diagrama de bloques ilustrativo de los sistemas informáticos de las figuras 25 - 28.

Explicación detallada de la invención y nuevo método

Aunque la invención es susceptible de realizaciones en muchas formas diferentes, se muestran en los dibujos y se describirán en el presente documento, con detalle, las realizaciones preferidas de la presente invención.

Lo siguiente es un glosario de términos usados en la presente solicitud:

Medio de memoria: cualquier forma de diversos tipos de dispositivos de memoria o dispositivos de almacenamiento y está destinado a incluir un medio de instalación; una memoria de sistema informático o memoria de acceso aleatorio tal como DRAM, DDR RAM, SRAM, EDO RAM, Rambus RAM, etc.; una memoria no volátil tal como Flash, medios magnéticos, por ejemplo, un disco duro o almacenamiento óptico; registros u otros tipos similares de elementos de memoria, etc. Además, el medio de memoria puede ubicarse en un primer ordenador en el que se ejecutan los programas, o puede ubicarse en un segundo ordenador diferente que se conecta al primer ordenador a través de una red, tal como Internet. En el último caso, el segundo ordenador puede proporcionar instrucciones de programa al primer ordenador para su ejecución. La expresión "medio de memoria" puede incluir dos o más medios de memoria que pueden residir en diferentes ubicaciones, por ejemplo, en diferentes ordenadores que se conectan a través de una red.

Medio portador - un medio de memoria, así como un medio de transmisión físico, tal como un bus, una red y/u otro medio de transmisión físico que soporta señales tales como señales eléctricas, electromagnéticas o digitales.

Elemento de hardware programable: incluye diversos dispositivos de hardware que comprenden múltiples bloques de función programables conectados a través de una interconexión programable. Un elemento de hardware programable también puede denominarse "lógica reconfigurable". Programa de software: se pretende que el término "programa de software" tenga toda la amplitud de su significado ordinario, e incluye cualquier tipo de instrucciones de programa, código, secuencia de comandos y/o datos, o combinaciones de los mismos, que pueden almacenarse en un medio de memoria y ser ejecutados por un procesador. Los programas de software ilustrativos incluyen programas escritos en lenguajes de programación basados en texto, tales como lenguaje C, C++, PASCAL, FORTRAN, COBOL, JAVA, ensamblador, etc.; programas gráficos (programas escritos en lenguajes de programación gráficos); programas en lenguaje ensamblador; programas que se han compilado en lenguaje de máquina; secuencias de comandos; y otros tipos de software ejecutable. Un programa de software puede comprender dos o más programas de software que interoperan de alguna forma. Obsérvese que diversas realizaciones descritas en el presente documento pueden implementarse por un ordenador o programa de software. Un programa de software puede almacenarse como instrucciones de programa en un medio de memoria.

Programa: se pretende que el término "programa" tenga toda la amplitud de su significado ordinario El término "programa" incluye 1) un programa de software que puede almacenarse en una memoria y es ejecutable por un procesador o 2) un programa de configuración de hardware utilizable para configurar un elemento de hardware programable.

Programa gráfico - Un programa que comprende una pluralidad de nodos o iconos interconectados, en donde la pluralidad de nodos o iconos interconectados indican visualmente una funcionalidad del programa. Los nodos o iconos interconectados son código fuente gráfico para el programa. Los nodos de función gráfica también pueden denominarse bloques funcionales, o simplemente bloques.

Lo siguiente proporciona ejemplos de diversos aspectos de programas gráficos. Los siguientes ejemplos y análisis no pretenden limitar la definición anterior de programa gráfico, sino proporcionar ejemplos de lo que abarca la expresión "programa gráfico":

Los nodos en un programa gráfico pueden conectarse en uno o más de un formato de flujo de datos, flujo de control y/o flujo de ejecución. Los nodos también pueden conectarse en un formato de "flujo de señal", que es un subconjunto de flujo de datos.

La expresión "programa gráfico" incluye modelos o diagramas de bloques creados en entornos de modelado gráfico, en donde el modelo o diagrama de bloques comprende bloques interconectados (es decir, nodos) o iconos que indican visualmente la operación del modelo o diagrama de bloques.

Un programa gráfico puede representarse en la memoria del sistema informático como estructuras de datos y/o instrucciones de programa. El programa gráfico, por ejemplo, estas estructuras de datos y/o instrucciones de programa, pueden compilarse o interpretarse para producir lenguaje de máquina que logra el método o proceso deseado como se muestra en el programa gráfico.

Los datos de entrada a un programa gráfico pueden recibirse desde cualquiera de diversas fuentes, tal como desde un dispositivo, una unidad bajo prueba, un proceso que se está midiendo o controlando, otro programa informático, una base de datos o desde un archivo. Asimismo, un usuario puede introducir datos en un programa gráfico o instrumento virtual usando una interfaz gráfica de usuario, por ejemplo, un panel frontal.

Un programa gráfico puede tener opcionalmente una GUI asociada con el programa gráfico. En este caso, la pluralidad de bloques o nodos interconectados se denomina a menudo porción de diagrama de bloques del programa gráfico.

Nodo: en el contexto de un programa gráfico, un elemento que puede incluirse en un programa gráfico. Los nodos de programa gráfico (o simplemente nodos) en un programa gráfico también pueden denominarse bloques. Un nodo puede tener un icono asociado que representa el nodo en el programa gráfico, así como código y/o datos subyacentes que implementan la funcionalidad del nodo. Los nodos (o bloques) ilustrativos incluyen nodos de función, nodos de subprograma, nodos de terminal, nodos de estructura, etc. Los nodos pueden conectarse entre sí en un programa gráfico mediante iconos o hilos de conexión.

Programa de Flujo de Datos - Un Programa de Software en el que la arquitectura de programa es la de un grafo dirigido que especifica el flujo de datos a través del programa, y por lo tanto las funciones se ejecutan siempre que están disponibles los datos de entrada necesarios. Los programas de flujo de datos pueden contrastarse con programas procedimentales, que especifican un flujo de ejecución de cálculos que hay que realizar. Como se usa en el presente documento, "flujo de datos" o "programas de flujo de datos" se refiere a "flujo de datos planificado dinámicamente" y/o "flujo de datos definido estáticamente".

Programa de Flujo de Datos Gráfico (o Diagrama de Flujo de Datos Gráfico) - Un Programa Gráfico que también es un Programa de Flujo de Datos. Un Programa de Flujo de Datos Gráfico comprende una pluralidad de nodos interconectados (bloques), en donde al menos un subconjunto de las conexiones entre los nodos indica visualmente que los datos producidos por un nodo son usados por otro nodo.

Interfaz gráfica de usuario: se pretende que este término tenga toda la amplitud de su significado ordinario El término "interfaz gráfica de usuario" a menudo se abrevia como "GUI". Una GUI puede comprender solo uno o más elementos de GUI de entrada, solo uno o más elementos de GUI de salida, o tanto elementos de GUI de entrada como de salida.

Una GUI puede comprender una única ventana que tiene uno o más elementos de GUI, o puede comprender una pluralidad de elementos de GUI individuales (o ventanas individuales que tienen cada una uno o más elementos de GUI), en donde los elementos o ventanas de GUI individuales pueden agruparse a modo de mosaico opcionalmente.

Una GUI puede estar asociada con un programa gráfico. En este caso, pueden usarse diversos mecanismos para conectar elementos de GUI en la GUI con nodos en el programa gráfico. Por ejemplo, cuando se crean controles de entrada e indicadores de salida en la GUI, pueden crearse automáticamente nodos correspondientes (por ejemplo, terminales) en el programa gráfico o diagrama de bloques. Como alternativa, el usuario puede colocar nodos de terminal en el diagrama de bloques, lo que puede provocar la visualización de objetos de panel frontal de Elementos de GUI correspondientes en la GUI, o bien en tiempo de edición o bien más tarde en tiempo de ejecución. Como otro ejemplo, la GUI puede comprender elementos de GUI embebidos en la porción de diagrama de bloques del programa gráfico.

Control de Entrada - un elemento de interfaz gráfica de usuario para proporcionar entrada de usuario a un programa. Un control de entrada visualiza el valor introducido por el usuario y puede manipularse a discreción del usuario.

Indicador de salida: un elemento de interfaz gráfica de usuario para visualizar la salida de un programa. Los indicadores de salida ilustrativos incluyen tablas, grafos, medidores, cuadros de texto de salida, pantallas numéricas, etc. Un indicador de salida se denomina a veces "control de salida".

Sistema informático: cualquiera de los diversos tipos de sistemas informáticos o de procesamiento, que incluyen un sistema informático personal (PC), un sistema informático central, una estación de trabajo, un dispositivo de red, un dispositivo de Internet, un asistente digital personal (PDA), un sistema de televisión, un sistema informático de red u otro dispositivo o combinaciones de dispositivos. En general, la expresión "sistema informático" puede definirse ampliamente para abarcar cualquier dispositivo (o combinación de dispositivos) que tenga al menos un procesador que ejecuta instrucciones desde un medio de memoria.

Automáticamente: se refiere a una acción u operación realizada por un sistema informático (por ejemplo, software ejecutado por el sistema informático) o dispositivo (por ejemplo, circuitería, elementos de hardware programables, ASIC, etc.), sin entrada de usuario que especifique o realice directamente la acción u operación. Por lo tanto, el término "automáticamente" contrasta con una operación que es realizada o especificada manualmente por el usuario, en donde el usuario proporciona una entrada para realizar directamente la operación. Un procedimiento automático puede iniciarse por entrada proporcionada por el usuario, pero las acciones posteriores que se realizan "automáticamente" no son especificadas por el usuario, es decir, no se realizan "manualmente", en donde el usuario especifica cada acción que va a realizarse. Por ejemplo, un usuario que rellena un formulario electrónico seleccionando cada campo

y proporcionando información de especificación de entrada (por ejemplo, tecleando información, seleccionando casillas de comprobación, selecciones de radio, etc.) está rellenando el formulario manualmente, aunque el sistema informático debe actualizar el formulario en respuesta a las acciones de usuario. El formulario puede rellenarse automáticamente por el sistema informático en donde el sistema informático (por ejemplo, software que se ejecuta en el sistema informático) analiza los campos del formulario y rellena el formulario sin ninguna entrada de usuario que especifique las respuestas a los campos. Como se ha indicado anteriormente, el usuario puede invocar la cumplimentación automática del formulario, pero no está implicado en la cumplimentación real del formulario (por ejemplo, el usuario no está especificando manualmente respuestas a campos sino que estos se completan automáticamente). La presente memoria descriptiva proporciona diversos ejemplos de operaciones que se realizan automáticamente en respuesta a acciones que ha emprendido el usuario.

Haciendo referencia a continuación a la figura 25 ilustra un sistema informático 700 configurado para implementar diversas realizaciones de la presente invención. El sistema informático 700 puede incluir un dispositivo de visualización configurado para visualizar uno o más programas a medida que estos se crean y/o se ejecutan. El dispositivo de visualización también puede configurarse para visualizar una interfaz gráfica de usuario del/de los programa(s) durante la ejecución. El sistema informático 700 puede incluir al menos un medio de memoria en el que pueden almacenarse uno o más programas informáticos o componentes de software de acuerdo con una realización de la presente invención. Por ejemplo, el medio de memoria puede almacenar uno o más programas gráficos o herramientas de software que son ejecutables para realizar los métodos descritos en el presente documento. Adicionalmente, el medio de memoria puede almacenar una aplicación de entorno de desarrollo de programación gráfica usada para crear y/o ejecutar tales programas gráficos. El medio de memoria también puede almacenar software de sistema operativo, así como otro software para operación del sistema informático.

Haciendo referencia a continuación a la figura 26, esta ilustra un sistema que incluye un primer sistema informático 700 que se acopla a un segundo sistema informático 705. El sistema informático 700 puede acoplarse a través de una red 710 (o un bus informático) al segundo sistema informático 705. Los sistemas informáticos pueden ser, cada uno, de cualquiera de diversos tipos, según se desee. La red también puede ser de cualquiera de diversos tipos, incluyendo una LAN (red de área local), WAN (red de área extensa), Internet o una Intranet, entre otros. Los sistemas informáticos pueden ejecutar un programa de una forma distribuida. Por ejemplo, el ordenador 700 puede ejecutar una primera porción del diagrama de bloques de un programa gráfico y el sistema informático 705 puede ejecutar una segunda porción del diagrama de bloques del programa gráfico.

En una realización, la interfaz gráfica de usuario del programa gráfico puede visualizarse en un dispositivo de visualización del sistema informático 700, y el diagrama de bloques puede ejecutarse en un dispositivo acoplado al sistema informático 700. En una realización, el programa gráfico puede descargarse y ejecutarse en el dispositivo.

Se observa que las realizaciones de la presente invención pueden usarse para una gran cantidad de aplicaciones y no se limitan a nada específico. En este sentido, las aplicaciones analizadas en la presente descripción son solo ilustrativas, y las realizaciones de la presente invención pueden usarse en cualquiera de diversos tipos de sistemas.

La figura 27 es un diagrama de bloques de alto nivel de un sistema ilustrativo que puede ejecutar o utilizar programas gráficos. La figura 3A ilustra un diagrama de bloques de alto nivel general de un sistema de control y/o simulación genérico que comprende un controlador 720 y una planta 725. El controlador 720 representa un sistema/algoritmo de control que el usuario puede estar intentando desarrollar. La planta 725 representa el sistema que el usuario puede estar intentando controlar. Por ejemplo, si el usuario está diseñando un programa para un robot, el controlador 720 es el programa y la planta 725 es el robot. Como se muestra, un usuario puede crear un programa, tal como un programa gráfico, que especifica o implementa la funcionalidad de uno o ambos del controlador 720 y la planta 725.

La figura 28 ilustra un sistema ilustrativo que puede realizar funciones de control y/o simulación. Como se muestra, el controlador 720 puede implementarse mediante un sistema informático 700 u otro dispositivo (por ejemplo, que incluye un procesador y un medio de memoria y/o que incluye un elemento de hardware programable) que ejecuta o implementa un programa gráfico, o un programa generado basándose en un programa gráfico. De forma similar, la planta 725 puede implementarse mediante un sistema informático u otro dispositivo 730 (por ejemplo, que incluye un procesador y un medio de memoria y/o que incluye un elemento de hardware programable) que ejecuta o implementa un programa gráfico, o puede implementarse en o como un sistema físico real, por ejemplo, un robot.

La figura 29 es un diagrama de bloques que representa una realización del sistema informático 700 y/o 705 ilustrado en las figuras 25 y 26, o el sistema informático 700 mostrado en las figuras 28. Se observa que puede usarse cualquier tipo de configuración o arquitectura de sistema informático según se desee, y la figura 29 ilustra una realización de PC representativa. También se observa que el sistema informático puede ser un sistema informático de propósito general, un ordenador implementado en una tarjeta u otros tipos de realizaciones. Los elementos de un ordenador que no son necesarios para entender la presente descripción se han omitido por simplicidad.

El ordenador puede incluir al menos una unidad de procesamiento central o CPU (procesador) 760 que se acopla a un procesador o bus de anfitrión 762. La CPU 760 puede ser de cualquiera de varios tipos. Un medio de memoria, que habitualmente comprende RAM y se denomina memoria principal 766, se acopla al bus de anfitrión 762 por medio del

controlador de memoria 764. La memoria principal 766 puede almacenar instrucciones de programa que implementan realizaciones de la presente invención. La memoria principal también puede almacenar software de sistema operativo, así como otro software para operación del sistema informático.

El bus de anfitrión 762 puede acoplarse a un bus de expansión o de entrada/salida 770 por medio de un controlador de bus 768 o lógica de puente de bus. El bus de expansión 770 puede ser el bus de expansión de PCI (Interconexión de Componentes Periféricos), aunque pueden usarse otros tipos de bus. El bus de expansión 770 incluye ranuras para diversos dispositivos tales como los descritos anteriormente. El ordenador 700 comprende además un subsistema de visualización de vídeo 780 y un disco duro 782 acoplado al bus de expansión 770. El ordenador 700 también puede comprender una tarjeta de GPIB 722 acoplada a un bus de GPIB 712.

Como se muestra, un dispositivo 790 también puede conectarse al ordenador. El dispositivo 790 puede incluir un procesador y memoria que puede ejecutar un RTOS. El dispositivo 790 también puede comprender, o en su lugar, un elemento de hardware programable. El sistema informático puede configurarse para desplegar un programa gráfico o un programa generado basándose en un programa gráfico en el dispositivo 790 para su ejecución en el dispositivo 790. El programa desplegado puede adoptar la forma de instrucciones de programa gráfico o estructuras de datos que representan directamente el programa gráfico, o que se generaron basándose en el programa gráfico. Como alternativa, el programa gráfico desplegado puede adoptar la forma de código de texto (por ejemplo, código C) generado a partir del programa gráfico. Como otro ejemplo, el programa gráfico desplegado puede adoptar la forma de código compilado generado a partir del programa gráfico o a partir de código de texto que a su vez se generó a partir del programa gráfico. En algunas realizaciones, el programa gráfico y/o el programa generado a partir del programa gráfico son programas de flujo de datos. En una realización adicional, el programa generado puede ser un programa de configuración de hardware, y puede desplegarse en un elemento de hardware programable. Además, en algunas realizaciones, el programa generado puede ser adecuado para el despliegue de una forma distribuida, por ejemplo, a través de múltiples objetivos, posiblemente heterogéneos. Por lo tanto, por ejemplo, una primera porción del programa puede dirigirse a una plataforma basada en CPU, mientras que otra porción puede dirigirse a un elemento de hardware programable.

Los ejemplos usados para describir esta invención usan el lenguaje de programación C textual, sin embargo, esta invención podría usar igualmente cualquier lenguaje tal como, pero sin limitación, C++, JavaScript, Python, Basic, ensamblador, etc. Los ejemplos usados para describir esta invención usan el lenguaje de programación gráfico de la aplicación Modkit que se basa en el lenguaje de programación gráfico Scratch, sin embargo, esta invención podría usar igualmente cualquier lenguaje de bloque gráfico tal como Blockly, Alice, etc.

El nuevo método permite un método lento en una única etapa de hacer una transición de bloques a texto, al tiempo que mantiene los beneficios de la modalidad de bloques, tales como el ciclo de edición-ejecución ajustado y disminuye la carga cognitiva para los estudiantes al reducir la cantidad de memorización de comandos y sintaxis que necesitan los estudiantes.

La primera etapa es permanecer dentro del entorno de programación gráfica mientras se aprende codificación textual. Esto retarda la transición de las aplicaciones a una fecha posterior, después de que el estudiante haya dominado la codificación de texto.

La segunda etapa es permitir que el estudiante vea el texto detrás de un bloque de codificación gráfica en incrementos pequeños. Un estudiante puede ver un único bloque convertido a texto. Mediante una acción simple, tal como un clic derecho o una pulsación larga, un bloque puede convertirse en un bloque de codificación textual. Esto permite el aprendizaje del formateo de texto y el lenguaje en pequeños fragmentos digeribles. Los estudiantes pueden exponerse lentamente a la sintaxis adicional incluida en la modalidad de texto, que se abstrae cuando se usa la modalidad de bloques. En las figuras 5A y 5B, los estudiantes verían cómo sería necesario añadir los paréntesis y las comillas emparejadas cuando se usa la modalidad de texto. Los estudiantes también pueden ver que el nombre del bloque Imprimir cambia ligeramente en la modalidad de texto a "printf".

La presente invención se proporciona, en una o más realizaciones, una interfaz de usuario configurada para ejecutarse en un ordenador, procesador de aplicación móvil o navegador web. La interfaz de usuario se separará en diversas áreas (descritas adicionalmente a continuación), tales como, pero sin limitación, un Área de Barra de Herramientas y un Área de Trabajo similar a otros entornos de programación de bloques. Sin embargo, como se proporciona adicionalmente, la presente invención se refiere a un entorno de programación de bloques que proporciona programación de modo mixto. El entorno de acuerdo con una o más realizaciones de la invención permite que el usuario cree programas usando una mezcla de Bloques Gráficos y código textual dentro de Bloques de Código. Adicionalmente, el usuario puede ver y/o convertir cualquier Bloque Gráfico como código textual. Aun así, el usuario puede crear y editar código textual dentro del programa de Bloques Gráficos.

La figura 5A muestra un programa 100 hecho de Bloques Gráficos. El primer bloque 102 es un bloque Cuando que se establece para activarse cuando se inicia el programa. El segundo bloque 104 es un bloque Imprimir que imprime el contenido textual 106 introducido por el usuario. En este caso, el usuario introdujo el texto:

¡Hola, mundo!

La figura 5B muestra un programa 110 que es equivalente al de la figura 5A. El programa 110 contiene un Bloque Gráfico 102 y un Bloque de Código 112. El Bloque Gráfico 102 es un bloque Cuando que se establece en Inicio. El Bloque de Código 112 usa la modalidad de texto y en este caso contiene el código C convencional 114. La única línea de código 114 es una sentencia printf con un argumento. La sentencia contiene formateo C convencional con paréntesis 116, comillas 118 y un punto y coma 120. La codificación con colores de las porciones del texto para hacer coincidir los bloques y argumentos también puede implementarse para ayudar al estudiante a entender qué partes del bloque se volvieron qué partes del texto. En este ejemplo, la palabra impresa de un bloque de color azul 116 se volvió

```
printf(              );
```

con el texto y los símbolos de color azul. ¡Hola, mundo!, como un ejemplo de color púrpura, se volvió

```
"¡Hola, mundo!"
```

con el texto de color púrpura.

Una vez que un Bloque Gráfico se convierte en un bloque de texto, el nuevo bloque textual puede editarse de una forma idéntica a la de una aplicación de edición de texto real. Esto introduce al estudiante a la edición textual de una forma línea a línea, antes de ver y navegar por un programa completo en un nuevo entorno. Adicionalmente, el texto en el bloque de codificación textual puede usar soportes de formateo de texto de editor moderno tales como codificación de color de términos e identificación de errores de formateo antes del tiempo de compilación.

Una vez que se entienden comandos individuales simples, pueden convertirse bloques de código más grandes con un comando de usuario simple. Como se ilustra en las figuras 6A y 6B, la sentencia Si se convierte con una acción por el usuario.

La investigación indica que un problema que tienen los estudiantes cuando hacen una transición de bloque a texto es que los estudiantes tienen dificultades para definir el ámbito (por ejemplo, usando corchetes) de su tarea. Este nuevo método permite a los estudiantes usar el bloque para definir el ámbito. Cuando los estudiantes convierten la sentencia SI a texto, se les proporcionan las llaves coincidentes. Los estudiantes pueden centrarse en la lógica necesaria dentro del ámbito.

La figura 6A muestra un programa 130 compuesto por todos los Bloques Gráficos. Los primeros cuatro bloques son un bloque Cuando 102, un bloque Imprimir 104, un bloque X es Igual a 132 y un bloque Para siempre 134. El bloque Imprimir 104 contiene un parámetro 105 de ¡Hola, mundo!. El bloque X es Igual a contiene un valor 136 de 0. Además, se proporciona un bloque Si 138 como un subbloque al bloque Para siempre 134. El bloque Si 138 contiene un bloque matemático de menor que 140, un bloque de sensor de distancia 142 y un valor 144 de 50. El bloque Si 138 contiene adicionalmente tres subbloques: un bloque Reproducir Sonido 150, un bloque X es Igual a 152 y un bloque Imprimir 154. El bloque Reproducir Sonido 150 contiene un Parámetro 156 ajustado a la Alarma 158. El bloque de X es Igual a 152 incluye un bloque matemático de más 160, un bloque de variable X 162 y un valor 164 de 1. El bloque Imprimir 154 contiene un bloque de variable X 166. El símbolo 168 al final del bloque Imprimir 154 es un botón de establecimiento de formato que permite que el usuario establezca el formateo de impresión.

La figura 6B muestra un programa de acuerdo con una realización de la invención 170 que es equivalente al programa de Bloques Gráficos 130 de la figura 6A. Los primeros cuatro Bloques Gráficos son el bloque Cuando 102, el bloque Imprimir 104, el bloque X es Igual a 132, y el bloque Para siempre 138 que son los mismos que en la figura 6A. El quinto bloque es un Bloque de Código 172. El contenido del Bloque de Código 172 contiene una sentencia "si" que usa el formateo en C convencional 174, que representa el bloque Si 138 de tipo Bloque Gráfico de la figura 6A, que incluye sangrías, llaves, paréntesis, comillas y puntos y coma. La sentencia printf 174 contiene formateo C convencional que incluye los argumentos ("%d", x) 176.

Como se ilustra, al usuario se le presentará el bloque Si 138 con los parámetros 140, 142 y 144 de la figura 6A que están todos representados en el Bloque de Código 172 de la figura 6B con la sentencia en formato C, 176,

```
if (rangefinder (distance) < 50)
{

}
```

El bloque Reproducir Sonido 150 de la figura 6A se representa en la figura 6B con la sentencia en formato C, 178,

```
playsound(alarm);
```

El bloque X = 132 de la figura 6A se representa en la figura 6B con la sentencia en formato C, 180,

```
x = x + 1;
```

El bloque Imprimir 154 de la figura 6A se representa en la figura 6B con la sentencia en formato C, 182,

```
printf("%d", x);
```

Las sentencias en C en el Bloque de Código 172 están en el orden apropiado comenzando con la sentencia Si 176,

seguida por una llave de apertura 183, seguida entonces por las tres líneas de código contenidas 178, 180 y 182, y completando con la llave de cierre 185 al final.

Una característica importante de esta programación de modo mixto es que los bloques y el código están en línea entre sí y en el orden secuencial correcto.

El entorno gráfico está diseñado ahora para manejar modos mixtos de entrada de programa, con bloques gráficos en el sentido tradicional, y nuevos bloques textuales que permiten teclear código dentro. La aplicación puede ahora analizar bloque a código y mezclar en el código textual en una única salida de programa para el objetivo previsto que ejecuta el programa.

El usuario puede continuar escribiendo programas usando una combinación de bloques gráficos y código. Como se ilustra en la figura 7, el estudiante ha añadido un bloque gráfico y un nuevo bloque de código textual a un único programa. Las lecciones y el aprendizaje pueden continuar al tiempo que se facilita gradualmente al estudiante la introducción a la programación de texto en pequeñas cantidades. Adicionalmente, a los estudiantes se les puede dar la libertad de estructurar su propio aprendizaje. Los estudiantes pueden introducir algunos comandos de texto en su código gráfico a su propio ritmo de instrucción. La investigación indica que los estudiantes fluctúan entre la modalidad de bloques y la modalidad de texto a medida que aprenden nuevos comandos y encuentran nuevas estructuras de programación. Esta funcionalidad permite que el usuario realice esta estructuración a un ritmo individualizado.

La figura 7 muestra un programa 190 compuesto por los Bloques Gráficos 192, 194, 196 y 198. En el interior del Bloque de Código 200 no se ha introducido ningún código C. El texto 202 "empezar a teclear código..." es ilustrativo y está diseñado para dar instrucciones a un nuevo usuario para comenzar a teclear aquí. El texto 202 desaparece habitualmente una vez que comienza el tecleo.

A medida que el estudiante progresa, podrá escribir lentamente más y más código textual, al tiempo que necesitará recurrir a cada vez menos bloques gráficos. En las figuras 8A a 8D, hay un ejemplo de las etapas finales de aprendizaje a medida que el estudiante domina las capas externas más complejas del programa hasta que todo lo que hay es código de texto y no hay ningún bloque gráfico.

La figura 8A representa un ejemplo de programación de bloques por un estudiante que ha dominado la modalidad de bloques y está listo para comenzar a realizar una transición a la modalidad de texto. La figura 8A muestra un programa 210 compuesto por todos los Bloques Gráficos. Este programa está compuesto por un bloque Cuando 212, un bloque Imprimir 214, un bloque X = 216, un bloque Para siempre 218 y un bloque Si 220 junto con sus parámetros y contenidos.

La figura 8B representa un ejemplo tanto de programación de bloques como de programación de texto por un estudiante que acaba de comenzar a realizar una transición a la modalidad de texto. La figura 8B muestra un programa 222 que es equivalente al programa 210 de la figura 8A. El programa 222 tiene los mismos Bloques Gráficos que incluyen el bloque Cuando 212, el bloque Imprimir 214, el bloque X = 216 y el bloque Para siempre 218. El programa 222 también contiene un Bloque de Código 224 que contiene sentencias en C 225 y formateo que es funcionalmente equivalente al bloque Si 220 junto con sus parámetros y contenidos en la figura 8A.

La figura 8C representa un ejemplo tanto de programación de bloques como de programación de texto por un estudiante que mayoritariamente ha hecho una transición a la modalidad de texto. La figura 8C muestra un programa 226 que es equivalente tanto al programa 210 de la figura 8A como al programa 222 de la figura 8B. El bloque Cuando 212 sigue siendo un Bloque Gráfico, sin embargo, el Bloque de Código 227 tiene las sentencias en C 225 del Bloque de Código 224 de la figura 8B con las sentencias en C 228 añadidas que son equivalentes al bloque Imprimir 214, el bloque X = 216 y el bloque Para siempre 218 de la figura 8B.

La figura 8D representa un ejemplo de toda la programación de texto por un estudiante que ha hecho una transición completa a la modalidad de texto. La figura 8D muestra un programa 230 que está hecho completamente de un Bloque de Código 232. El programa 230 es equivalente al programa 210 de la figura 8A, el programa 222 de la figura 8B y el programa 226 de la figura 8C. El Bloque de Código 232 tiene las sentencias en C 228 y 225 del Bloque de Código 227 de la figura 8C con las sentencias en C 234 añadidas que son equivalentes al bloque Cuando 212 de la figura 8C. Se prevén varias opciones para la implementación de la nueva Programación de Modo Mixto. En una realización existe la siguiente funcionalidad.

- Una característica de vista de código que permite que el usuario eche un vistazo al código que compone un bloque sin convertir realmente el bloque.
- Una característica de Conversión de Bloque que permite que el bloque se vuelva un Bloque de Código. El Bloque de Código puede ahora editarse y modificarse.
- Un nuevo Bloque de Código en blanco puede arrastrarse desde la caja de herramientas a la izquierda y usarse con otros bloques normales o como un código autónomo.
- Los Bloques de Código pueden tener características similares a los editores de texto, tales como texto codificado por colores, sangría automática, autocompletar, soporte de corchetes, etc.

- Cuando se edita un Bloque de Código, la barra de herramientas izquierda puede cambiar de bloques a fragmentos de texto, permitiendo que los mismos se arrastren al texto del Bloque de Código.
- Los Bloques Normales pueden arrastrarse a un Bloque de Código existente y, cuando se sueltan, se convierten en texto y se insertan.
- Se puede empezar a introducir tipos de datos dentro de los vistazos y conversiones de Bloque de Código.
- Cuando existe un Bloque de Código dentro de un Bloque normal contenedor tal como un SI o REPETIR, y cuando el SI o REPETIR se convierte en un Bloque de Código, el resultado es un único Bloque de Código con tanto el nuevo SI o REPETIR como el código preexistente.
- El Bloque de Código adyacente puede fusionarse.
- Colocar el cursor sobre código textual puede proporcionar descripciones o ayuda adicionales.
- El código puede asimismo convertirse de vuelta a bloques. Esto puede ser difícil debido a que hay cosas en el texto que pueden no ser fácilmente convertibles de nuevo a bloque, pero ese caso podría detectarse o evitarse.
- ¿Coloración de bloques para "errores" para identificar problemas? Es decir, todo el bloque se vuelve rojo para identificar problemas antes de la descarga. Además de la coloración de bloques, el texto puede resaltarse para señalar el problema de codificación específico. Colocar el cursor sobre el resaltado dará a los usuarios pistas acerca de lo que es incorrecto.
- Siguen existiendo las características habituales de compilación y descarga.

Vistazo al código: Las figuras 9A - 9E ilustran una Función de Vistazo que permite que el usuario eche un vistazo a un Bloque de Código con sentencias en C que constituye un bloque gráfico sin convertir realmente el bloque gráfico. Esta característica puede permitir que el usuario vea tanto el código como los bloques normales que crearon ese código al mismo tiempo. Una implementación posible de una Función de Vistazo es hacer clic con el botón derecho sobre un bloque gráfico. En un entorno táctil, esto podría ser una pulsación larga.

La figura 9A muestra un programa 250 que contiene dos Bloques Gráficos: el bloque Cuando 252 y el bloque Imprimir 254. En la figura 9B, se ilustra un método de la Función de Vistazo y que permite que el usuario echar un vistazo al Bloque de Código 256 que es equivalente a uno de los Bloques Gráficos. El programa en la figura 9B es el mismo programa 250 de la figura 9A que contiene dos Bloques Gráficos, el bloque Cuando 252 y el bloque Imprimir 254. Un cursor 258 se muestra haciendo clic en el bloque Imprimir 254, que como resultado de la acción de clic, el Bloque de Código 256 equivalente se muestra a la derecha del bloque Imprimir 254. O bien liberar el botón de clic ocultaría el Bloque de Código 256 equivalente o bien, como alternativa, hacer clic de nuevo podría ocultar el Bloque de Código 256 equivalente.

La figura 9C muestra un ejemplo de una Función de Vistazo para echar un vistazo al Bloque de Código que es equivalente a múltiples Bloques Gráficos, algunos de los cuales están incompletos. El programa 260 contiene diversos Bloques Gráficos. Se muestra un cursor 258 haciendo clic en el bloque Si 260. Como resultado de la acción de clic, el Bloque de Código 262 equivalente se muestra a la derecha del bloque Si 260. En este ejemplo, al bloque Si 260 le falta un argumento 264 y le faltan el/los bloque(s) en la ranura Si No 266 A las sentencias de código C equivalentes en el Bloque de Código 262 les faltan apropiadamente las sentencias de argumento Si 268 y las sentencias Si No 270.

La figura 9D muestra un ejemplo de la Función de Vistazo para echar un vistazo al Bloque de Código que es equivalente a múltiples Bloques Gráficos en el programa 272. El cursor 258 hace clic en el bloque Si 274. Como resultado de la acción de clic, el Bloque de Código 276 equivalente se muestra a la derecha del bloque Si 274.

La función de vistazo puede tener dos opciones cuando se usa sobre el primer bloque de un programa; 1) ver solo el código equivalente del primer bloque, o 2) ver el programa completo. La figura 9E muestra un ejemplo de la Función de Vistazo usada para echar un vistazo al Bloque de Código que es equivalente a un programa completo 278 de Bloques Gráficos.

Convertir Bloque Gráfico en Bloque de Código: Las figuras 10A - 10D muestran una implementación de la función de Conversión de Bloque que convierte un Bloque Gráfico en un Bloque de Código. Una vez convertido, el Bloque de Código puede editarse usando sentencias en C convencionales. Pueden idearse muchas acciones de usuario alternativas para lograr los mismos resultados. En la figura 10A se ilustra un programa 300 compuesto por diversos Bloques Gráficos, incluyendo un bloque Cuando 302, un bloque Imprimir 304 y un bloque Repetir 306 con un subbloque Reproducir Sonido 308. Como se muestra en la figura 10B como una primera etapa, el cursor 258 se usa para hacer clic en uno de los Bloques Gráficos como una Función de Vistazo inicial para invocar el Bloque de Código 310 equivalente. El programa 300 es el mismo programa de la figura 10A. Como se ilustra, el cursor 258 se muestra haciendo clic en el Bloque Gráfico, bloque Imprimir 304. Como resultado de la acción de clic, el Bloque de Código 310 equivalente se muestra a la derecha del bloque Imprimir 304. Además y de acuerdo con la presente invención, el Bloque de Código 310 tiene un botón Editar 312.

La figura 10C es una segunda etapa de la función de Conversión de Bloque. La figura 10C muestra al usuario que inicia una conversión de un Bloque Gráfico a un Bloque de Código equivalente. El programa 300 es el mismo programa de las figuras 10A - 10B. El cursor 258 se muestra haciendo clic en el botón Editar 312. La figura 10D muestra el resultado de que el usuario haga clic en el botón Editar 312 de la figura 10C. El programa mixto 300' incluye una mezcla de Bloques Gráficos 302 y 306 y ahora un Bloque de Código 310 que ha sustituido al bloque Imprimir 304 de

las figuras previas. El botón Editar 312 es parte del Bloque de Código 310 y puede usarse como un medio para permitir que el usuario edite el Bloque de Código 310 usando sentencias de programación C 314. Como alternativa, el usuario puede hacer clic dentro del Bloque de Código 310 para comenzar el proceso de edición.

Nuevo Bloque de Código: Las figuras 11A - 11C muestran, de acuerdo con una realización de la invención, una plantilla de pantalla 320 que permite a un usuario añadir, desde un Área de Barra de Herramientas 322, un nuevo Bloque de Código en blanco 324 que puede arrastrarse desde el área de caja de herramientas 322 a un Área de Trabajo 324. El Bloque de Código en blanco 324 puede usarse con otros Bloques Gráficos o como un Bloque de Código autónomo. Una vez insertado, el usuario puede usar entonces el botón Editar 312 para insertar código, es decir, se elimina el texto "insertar código" y el cursor se muestra parpadeando. El usuario puede escribir ahora cualquier código nuevo desde cero.

Con detalle más específico, la figura 11A muestra una plantilla de pantalla 320 que incluye el Área de Barra de Herramientas 322 en el lado izquierdo y un Área de Trabajo 324 en el lado derecho. Se muestran múltiples Bloques Gráficos en el Área de Barra de Herramientas 322 que incluyen un bloque Cuando 326, un bloque Radiodifundir 328, un bloque Para siempre 330 y un bloque Repetir 332. También se muestra un Bloque de Código en blanco 334 en el Área de Barra de Herramientas 322. El Bloque de Código en blanco 334 contiene el texto "insertar código" para que el usuario sepa que este bloque permite teclear dentro. Los bloques en el Área de Barra de Herramientas 322 pueden arrastrarse al Área de Trabajo 324 a la derecha. El Área de Trabajo 324 ya contiene múltiples Bloques Gráficos que forman el inicio de un programa 340.

La figura 11B muestra la misma escena que en la figura 11B. El usuario ha arrastrado el Bloque de Código en blanco 334 desde el Área de Barra de Herramientas 322 a la izquierda al Área de Trabajo 324 a la derecha, conectando el nuevo Bloque de Código en blanco 334' a la parte inferior del bloque Repetir 332. El nuevo Bloque de Código en blanco 334 tendrá, cuando se inserte en el Área de Trabajo 324, un botón Editar 312. La figura 11C muestra la misma escena que la figura 11B excepto que el usuario ha hecho clic en el botón Editar 312 y el usuario ha comenzado a teclear una sentencia en C parcial 342. Aunque se ilustra como una sentencia de impresión de programación, el usuario puede programar cualquier sentencia en el Bloque de Código en blanco 334'.

Características de editor en un Bloque de Código: Las figuras 12A y 12B ilustran Bloques de Código que tienen características similares a los editores de texto, tales como texto codificado por colores, sangría automática, autocompletar, soporte de corchetes, etc. Una opción es imitar la coloración de los bloques con el texto nuevo. En este ejemplo, los corchetes { y } también están codificados por colores con la línea que está asociada con los mismos. Una función de "Colocar Cursor Sobre" permite que los estudiantes coloquen el cursor sobre el código de texto %d para ver que este es un especificador de formato. De forma similar con los tipos de datos, debido a que el bloque gráfico abstrae esa información, presentar el "pasar el ratón por encima" como una técnica de estructuración ayudaría a los estudiantes.

La figura 12A muestra un programa 350 hecho a partir de Bloques Gráficos. Colocando un cursor (no mostrado) sobre el bloque Repetir 352, el usuario ha echado un vistazo al bloque Repetir 352 revelando el Bloque de Código 354 equivalente. Cada uno de los Bloques Gráficos puede tener un color que ayuda al usuario a inferir su función. El color del bloque Cuando 356 y el bloque Repetir 352 es naranja. El color de los bloques Imprimir 358 y 360 es azul. El color del bloque X 362 y el bloque X = 364 es rojo. El color del bloque + 366 es verde. Las sentencias en C 368 en el Bloque de Código 354 se derivan de los Bloques Gráficos 352 a 366. La porción de las sentencias en C 370, 372 y 374 (también mostradas a continuación) están codificadas en color naranja debido a que son el equivalente del bloque Repetir 352 que también es naranja.

```
for(int 1 = 0; i < 10; i++)
{

}
```

Las porciones de las sentencias en C 376 y 380 (también mostradas a continuación) están codificadas en color rojo debido a que son el equivalente del bloque x = 364 y los bloques x 362 que también es rojo.

```
x = x;
    x
```

Las porciones de sentencias en C 382 (también mostradas a continuación) están codificadas por color verde debido a que son el equivalente del bloque + 366 que también es verde.

```
+
```

Las porciones de sentencias en C 384 (también mostradas a continuación) están codificadas por color azul debido a que son el equivalente del bloque Imprimir 360 que también es azul.

```
print("%d", );
```

La figura 12B es igual que la figura 12A. El usuario, sin embargo, ha pasado el cursor 258 sobre una porción de texto "%d" 390 con el que el usuario no está familiarizado, debido a que no aparece en el bloque Imprimir 360. Aparece una

ventana emergente 392 con una descripción de ayuda 394 para ayudar al usuario a entender la porción de texto 390 de la sentencia en C 368.

Código dentro de un Bloque: la figura 13 muestra que un nuevo Bloque de Código puede soltarse en la ventana de argumento de un bloque. Esto permite que las fórmulas se expresen en formato de código. El siguiente ejemplo usa este método en el bloque Imprimir y el bloque X =. Haciendo referencia a continuación a la figura 13, se muestra de acuerdo con una realización de la invención, un programa 400 que contiene Bloques Gráficos: el bloque Cuando 402 y el bloque Repetir 404. También se proporcionan bloques Imprimir 406 y 408 y son una nueva forma de bloques Imprimir que permite tanto bloques como argumentos C dentro de la ventana de argumento. El usuario ha introducido el argumento

"Hola"

en la ventana de argumento 410 del bloque Imprimir 406, mientras que el usuario ha introducido el argumento

"%d", x

en la ventana de argumento 412 del bloque Imprimir 408.

El bloque x = 414 es una nueva forma de bloque x = que permite tanto bloques como argumentos en C dentro de la ventana de argumentos. En este ejemplo, el usuario ha introducido el argumento

2*x^2 + 0,5

en la ventana de argumento 416. Otras opciones para el argumento ## son cualquier argumento que sea válido para un argumento de ecuación C.

Cambio de barra de herramientas: Las figuras 14A y 14B ilustran en una realización de la invención la capacidad de cambiar el Área de Barra de Herramientas izquierda 420 de bloques a fragmentos de texto dependiendo de lo que el usuario está editando en el Área de Trabajo derecha 422. La figura 14A muestra al usuario arrastrando un Bloque Gráfico 424 desde el Área de Barra de Herramientas 420 al Área de Trabajo 422. La figura 14B muestra la barra de herramientas izquierda transformada en fragmentos de código. El usuario puede ahora arrastrar fragmentos de código desde el Área de Barra de Herramientas izquierda 420 al Bloque de Código 426. Pueden usarse varios métodos para cambiar la barra de herramientas izquierda de bloques a texto. Una opción es un conmutador controlado por usuario. Otra opción es cambiar automáticamente a fragmentos de texto en la barra de herramientas cuando el usuario hace clic dentro de un Bloque de Código, y cambiar automáticamente a bloques en la barra de herramientas cuando el usuario hace clic en cualquier lugar fuera de un Bloque de Código.

Soltar bloques en un Bloque de Código: En las figuras 15A a 15D, se ilustra una función proporcionada por una realización de la invención que permite que el usuario arrastre Bloques Gráficos a un Bloque de Código existente y cuando se libera/suelta/añade en la invención convierte automáticamente el Bloque Gráfico en texto e inserta el texto en la posición que el usuario desea / posición del cursor. Esto ayudaría a los usuarios con dificultades de transición. Los estudiantes podrían soltar un comando y ver la sintaxis que se añadió. A medida que el bloque se está soltando en su lugar, el texto se actualizará dinámicamente. Pero el sistema también puede asegurarse de que el bloque solo puede soltarse en una ubicación que tenga sentido. Por lo tanto, el usuario no puede soltar este en una ubicación que provocaría un error de codificación.

La figura 15A muestra un grupo de Bloques Gráficos y un Bloque de Código 430. El Bloque de Código 430 contiene sentencias en C 432. La figura 15B muestra la misma escena que la figura 15A con la adición de que el usuario está arrastrando un bloque de Iterador (Iterator) de variable 434 al Bloque de Código 430. La aplicación o invención permitirá que el bloque de Iterador (Iterator) de variable 434 se suelte solo en ubicaciones válidas. Las ubicaciones válidas son como parte de la sentencia Si 436 antes o después de == 438 o como parte de la sentencia Imprimir 440 antes o después del texto "¡Sabemos contar!" 442. La figura 15C muestra la misma escena que la figura 15B después de que el usuario haya arrastrado y soltado el bloque de Iterador (Iterator) de Variable 434 en la sentencia Si 436 y a la izquierda del == 438. La sentencia en C válida resultante if (Iterator == "") 444 es el resultado de la acción de arrastrar y soltar. La figura 15D muestra la misma escena que la figura 15C con el usuario interaccionando adicionalmente con el Bloque de Código 430 borrando el de la sentencia Si 436 y tecleando el nuevo texto "8" 448.

Tipos de datos: La figura 16 ilustra que al usuario se le pueden presentar tipos de datos. Los bloques abstraen los tipos, pero en algún momento el estudiante tendrá que aprender cuáles son. La introducción de tipos de datos ayudaría a la transición al código textual. Los tipos de datos son uno de los puntos débiles que identifica la investigación cuando se realiza una transición de los bloques a texto. Cuando los estudiantes están en "vistazo de código" y ven el texto, el tipo de datos podría identificarse por peso, color, formato, etc., para resaltar al estudiante que esto no estaba en el bloque gráfico. En la figura 16 el usuario ha echado un vistazo al Iterador (Iterator) de Bloque Gráfico = 0 450 que revela el Bloque de Código 452 que contiene la sentencia en C:

Int Iterator = 0;

introduciendo el concepto de un tipo de datos. Adicionalmente, en la figura 16 el usuario ha echado un vistazo al grupo de Bloques Gráficos contenidos dentro de la repetición 10 454 que revela el Bloque de Código 456 que contiene las sentencias en C:

```
for(int i=0; i<10; I++)
{
```

```
        Iterator = Iterator + 1;
        if (Iterator == 8)
        {
        printf("¡Sabemos contar!);
        }
        else
        {
        }
    }
```

introduciendo adicionalmente el concepto de tipos de datos en la "sentencia para".

Conversión de bloque mejorada - Pueden implementarse varios métodos para convertir un Bloque Gráfico cuando el Bloque Gráfico contiene un Bloque de Código. La figura 17 muestra un Bloque de Código 462 contenido dentro de un bloque Repetir 460 de tipo Bloque Gráfico. Cuando el usuario solicita convertir el bloque Repetir 462 en un Bloque de Código, el programa de lado derecho 470 es el resultado. El nuevo Bloque de Código fusiona el antiguo Bloque de Código 462 y envuelve la funcionalidad de repetición alrededor del mismo. El Bloque de Código 472 resultante contiene tanto las sentencias en C para el bloque Repetir 460 como el Bloque de Código 462 antiguo.

Como se ilustra, la figura 17 muestra un programa 464 a la izquierda que contiene una mezcla de Bloques Gráficos y Bloques de Código. Como se muestra a la izquierda, el Bloque de Código 462 está contenido dentro del bloque Repetir de Bloque Gráfico 460. A través de interacción de usuario, el bloque Repetir se selecciona para su conversión a un Bloque de Código. El programa resultante 470 a la derecha muestra que la aplicación ha convertido el bloque Repetir 460 a las sentencias en C equivalentes y las ha fusionado en el Bloque de Código 472. Las sentencias en C equivalentes del bloque Repetir 460 474 también se muestran a continuación:

```
for (int i = 0, i<10, i++)
{
}
La sentencia en C
Counter = Counter*10;
```

476 del Bloque de Código 462 del lado izquierdo se insertó en la ubicación apropiada dentro de las sentencias en C equivalentes 474 del bloque Repetir 460. El Bloque de Código 472 resultante a la derecha es equivalente al bloque Repetir 460 a la izquierda más el Bloque de Código 462 a la izquierda.

Fusión de Bloques de Código: Pueden implementarse varios métodos para fusionar Bloques de Código. Un método podría ser arrastrar un Bloque de Código a otro, indicando la intención de fusionarlos. La posición exacta del bloque arrastrado podría indicar si se desea añadir el bloque arrastrado antes o después del código de bloques de destino. En la figura 18, el Bloque de Código inferior 480 se desacopló y se soltó sobre la mitad inferior del Bloque de Código superior 482. Un nuevo Bloque de Código fusionado 484 contiene las sentencias en C de ambos bloques 480 y 482. La aplicación simplemente fusionaría automáticamente las dos cuando se soltara.

Indicación de error: También es un aspecto de la invención proporcionar coloración de Bloque de Código para "errores" para permitir que el usuario identifique problemas fácilmente. También podrían implementarse otros métodos para identificar errores. La figura 19A muestra un Bloque de Código 500 que usa números de línea para ayudar a los usuarios a identificar una línea específica. En este caso, la línea 10 tiene un error. El especificador incorrecto "%s" para un número entero en la instrucción printf. La aplicación puede indicar errores haciendo que el fondo del bloque sea rojo. Estos problemas pueden identificarse inmediatamente después de teclear en lugar de esperar hasta que el usuario intente compilar y descargar. La figura 19B muestra la línea 502 que contiene el error resaltado. También podría resaltarse o indicarse la parte específica de la línea. La figura 19C debería ayudar adicionalmente al usuario en donde el usuario ha hecho clic en la línea resaltada y una ventana emergente 504 proporciona sugerencias acerca de cómo arreglar el problema. La figura 19D muestra el programa con el error corregido. La figura 19E muestra que la aplicación puede indicar opcionalmente que existen errores haciendo que el fondo del error sea rojo 506. La figura 19F muestra que la aplicación puede indicar opcionalmente que no existen errores haciendo verde el fondo 508 de todo el Bloque de Código.

Funciones avanzadas: La figura 20A divulga la capacidad de crear bloques con las funciones 520 y 522 *(estos son los dos bloques autónomos a la derecha que empieza con int y también empieza con bool)* para ayudar a los usuarios a aprender cómo descomponer el código en unidades reutilizables. Los bloques de función no tienen "conectores" debido a que los mismos son unidades de código autónomas (524). La forma del bloque de función puede corresponder con el tipo de retorno de la función adjunta. Las llamadas a los bloques de función pueden embeberse en bloques normales, y solo encajarán con lo que permite su tipo de retorno. Esto también tiene la ventaja singular de poder disponer su código en vertical o en horizontal. La figura 20A muestra dos bloques de función colocados junto al código de bloque que llama a las funciones.

La figura 20B muestra características adicionales de un editor de texto que pueden añadirse a los Bloques de Código,

tales como, pero sin limitación, números de línea, puede añadirse una barra de estado 530 en la parte inferior para indicar el número de línea, tipo de código, etc., y puede añadirse una barra superior 532 con menús desplegables tales como archivo, editar, etc. Todo esto puede estar contenido dentro del Bloque de Código. Como alternativa, esto puede hacerse con botones e iconos. Como alternativa, esto puede hacerse dentro de la estructura de menú de programas principales.

Diferentes tipos de Código: El Bloque de Código analizado previamente podría tener varios tipos. Se muestra un ejemplo en la figura 21 en donde el usuario puede seleccionar el tipo de código 510 que este desea añadir. Esto permite que el editor dentro del bloque haga ajustes en sugerencias de sintaxis y comprobación de errores basándose en el tipo de código. Ejemplos que funcionan bien con bloques normales son C++, C y Ensamblador. Sin embargo, podría usarse cualquier lenguaje tal como JavaScript y Python.

Bloques de código contraíbles: La figura 22A muestra un programa de Modo Mixto 550 con dos Bloques de Código 552 y 554. Cada Bloque de Código tiene un icono de contracción 556 como se muestra que permite que el usuario contraiga el Bloque de Código a un tamaño más pequeño. La figura 22B muestra el mismo programa de Modo Mixto 550 después de que el usuario haya hecho clic en el icono de contracción 556 para el primer Bloque de Código 552. Una porción de las sentencias en C dentro del Bloque de Código 552 se muestra en la línea única del Bloque de Código Contraído 552' junto con el número de líneas de código. La figura 22C muestra el mismo programa de Modo Mixto 550 después de que el usuario haya hecho clic en ambos iconos de contracción 556 para los Bloques de Código 552 y 554. Una porción de las sentencias en C dentro del Bloque de Código se muestra en la línea única de los Bloques de Código Contraídos 552' y 554' junto con el número de líneas de código. Hay muchas otras formas de mostrar un atisbo del código dentro del Bloque de Código Contraído

Método alternativo: La figura 24A muestra código como un ejemplo de un programa completamente textual 600. Como se ilustra adicionalmente en la figura 24B, se proporciona como una realización alternativa de la programación de modo mixto un nuevo entorno de programación textual mixto 602 que incluye un programa 610 que es idéntico al programa completamente textual 600 de la figura 24A. Este método usa un entorno de programación textual que es más como un procesador de textos. Este método alternativo tiene muchas de las mismas ventajas de la programación de modo mixto usando Bloques de Código. Este método puede tener ventajas para usuarios más experimentados que ya prefieren un editor de texto para programación y que desean las ventajas de usar Bloques Gráficos para algunas funciones. Pueden soltarse Bloques Gráficos en el área de texto 602. El área de trabajo de ejemplo en la figura 24B muestra números de línea a la izquierda de cada fila. La línea 1 muestra un bloque Cuando 612, idéntico a los bloques Cuando de tipo Bloque Gráfico proporcionados en las realizaciones previas pero que ahora aparece en un área de texto. La línea 2 muestra una sentencia de código de texto 614 entre dos bloques en la línea 1 y 3. La línea 3 es el comienzo de un bloque Repetir 616 que se extiende hasta la línea 12. El bloque Repetir 616 contiene código textual en las líneas 4, 6, 8, 9 y 10, así como la sentencia Si 618 que empieza en la línea 5.

La figura 24C muestra el mismo programa que en la figura 24B, excepto que los usuarios han hecho clic en el texto para insertar su cursor al final de la línea 2, y han presionado volver para obtener un nuevo número de línea vacío en la línea 3 (n.º de referencia 620). Todos los Bloques Gráficos y líneas de texto posteriores han fluido hacia abajo una fila de forma similar a un editor de texto. Haciendo referencia a continuación también a la figura 24D, se muestra el mismo programa 610, excepto que el usuario ha borrado la línea 2 de texto. Todos los Bloques Gráficos y líneas de texto han fluido hacia arriba una fila. Adicionalmente, el bloque Cuando 612 y el bloque Repetir 616 se han unido ahora entre sí.

La figura 24E muestra un nuevo programa que se está desarrollando en el entorno de programación textual mixto 602 en donde el usuario ha comenzado arrastrando un bloque Cuándo 612 a un área de trabajo 622. El área de trabajo en este ejemplo tiene los números de línea 624.

Ventajas del nuevo método:

La investigación establece que una clara ventaja de la modalidad de bloques es que permite que los estudiantes escriban y compilen rápidamente un programa, y entonces consigan inmediatamente realimentación en cuanto a lo que logra ese programa. Permitiendo que los estudiantes, comenzando con una línea a la vez, manipulen directamente un bloque de programación con texto, se mantienen los beneficios del ciclo de edición-ejecución ajustado. La investigación también nos dice que usar la modalidad de bloques no enseña a los estudiantes un entendimiento conceptual de las variables, debido a que los estudiantes no tienen que hacer nada con los tipos de datos asociados con las variables. Al poder manipular directamente un bloque, puede pedirse a los estudiantes que identifiquen tipos de datos para variables y, por lo tanto, comenzar a construir ese entendimiento conceptual. Debido a que el resto del programa aún puede ser bloques, los estudiantes aún experimentan los beneficios del ciclo de edición-ejecución ajustado.

En la modalidad de bloques, la estructura del bloque define el ámbito. Cuando los estudiantes programan en modalidad de texto, deben usar sintaxis para proporcionar ese ámbito. La investigación establece que este es uno de los errores más frecuentes que cometen los estudiantes cuando hacen una transición de bloque a texto. El problema no hace más que agravarse cuando los estudiantes están usando múltiples sentencias Si y sentencias de anidación. Con estos

nuevos métodos, los estudiantes pueden hacer clic en un bloque, convertirlo en texto, y se les proporciona la sintaxis que determina el ámbito. Este nuevo método estructura el proceso para los estudiantes y aligera la carga cognitiva que los estudiantes experimentan cuando están intentando centrarse en la lógica de su programa. Los estudiantes también pueden pasar gradualmente a añadir sus propios símbolos al ámbito a medida que pasan a la modalidad de texto completo. Este nuevo método ayudará a los estudiantes a construir un entendimiento del ámbito de una forma estructurada. Una vez más, la investigación es clara de que los estudiantes actualmente tienen dificultades para entender el ámbito cuando crean programas puramente de bloques y cuando tienen que crear programas puramente de texto.

Este nuevo método permite que los estudiantes combinen programación de bloques y texto. Un beneficio de usar la modalidad de bloques es que los estudiantes no tienen la tarea de memorizar los nombres de los comandos. Como resultado, los estudiantes experimentan menos carga cognitiva cuando intentan centrarse en su tarea de programación. Estos nuevos métodos estructuran este proceso debido a que los estudiantes no irán directamente de bloque a texto, sino que en su lugar podrán escribir nuevos comandos a medida que comienzan a familiarizarse con ellos, al tiempo que siguen recurriendo a bloques para aquellos comandos que no se memorizan.

Los estudiantes pueden reconocer rápidamente lo que hace un comando de bloque debido a que los nombres del bloque a menudo están en un lenguaje que es más familiar para los estudiantes que los nombres usados en la modalidad de texto. Este nuevo método permite que los estudiantes suelten un bloque gráfico en un bloque de código de texto, permitiendo de este modo que los estudiantes vean el equivalente de texto de un bloque. Esto también permite a los estudiantes ver otra información (por ejemplo, tipos de datos, identificadores) que normalmente es abstraída por el bloque. Esto introduce a los estudiantes a estos conceptos de una forma más estructurada, y permite que los estudiantes sigan experimentando los beneficios del ciclo de edición-ejecución ajustado.

La investigación también nos dice que los estudiantes tienen dificultades para escribir expresiones cuando hacen una transición a la modalidad de texto. Este nuevo método permite al estudiante soltar un nuevo Bloque de Código en el argumento de un bloque gráfico. Los estudiantes pueden ahora practicar escribir expresiones con carga cognitiva limitada, debido a que el resto del programa todavía está utilizando la modalidad de bloques. Una vez más, este nuevo método permite a los usuarios realizar una transición de una forma más estructurada, y permite que los estudiantes sigan experimentando los beneficios del ciclo de edición-ejecución ajustado.

Una de las desventajas de los estudiantes que usan la modalidad de bloques es que los estudiantes recurren a una programación ascendente, mientras que identifican herramientas de bajo nivel (por ejemplo, bloques gráficos) para componer programas. Con este nuevo método, los estudiantes tienen la capacidad de crear bloques que pueden usarse como funciones. Esta capacidad permite a los estudiantes comenzar a emplear programación de arriba abajo, al tiempo que son capaces de dividir desafíos de programación en fragmentos más pequeños y usar funciones para programar esos fragmentos más pequeños.
Un principio de buen diseño instruccional es que el contenido presentado a los estudiantes sea específico, preciso y reducido. En un contexto de programación, la investigación nos dice que hay muchos puntos débiles que los estudiantes experimentan cuando son programadores novatos y cuando intentan pasar de la modalidad de bloques a la de texto. Este nuevo método ayudará a poner en primer plano los conceptos objetivo que se presentan a los estudiantes, debido a que el entorno de programación único impulsará otros conceptos, que no son los objetivos de aprendizaje, en segundo plano. Por ejemplo, si los estudiantes estuvieran realizando una transición desde la modalidad de bloques, como Scratch, a la modalidad de texto, como Python, el aprendizaje no sería preciso debido a que los estudiantes tendrían que aprender un nuevo entorno de programación además de aprender Python. Con este nuevo método, cuando los estudiantes hacen una transición de una modalidad de bloques a texto, no tendrán que aprender un nuevo entorno de programación. Como resultado, el contenido de aprendizaje para los estudiantes puede ser dirigido, preciso y reducido y, por lo tanto, puede evitarse la carga cognitiva. Permitir que los estudiantes permanezcan en un entorno de programación y, por lo tanto, eviten la carga cognitiva no es un logro trivial. La investigación nos dice que los aprendices novatos se vuelven "expertos" cuando los estudiantes pueden almacenar conocimiento dentro de un esquema. La pericia se desarrolla cuando los estudiantes combinan ideas y conceptos en un esquema. Esta pericia no puede desarrollarse cuando la memoria de trabajo de un alumno está sobrecargada con información nueva. Si los estudiantes están cambiando entornos de programación, al tiempo que también intentan aprender la transición entre la modalidad de texto y la modalidad de bloques, es probable que la memoria de trabajo de los estudiantes se sobrecargue con información nueva y adicional. Además, lleva tiempo que los aprendices desarrollen un esquema, por lo tanto, no es realista pensar que los estudiantes dominarán un nuevo entorno de programación con un tutorial rápido y entonces sean capaces de comenzar con los objetivos de aprendizaje reales de aprender un lenguaje basado en texto.

Adicionalmente, otro beneficio de la modalidad de bloques es que la secuenciación dentro de un programa también está determinada por los bloques. Por ejemplo, si un bloque no "encaja" en un punto particular, el usuario sabe que no puede colocarlo allí. Esto ayuda a los programadores novatos a colocar los comandos en la secuencia correcta. Si hay un error en la secuencia, el usuario puede ver rápidamente esa realimentación debido, una vez más, al ciclo de edición-ejecución ajustado. Al colocar la modalidad de texto dentro de la modalidad de bloques, el nuevo método permite que el usuario use las ventajas del ámbito y la secuencia definidos del programa, al tiempo que presenta a los estudiantes las cosas con las que los usuarios suelen tener dificultades cuando comienzan con la programación

basada en texto, tales como la asignaciones de variables, las expresiones, etc. Esto permite al profesor estructurar estos conceptos al tiempo que se mantienen las ventajas de la modalidad de bloques.

Implementación en un Editor de Programación Visual:

Los bloques que se usan en la interfaz de usuario tienen dos archivos de definición asociados. El primero contiene información que describe la apariencia gráfica del bloque, esto incluye cualquier campo de entrada o salida que pueda tener. El segundo archivo describe cómo se convierte un bloque en texto. Por ejemplo, en las figuras 23A a 23B, puede aparecer un bloque 570 que implementa la función de biblioteca C convencional denominada printf. La apariencia del bloque se controla por esta estructura contenida en el archivo de definición.

```
Blockly.Blocks['printf_block'] = {
init: function() {
    this.appendValueInput("TEXT")
    .setCheck(null)
    .appendField("printf_text");
    this.setInputsInline(true);
    this.setPreviousStatement(true, null);
    this.setNextStatement(true, null);
    this.setColour(Blockly.Blocks.console. HUE);
    this.setTooltip('');
}
};
```

La función de generación de código para este bloque adopta la siguiente forma.

```
Blockly.JavaScript['printf_block'] = function(block) {
var value_text =
        Blockly.JavaScript.valueToCode(block,'TEXT',
        Blockly.JavaScript.ORDER_ATOMIC);
// formato para salida de código C
value_text = value_text.replace(/'/g,"'");
// return code
var code = 'printf(' + value_text + ');\n';
return code;
};
```

Esta función recupera los parámetros que se han suministrado al bloque, en este ejemplo solo hay un único campo de entrada, y entonces crea el texto de programa para el lenguaje de programación apropiado, en este caso C.

Los bloques que encierran otros bloques anidados 575 (figura 23B) llamarán recursivamente a las funciones necesarias para construir una sección completa de código. Por ejemplo, el siguiente grupo de bloques contiene una sentencia "si" que recuperaría el código para que se ejecuten bloques anidados cuando la condición se cumpla.

```
Blockly.JavaScript['controls_if] = function(block) {
// Condición if/elseif/else,
var n = 0;
var argument = Blockly.JavaScript.valueToCode(block, 'IF' + n,
    Blockly.JavaScript.ORDER_NONE) || 'false';
var branch = Blockly.JavaScript.statementToCode(block, 'DO' + n);
var code = 'if (' + argument + ') {\n' + branch + '}';
for (n = 1; n <= block.elseifCount_; n++) {
    argument = Blockly.JavaScript.valueToCode(block, 'IF' + n,
    Blockly.JavaScript.ORDER_NONE) || 'false';
    branch = Blockly.JavaScript.statementToCode(block, 'DO' + n);
    code += ' else if (' + argument + ') {\n' + branch + '}';
}
if (block.elseCount_) {
    branch = Blockly.JavaScript.statementToCode(block, 'ELSE');
    code += ' else {\n' + branch + '}';
}
return code + '\n';
};
```

El código final que se genera para la función de ejemplo sería como sigue.

```
/**
```

Función de prueba

```
*/
void foo() {
if (button_pressed == 1) {
printf("Hola Mundo");
}
}
```

En estos ejemplos, se usa JavaScript como el lenguaje para los archivos de definición de bloque y generación de código, también podrían usarse otros lenguajes y formatos para estas definiciones. En implementaciones existentes, la colección completa de bloques se convierte a código en segundo plano. Una vez que se genera el código final completo, puede pasarse a un compilador para entonces a la plataforma objetivo para su ejecución. Todo esto sucede entre bastidores, lo que permite que el estudiante esté protegido del código real. En algunas implementaciones, el código final completo se pone a disposición del usuario. En la nueva implementación, los bloques normales se convierten a código textual en segundo plano como antes, entonces se combinan con el código textual de Bloques de Código en el orden prescrito por el orden de bloques, haciendo de este modo un código final completo. De nuevo se genera el código final completo, que puede pasarse a un compilador para entonces a la plataforma objetivo para su ejecución. Por lo tanto, se mantiene el ciclo de edición-ejecución ajustado.

La nueva implementación permite a los estudiantes ordenar que un único bloque o grupo de bloques genere su código. Un nuevo bloque de "editor de texto" sustituye al bloque o bloques originales, el bloque de editor de texto se describe de la misma forma que otros bloques gráficos con una definición de cómo se mostrará y también cómo debería generar código para presentarlo al compilador. El único campo de entrada que contiene este bloque podría tener muchas de las características de un editor de texto de programadores tradicional, por ejemplo, coloración de sintaxis, plegado de código, sangría automática y compleción de palabras clave. Los bloques gráficos originales pueden vincularse al nuevo bloque y ocultarse de la vista, en la situación en la que el estudiante desea abandonar la edición de texto, los mismos pueden restaurarse, esto sería además de la capacidad de deshacer habitual de la aplicación.

Los bloques que rodean el bloque de editor de texto pueden fusionarse según sea necesario, por ejemplo, si el bloque de código está rodeado por un bloque de sentencia condicional, los dos pueden fusionarse de modo que el editor de texto contenga las sentencias de código originales dentro de la sentencia condicional. Un estudiante puede aumentar la complejidad de su programa arrastrando nuevos bloques gráficos al programa y fusionándolos en el bloque de texto después de que se haya probado su funcionalidad. Los bloques gráficos también pueden arrastrarse directamente dentro del bloque de texto, el código correspondiente se insertaría directamente sin la necesidad de teclearlo.

Ahora se permite que un usuario trabaje tanto en modalidad de bloques como en modalidad de texto. Cuando es el momento de compilar, los bloques se analizan como antes y los bloques de código generan su salida de una forma similar a todos los otros bloques gráficos para completar el programa final pasado al compilador.

REIVINDICACIONES

1. Un sistema, que comprende:

- un procesador;
- un medio de memoria, acoplado al procesador, en donde el medio de memoria almacena instrucciones de programa ejecutables por un sistema informático, y en donde las instrucciones de programa están configuradas para:
  - crear un entorno de codificación gráfica único,
    - en donde el entorno de codificación gráfica único define una pluralidad de bloques de programación gráfica, cada bloque de programación gráfica, de la pluralidad de bloques de programación gráfica, está configurado para representar un elemento de programación predefinido; y
    - en donde el entorno de codificación gráfica único define adicionalmente un bloque de programación de codificación, el bloque de programación de codificación está configurado para representar un bloque de programación para su uso en el entorno de codificación gráfica único y configurado adicionalmente para usar lenguaje de codificación textual convencional dentro del bloque de programación, y
    - en donde se define un modo de vistazo, para que un usuario seleccione un bloque de programación gráfica, para el que el conjunto de instrucciones para el modo de vistazo está configurado para:
      - convertir el bloque de programación gráfica seleccionado en un bloque de programación de codificación,
      - visualizar dentro del entorno de codificación gráfica único el bloque de programación de codificación en una ventana de visualización adyacente al bloque de programación gráfica; y
      - sustituir el bloque de programación gráfica seleccionado completamente con el bloque de programación de codificación dentro del entorno de codificación gráfica único, representando el bloque de programación de codificación de forma idéntica el bloque de programación gráfica seleccionado;
  - crear un programa gráfico dentro del entorno de codificación gráfica único en respuesta a una entrada de usuario, en donde el programa gráfico comprende, en respuesta a la entrada de usuario, al menos un bloque de programación gráfica y al menos un bloque de programación de codificación representado visualmente juntos dentro del entorno de codificación gráfica único y, en donde el al menos un bloque de programación gráfica y el al menos un bloque de programación de codificación se conectan adicionalmente entre sí en el entorno de codificación gráfica único de una forma que indica visualmente y crea físicamente una funcionalidad del programa gráfico de acuerdo con la entrada de usuario y manteniéndose todo dentro del entorno de codificación gráfica único; y
  - generar un programa de salida basándose en el programa gráfico creado dentro del entorno de codificación gráfica único, en donde el programa de salida implementa la funcionalidad del programa gráfico,

y en donde el programa de salida, cuando se ejecuta, controla o bien un objeto virtual o bien un objeto físico de acuerdo con la funcionalidad definida por usuario del programa gráfico.

2. El sistema de la reivindicación 1, en donde las instrucciones de programa configuradas para crear el entorno de codificación gráfica definen adicionalmente un conjunto de instrucciones, almacenadas en la memoria, para definir un modo de edición de variables, y en donde uno o más de los bloques de programación gráfica, de la pluralidad de bloques de programación gráfica, está configurado para incluir un elemento variable establecido por un usuario que activa el modo de edición de variables.

3. El sistema de la reivindicación 1, en donde el conjunto de instrucciones para el modo de vistazo está configurado adicionalmente para crear un bloque de programación de codificación equivalente al bloque de programación gráfica seleccionado, y en donde el bloque de programación de codificación es accesible para editar con lenguaje de codificación textual convencional dentro del entorno de codificación gráfica único.

4. El sistema de la reivindicación 1, en donde las instrucciones de programa configuradas para crear el entorno de codificación gráfica definen adicionalmente un conjunto de instrucciones, almacenadas en la memoria, para codificar por color dos o más bloques de programación gráfica con diferentes colores predefinidos dentro del entorno de codificación gráfica único.

5. El sistema de la reivindicación 4, en donde el conjunto de instrucciones de código de colores está configurado además para codificar por colores el lenguaje de programación textual en la ventana de visualización adyacente a los uno o más bloques de programación gráfica de tal modo que el color del lenguaje de programación textual coincide con el color del bloque de programación gráfica.

6. El sistema de la reivindicación 1, en donde las instrucciones de programa configuradas para crear el entorno de codificación gráfica único definen adicionalmente un conjunto de instrucciones, almacenadas en la memoria, para definir un modo de conversión dentro del entorno de codificación gráfica único, y en donde el modo de conversión está

configurado para que un usuario seleccione uno o más bloques de programación gráfica, en donde, tras la activación, el conjunto de instrucciones para el modo de conversión está configurado para:

- convertir los uno o más bloques de programación gráfica seleccionados en un lenguaje de programación textual convencional, y
- crear uno o más bloques de programación de codificación equivalentes a los uno o más bloques de programación gráfica seleccionados, y en donde los uno o más bloques de programación de codificación son accesibles para editar con lenguaje de codificación textual convencional.

7. El sistema de la reivindicación 1, en donde las instrucciones de programa configuradas para crear el entorno de codificación gráfica único definen adicionalmente un conjunto de instrucciones, almacenadas en la memoria, para definir un bloque de argumento de programación gráfica para su uso en la creación del programa gráfico, y en donde el bloque de argumento de programación gráfica está configurado como un bloque gráfico con un segmento de argumento embebido dentro del bloque gráfico, y el conjunto de instrucciones configurado adicionalmente para aceptar lenguaje de codificación textual convencional en el segmento de argumento.

8. El sistema de la reivindicación 1, en donde las instrucciones de programa configuradas para crear el entorno de codificación gráfica único definen adicionalmente un conjunto de instrucciones, almacenadas en la memoria, para:

convertir automáticamente un bloque gráfico a lenguaje de codificación textual convencional dentro del entorno de codificación gráfica único, e insertar el lenguaje de codificación textual convencional directamente en el bloque de programación de codificación definido en el programa gráfico dentro del entorno de codificación gráfica único, en una posición definida por un usuario y dentro del bloque de programación de codificación, cuando el bloque gráfico es seleccionado por un usuario y el usuario define dicha posición para la inserción.

9. El sistema de la reivindicación 1, en donde las instrucciones de programa configuradas para crear el entorno de codificación gráfica único definen adicionalmente un conjunto de instrucciones, almacenadas en la memoria, para fusionar bloques, y en donde el conjunto de instrucciones para fusionar bloques está configurado para:

añadir automáticamente un segundo bloque de programación de codificación a un primer bloque de programación de codificación dentro del entorno de codificación gráfica único, definiendo un bloque de programación de codificación fusionado que comprende: tanto un segundo lenguaje de codificación textual convencional definido por el segundo bloque de programación de codificación; como un primer lenguaje de codificación textual convencional definido por el primer bloque de programación de codificación.

10. El sistema de la reivindicación 9, en donde el conjunto de instrucciones para fusionar bloques está configurado adicionalmente para insertar el segundo lenguaje de codificación textual convencional en una posición dentro del primer lenguaje de codificación textual convencional seleccionado por un usuario.

11. El sistema de la reivindicación 1, en donde las instrucciones de programa configuradas para crear el entorno de codificación gráfica único definen adicionalmente un conjunto de instrucciones, almacenadas en la memoria, para identificar errores en el bloque de programación de codificación, y en donde el conjunto de instrucciones para identificar errores está configurado para:

- comprobar un lenguaje de codificación textual convencional definido por usuario dentro del bloque de programación de codificación para determinar si el programa de salida puede ejecutarse apropiadamente para controlar o bien el objeto virtual o bien un objeto físico de acuerdo con la funcionalidad definida por usuario; y
- cambiar automáticamente el color del lenguaje de codificación textual convencional definido por usuario dentro del entorno de codificación gráfica único cuando el programa de salida no logra ejecutarse apropiadamente.

12. El sistema de la reivindicación 11, en donde el conjunto de instrucciones para identificar errores está configurado adicionalmente para cambiar el color del lenguaje de codificación textual convencional definido por usuario antes de que se genere el programa de salida.

13. El sistema de la reivindicación 1, en donde el entorno de codificación gráfica único incluye además un conjunto de instrucciones para definir un modo de programación mixto, y en donde el modo de programación mixto está configurado para:

- crear un entorno de programación de texto dentro del entorno de codificación gráfica único, en donde el entorno de programación de texto define líneas de texto de codificación,
- recibir bloques de programación gráfica, y
- recibir lenguaje de codificación textual convencional en las líneas de texto de codificación; y
- crear el programa gráfico en respuesta a una entrada de usuario, en donde el programa gráfico comprende, en respuesta a la entrada de usuario, al menos un bloque de programación gráfica y un lenguaje de codificación textual convencional interconectados en el entorno de codificación gráfica único que indica visualmente una funcionalidad del programa gráfico de acuerdo con la entrada de usuario.

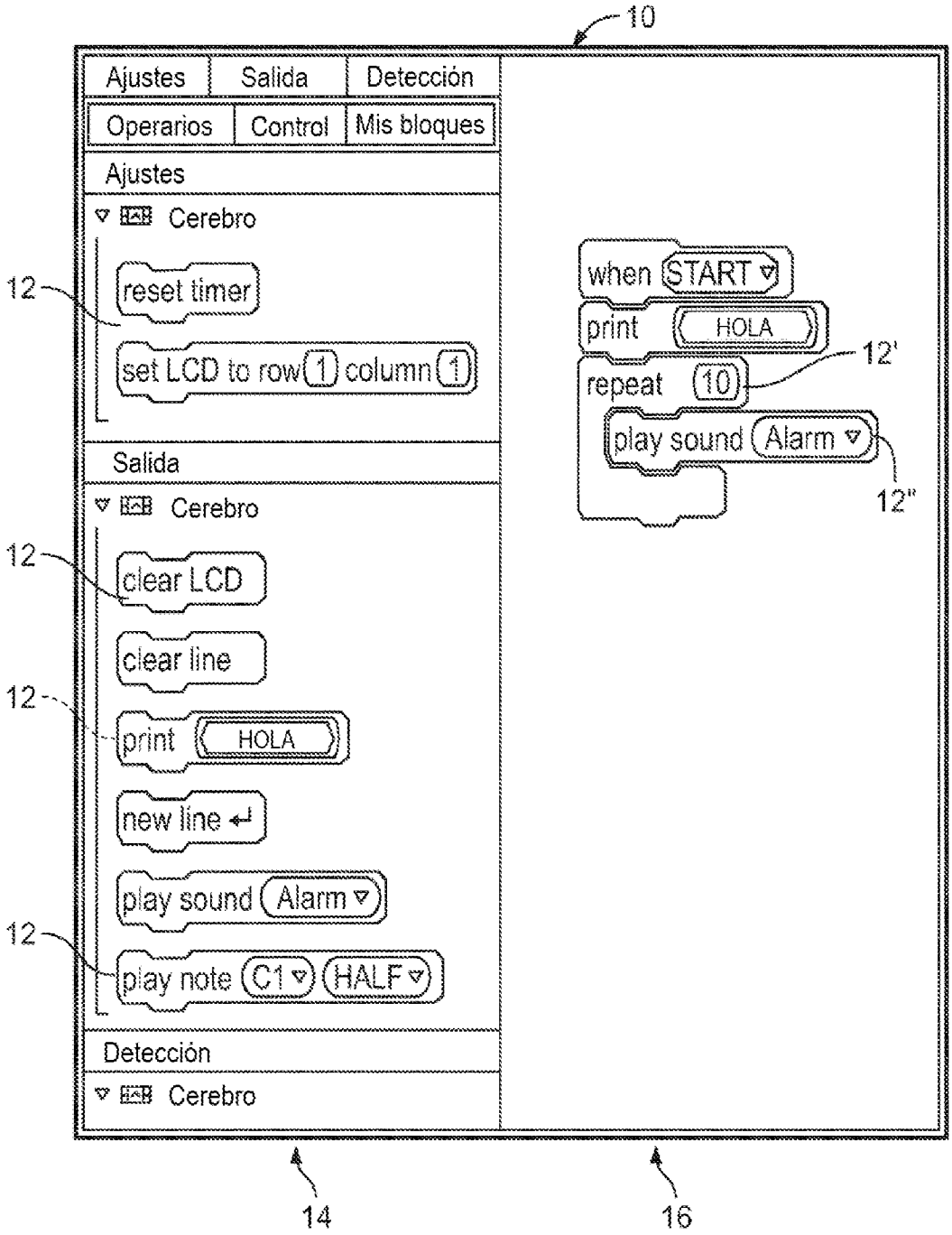

FIG. 1
(Técnica anterior)

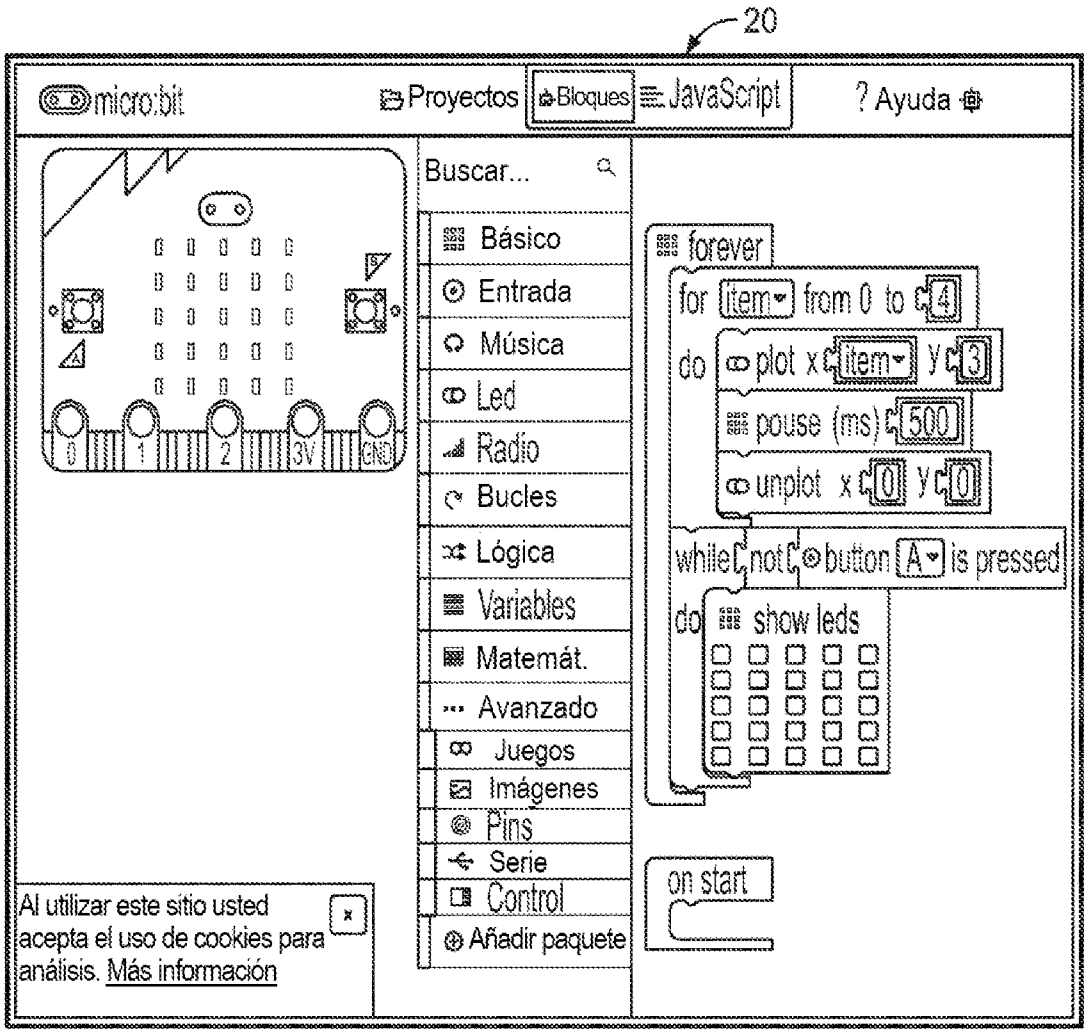

FIG. 2
(Técnica anterior)

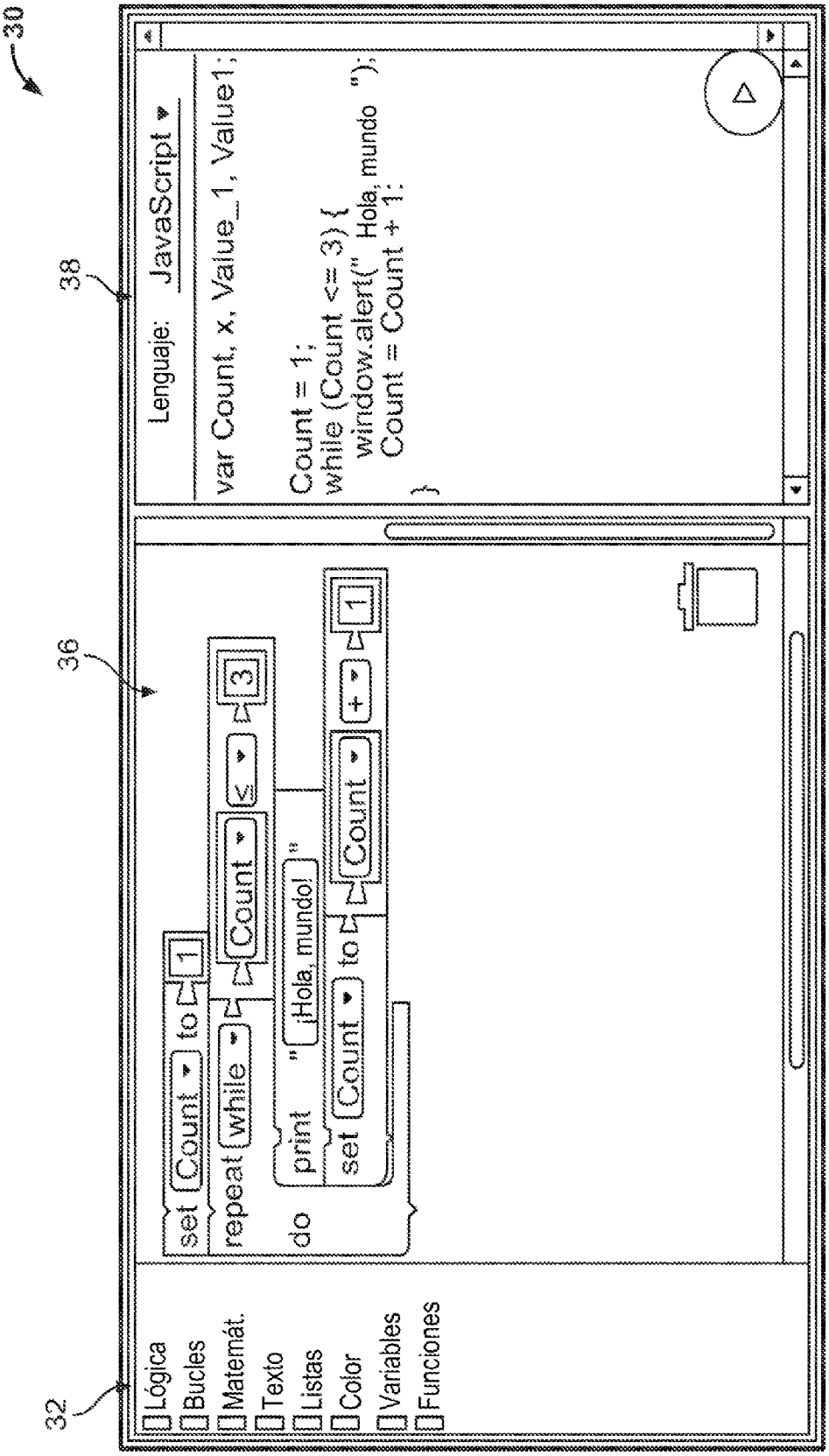

FIG. 3
(Técnica anterior)

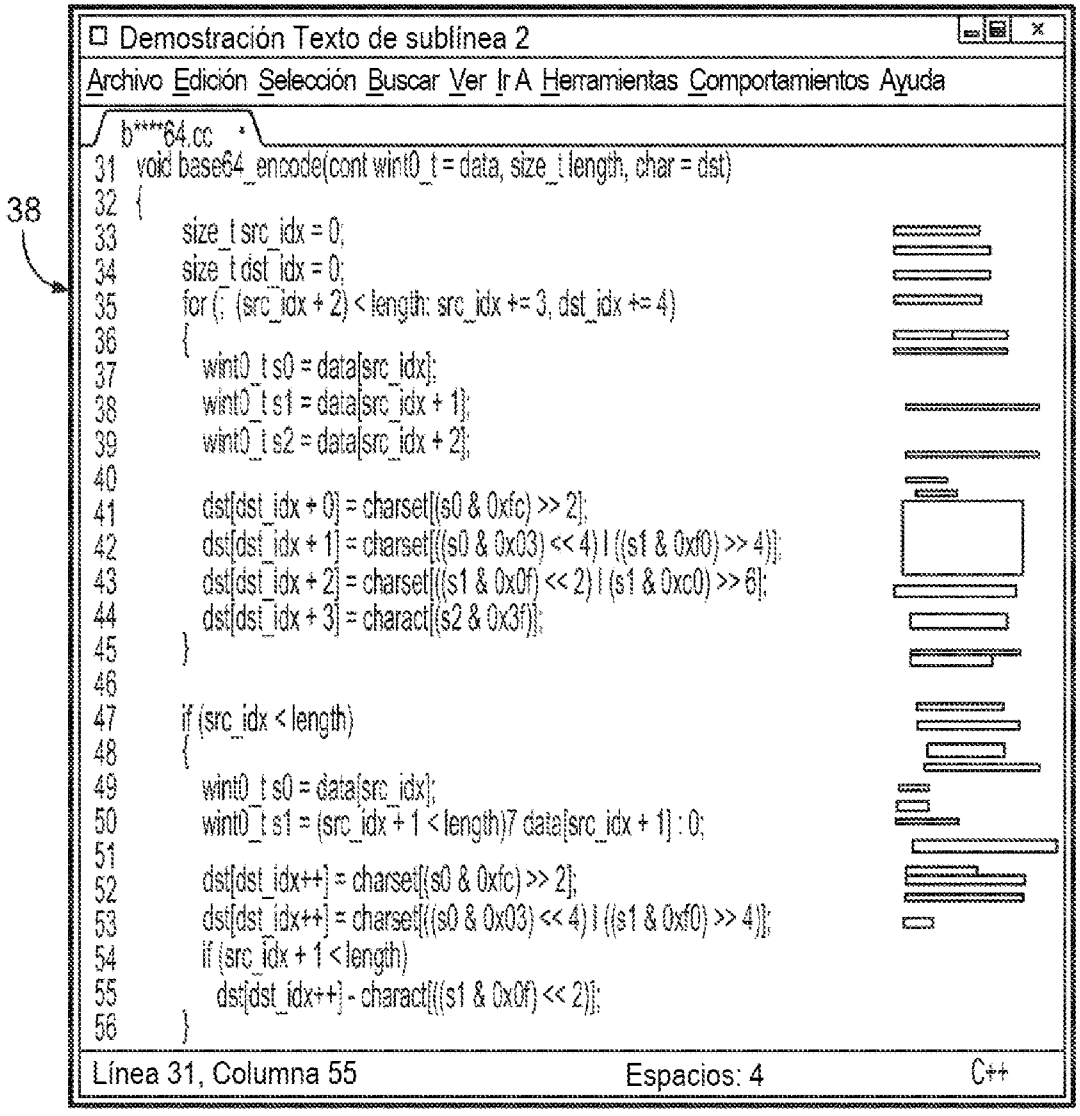

FIG. 4
(Técnica anterior)

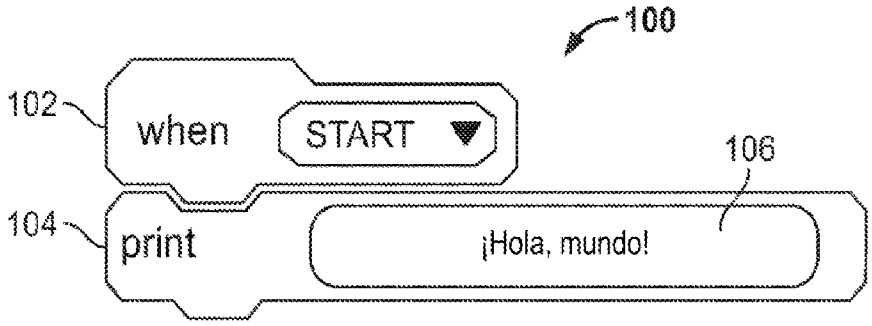
100
102
when
START
106
104
print
¡Hola, mundo!

FIG. 5A

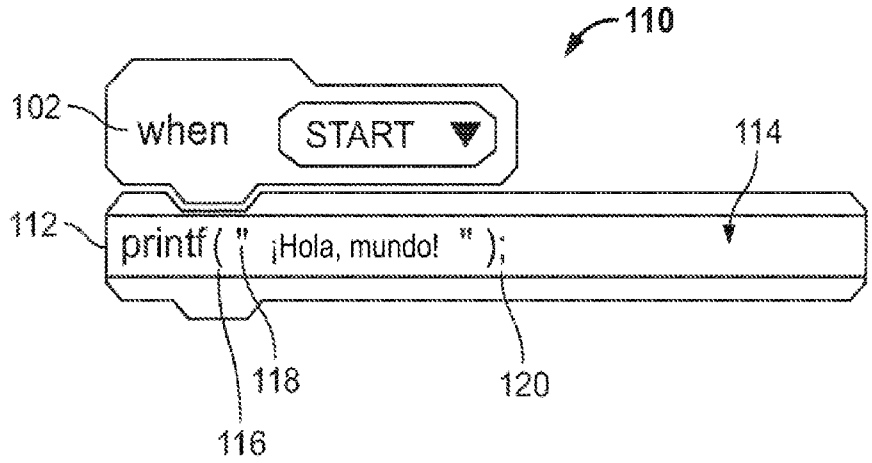
110
102
when
START
114
112
printf (" ¡Hola, mundo! ");
118
120
116

FIG. 5B

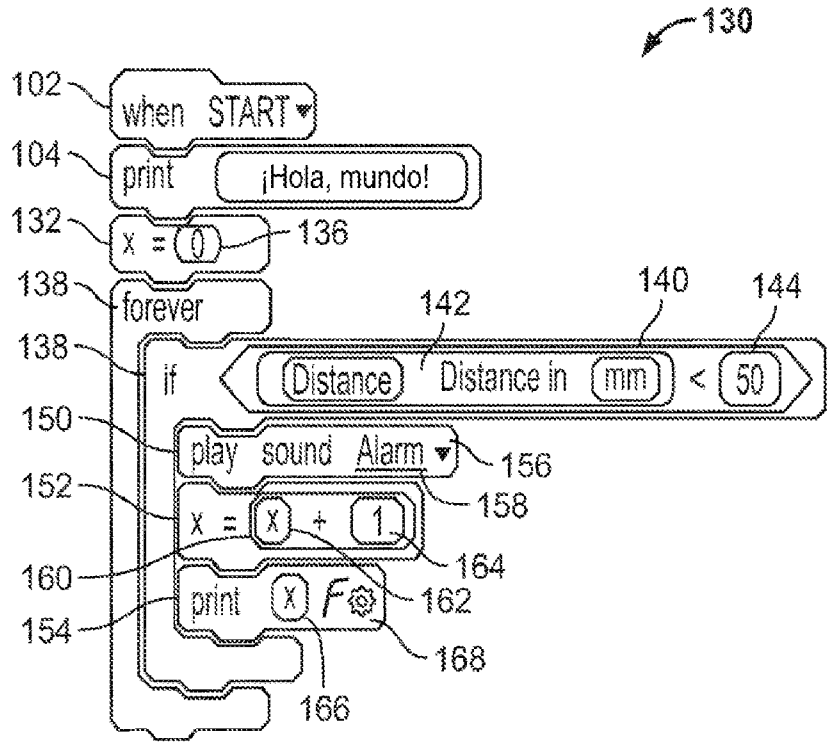
130
102
when START
104
print ¡Hola, mundo!
132
x = 0
136
138
forever
140
142
144
138
if Distance Distance in mm < 50
150
play sound Alarm
156
152
158
x = x + 1
164
160
print x
162
154
168
166

FIG. 6A

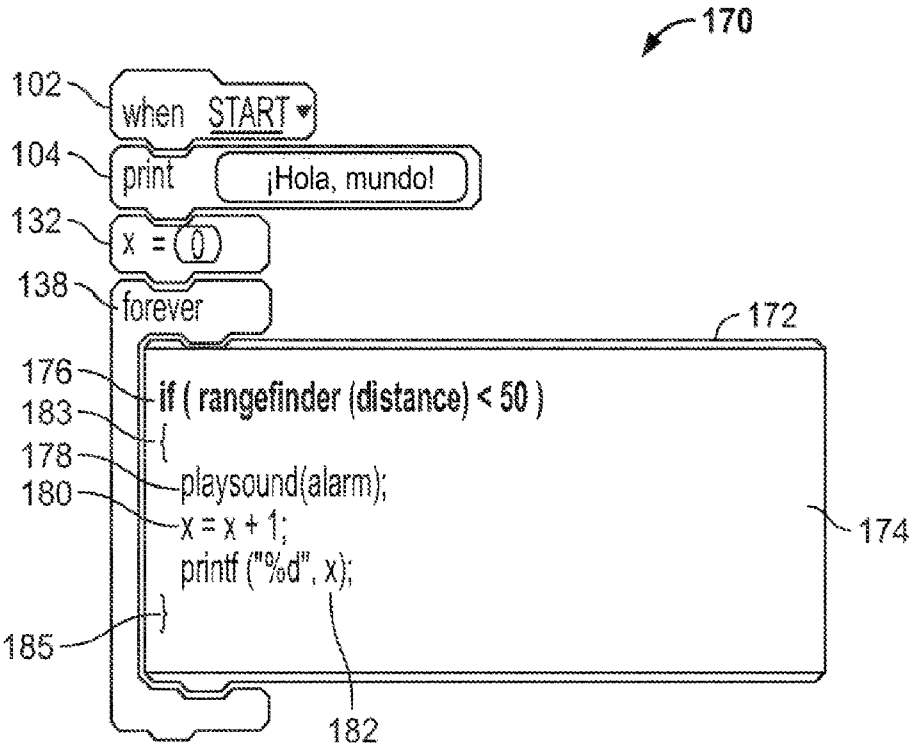
170
102
when START
104
print ¡Hola, mundo!
132
x = 0
138
forever
172
176
if (rangefinder (distance) < 50)
183
{
178
playsound(alarm);
180
x = x + 1;
printf ("%d", x);
174
185
182

FIG. 6B

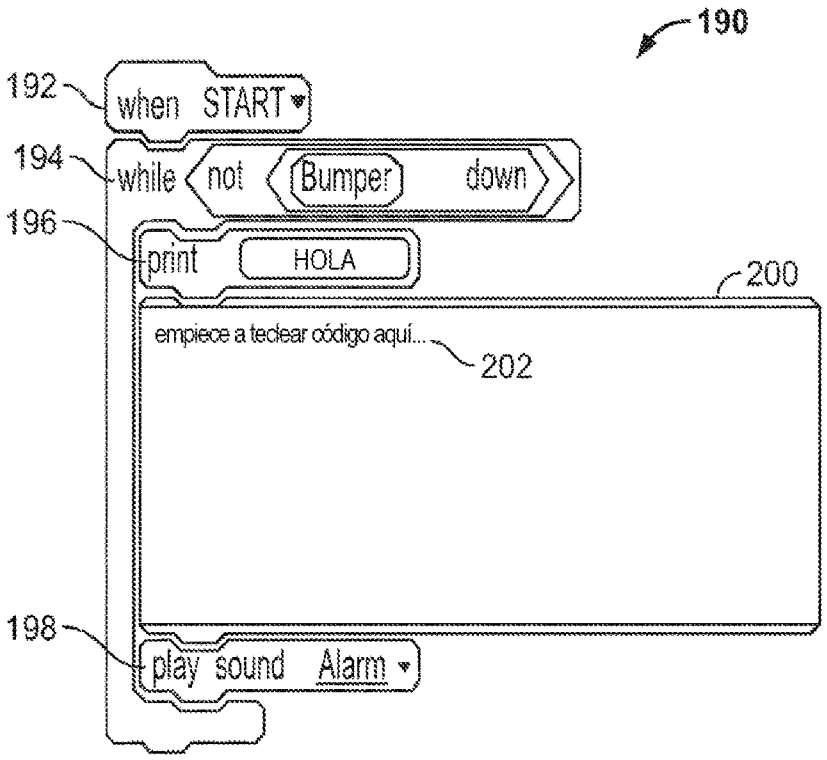
190
192
when START
194
while not Bumper down
196
print HOLA
200
empiece a teclear código aquí...
202
198
play sound Alarm

FIG. 7

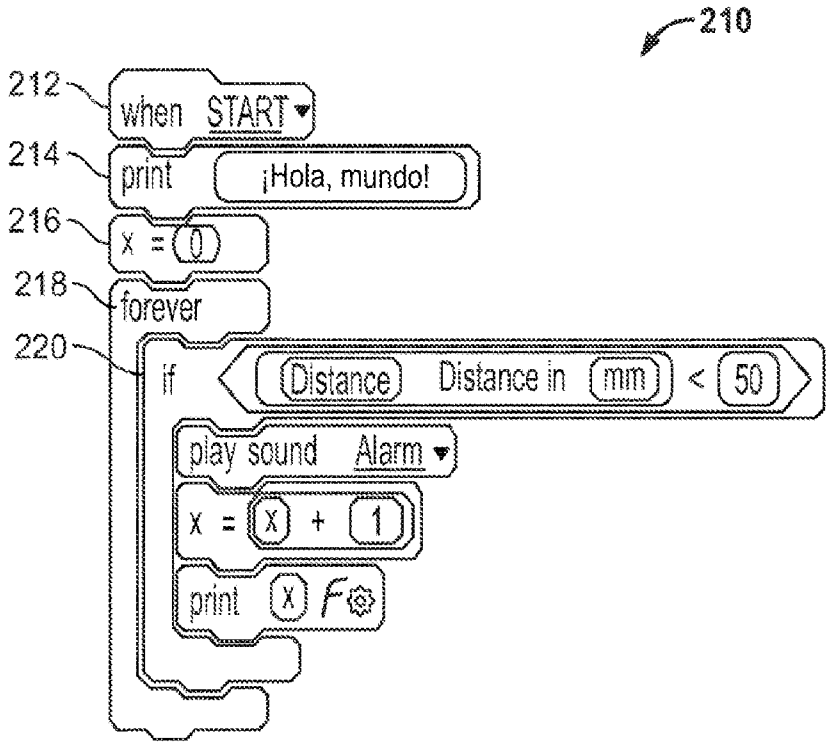
210
212
when START
214
print ¡Hola, mundo!
216
x = 0
218
forever
220
if Distance Distance in mm < 50
play sound Alarm
x = x + 1
print x

FIG. 8A

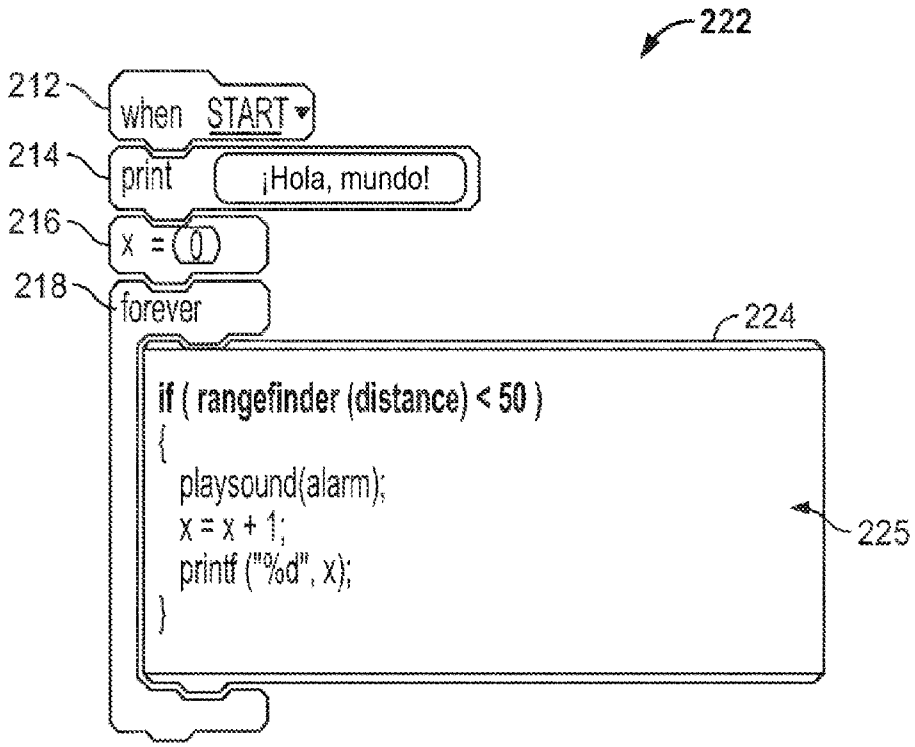
222
212
when START
214
print ¡Hola, mundo!
216
x = 0
218
forever
224
if (rangefinder (distance) < 50)
{
playsound(alarm);
x = x + 1;
printf ("%d", x);
}
225

FIG. 8B

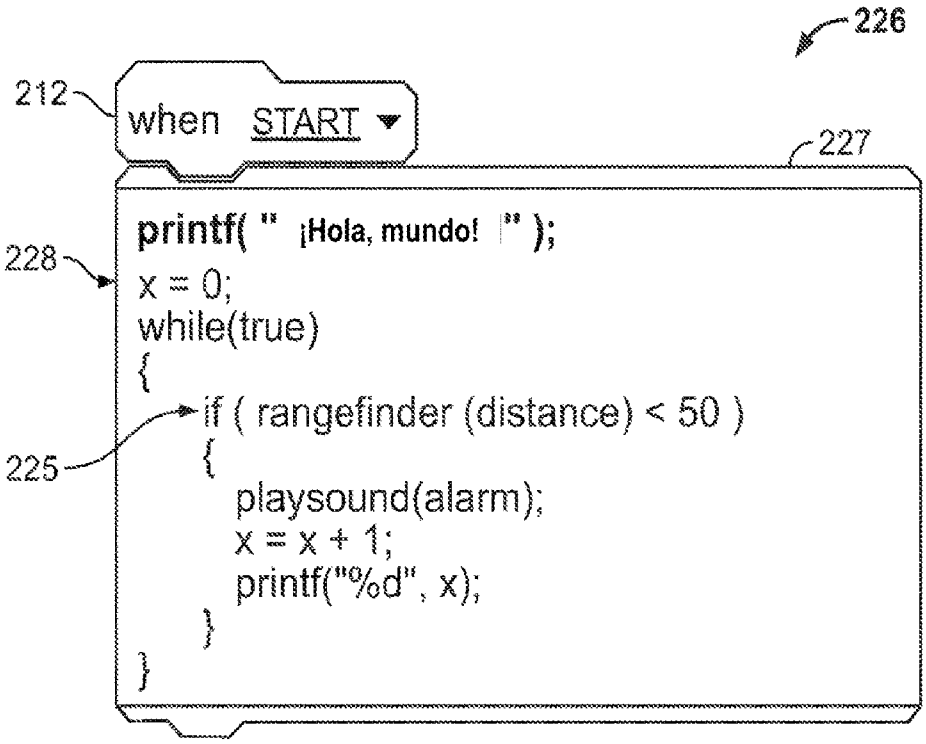
226
212
when START
227
printf(" ¡Hola, mundo! ");
228
x = 0;
while(true)
{
if (rangefinder (distance) < 50)
225
{
playsound(alarm);
x = x + 1;
printf("%d", x);
}
}

FIG. 8C

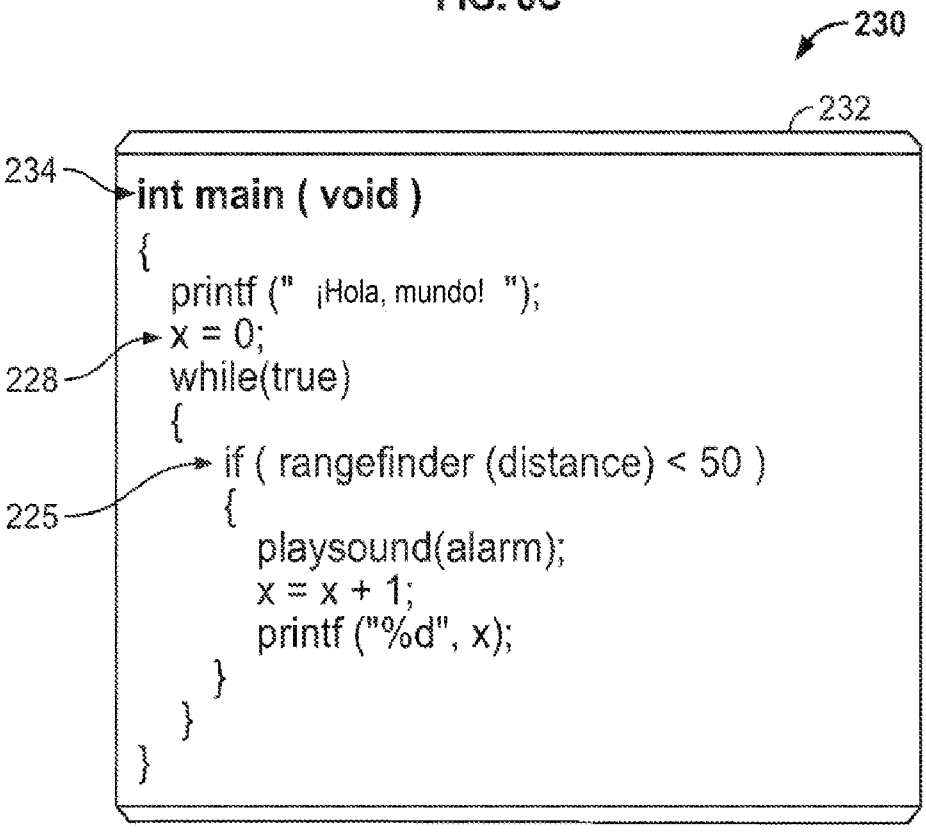
230
232
234
int main (void)
{
printf (" ¡Hola, mundo! ");
x = 0;
228
while(true)
{
if (rangefinder (distance) < 50)
225
{
playsound(alarm);
x = x + 1;
printf ("%d", x);
}
}
}

FIG. 8D

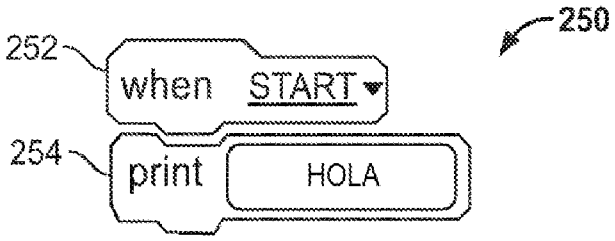

FIG. 9A

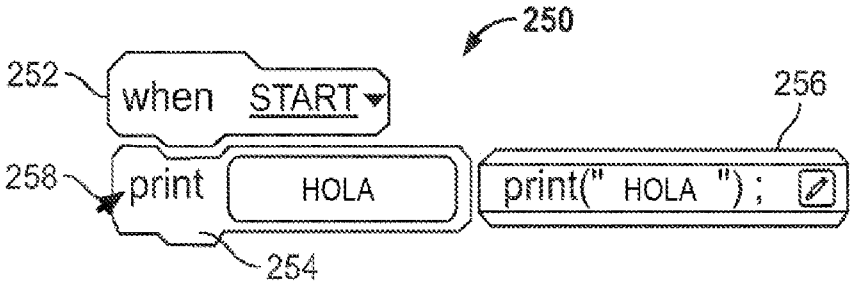

FIG. 9B

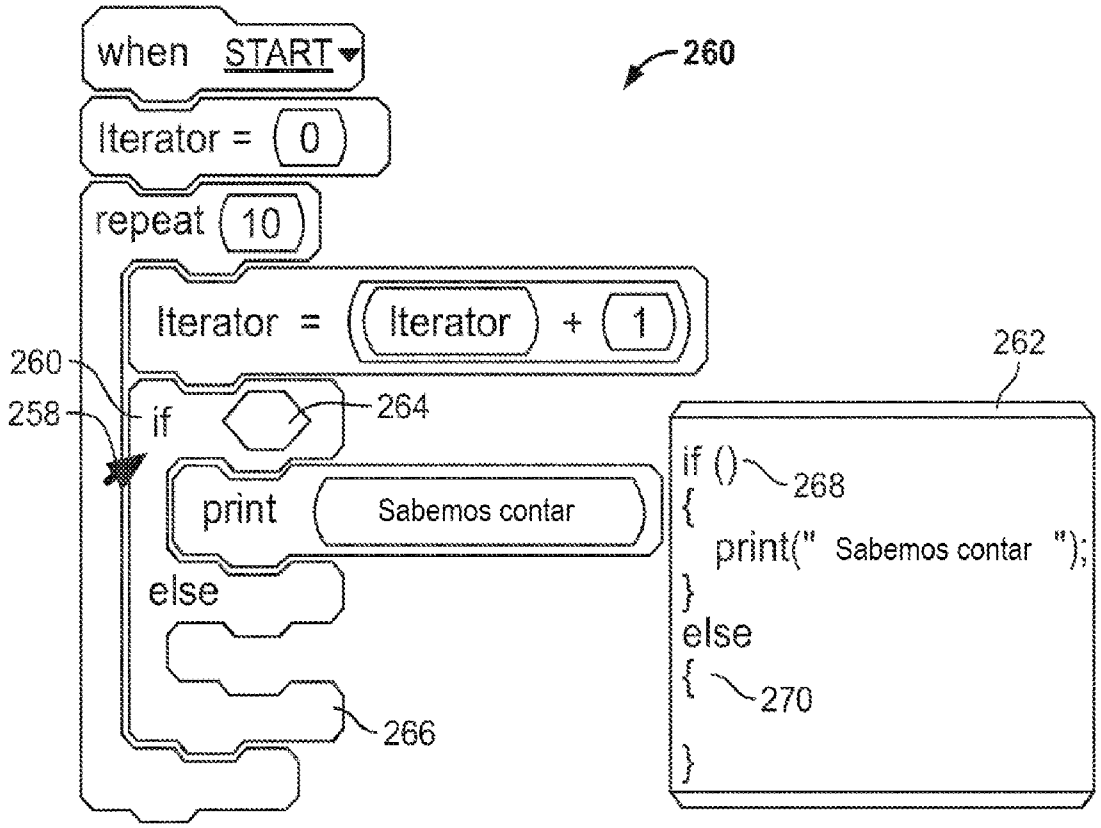

FIG. 9C

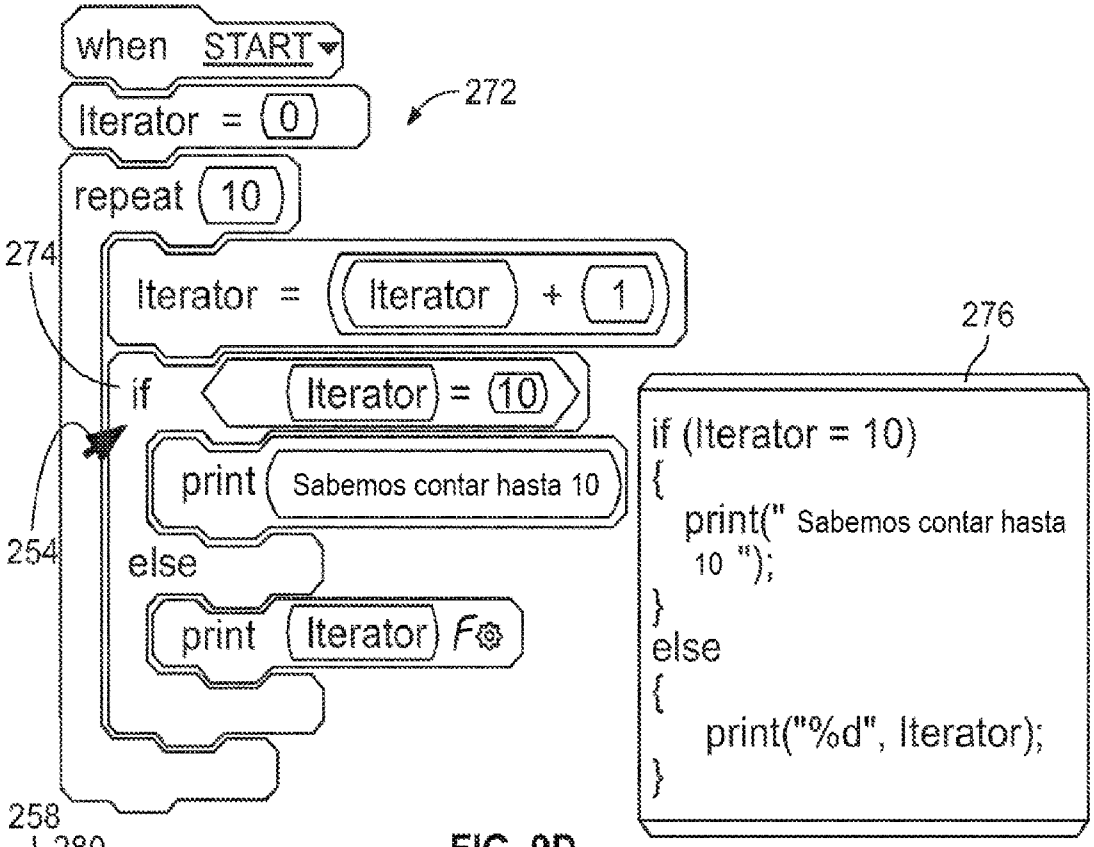
when START
Iterator = 0
272
repeat 10
274
Iterator = Iterator + 1
if Iterator = 10
print Sabemos contar hasta 10
254
else
print Iterator
276
if (Iterator = 10)
{
print(" Sabemos contar hasta 10 ");
}
else
{
print("%d", Iterator);
}

FIG. 9D

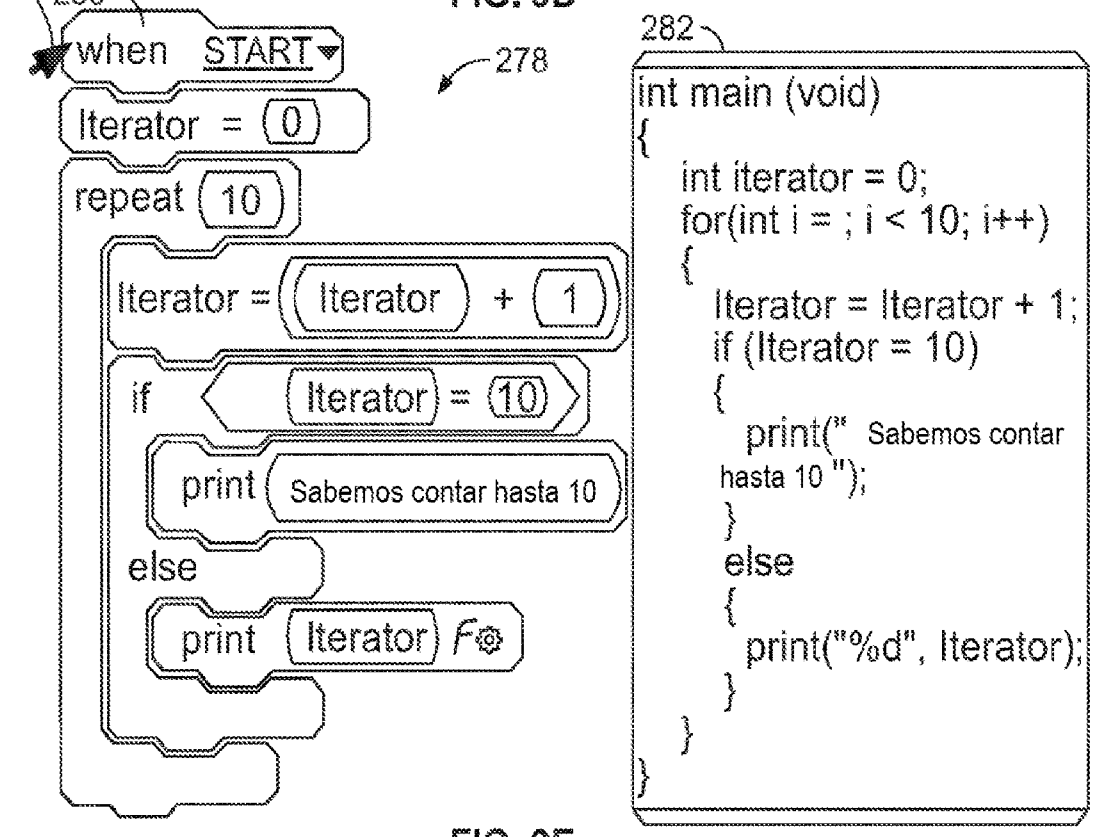
258
280
when START
278
Iterator = 0
repeat 10
Iterator = Iterator + 1
if Iterator = 10
print Sabemos contar hasta 10
else
print Iterator
282
int main (void)
{
int iterator = 0;
for(int i = ; i < 10; i++)
{
Iterator = Iterator + 1;
if (Iterator = 10)
{
print(" Sabemos contar hasta 10 ");
}
else
{
print("%d", Iterator);
}
}
}

FIG. 9E

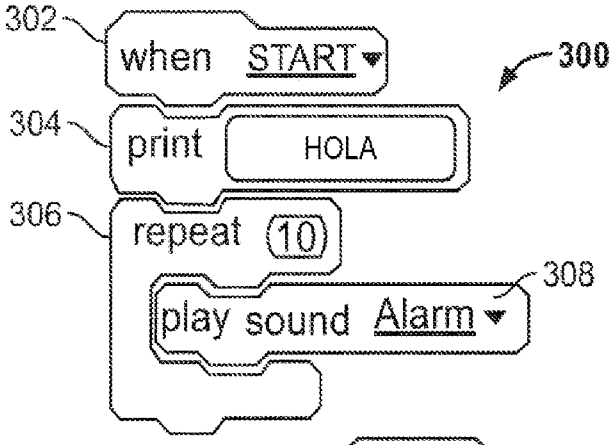

FIG. 10A

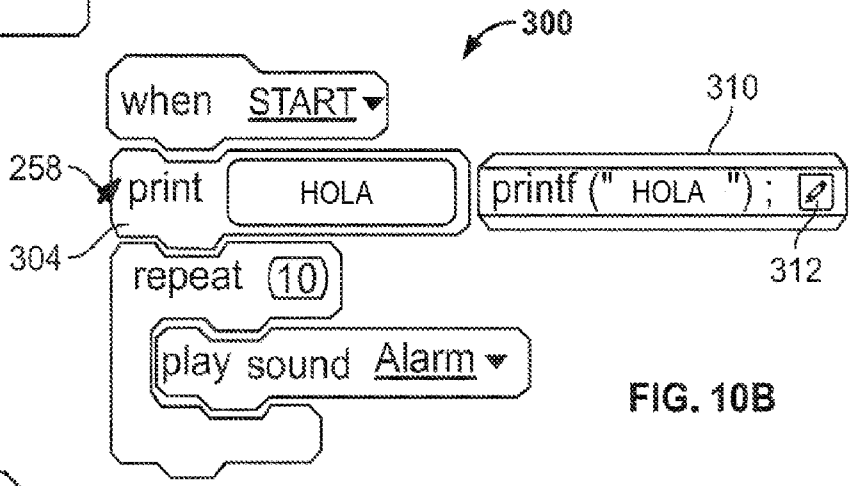

FIG. 10B

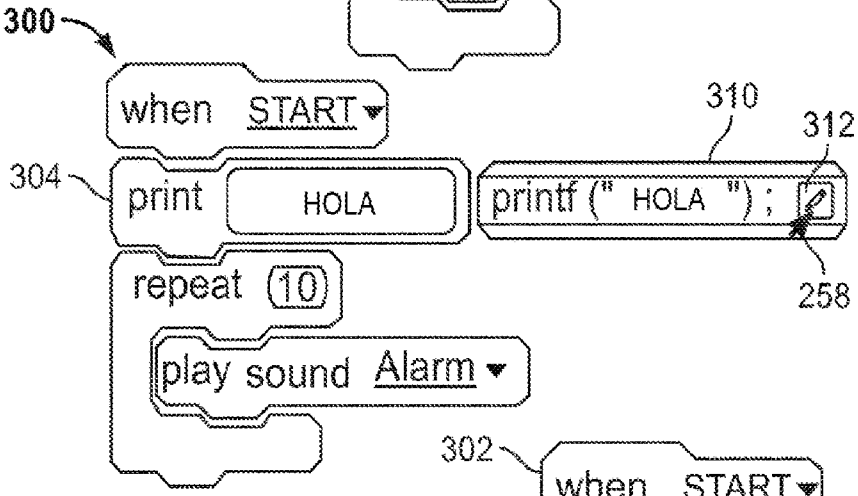

FIG. 10C

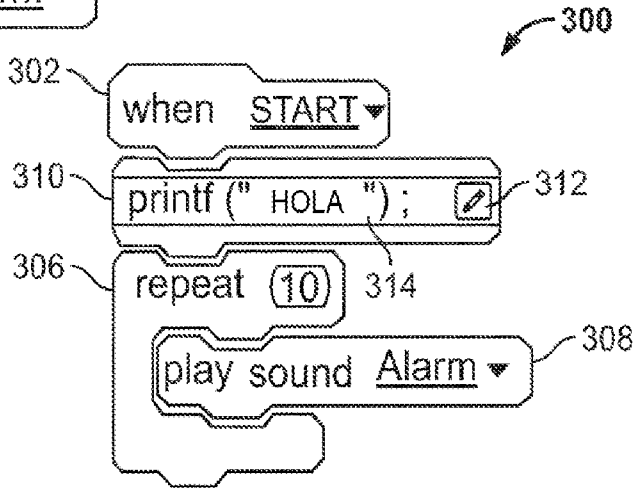

FIG. 10D

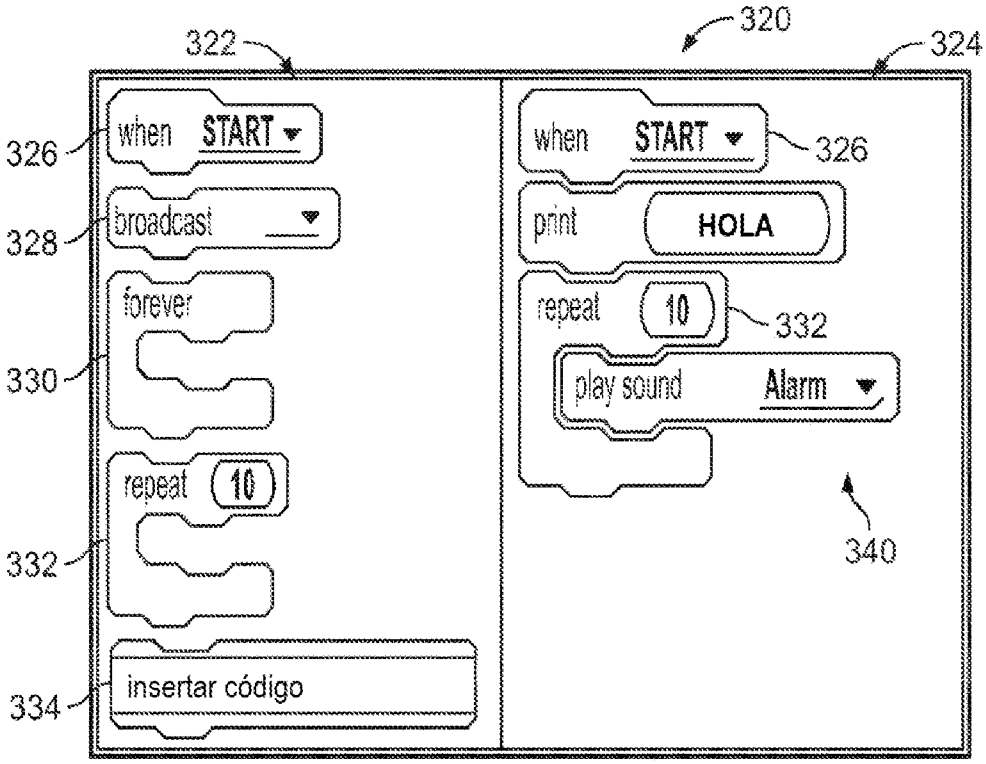

FIG. 11A

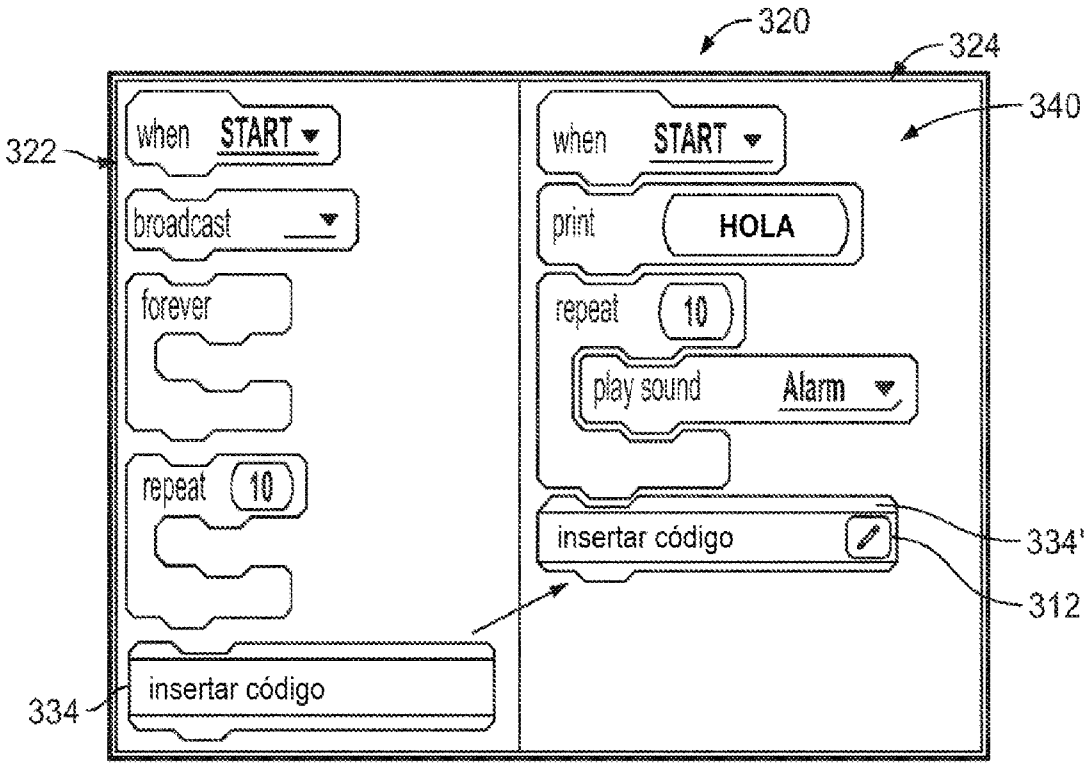

FIG. 11B

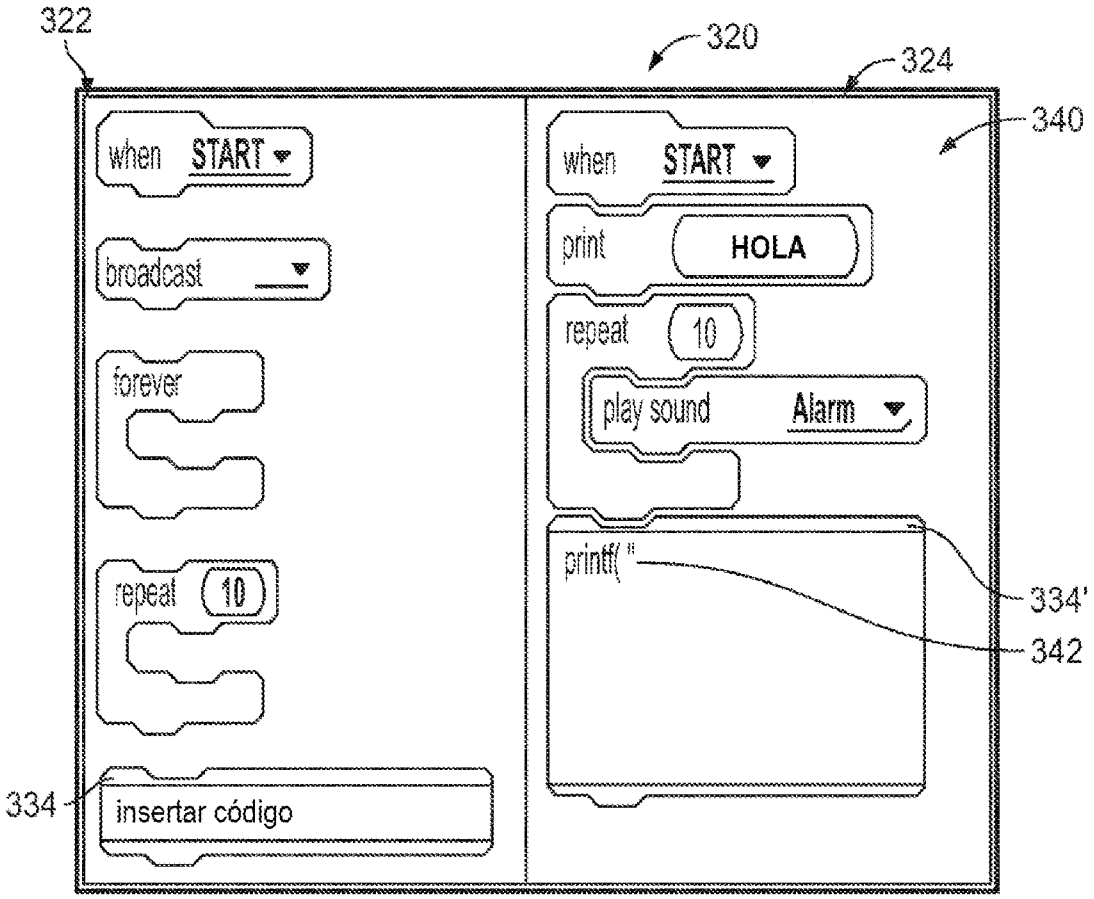
322
320
324
340
when START
broadcast
forever
repeat 10
334
insertar código
when START
print HOLA
repeat 10
play sound Alarm
printf("
334'
342

FIG. 11C

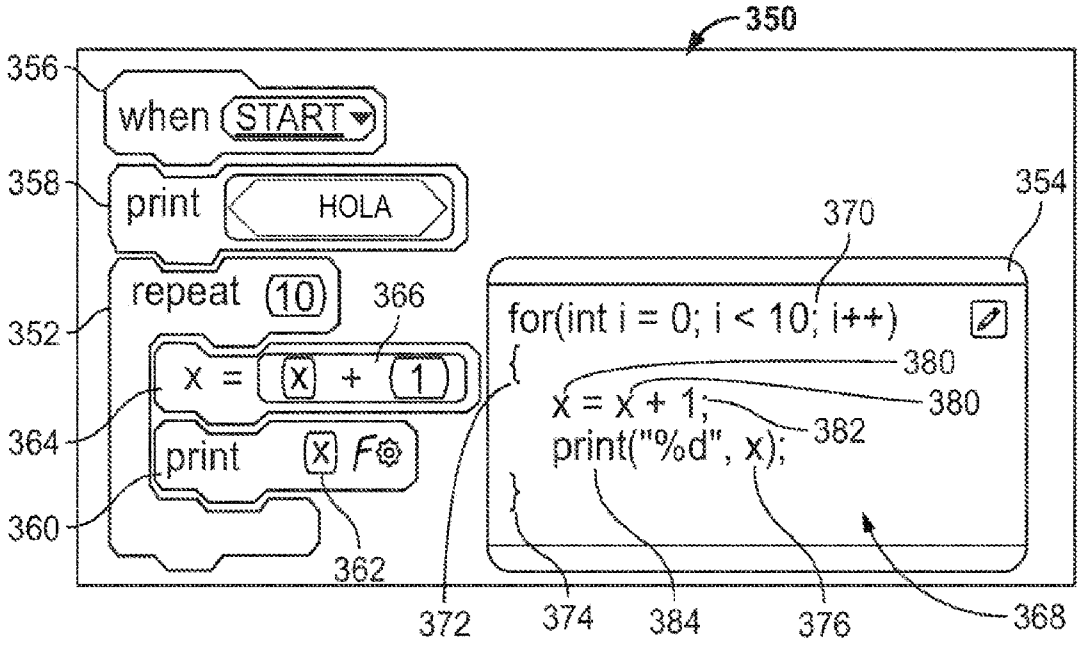

FIG. 12A

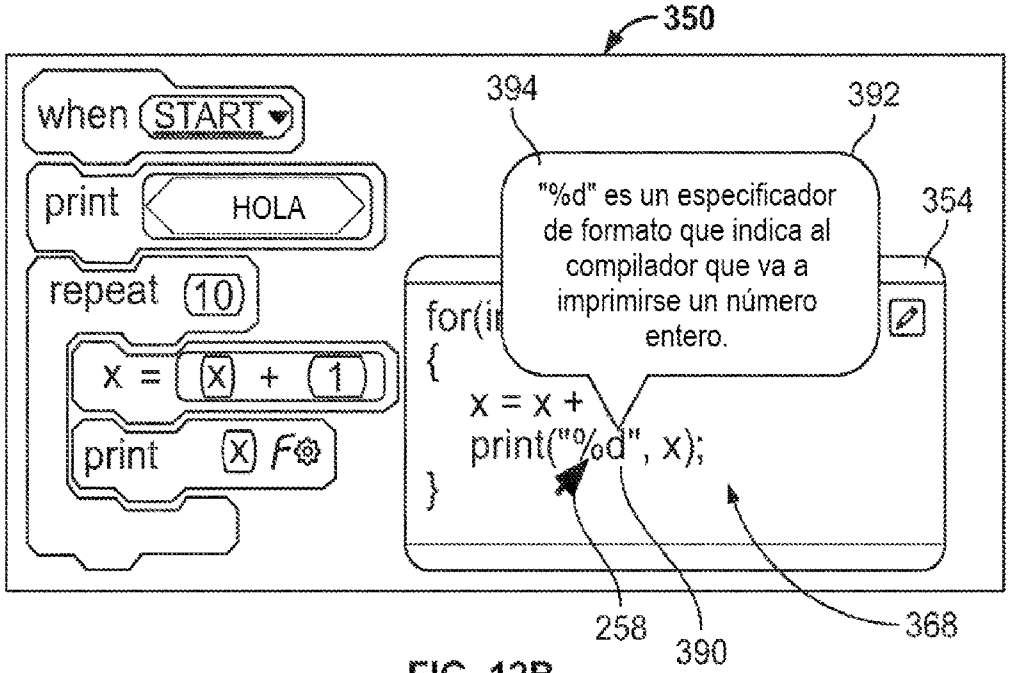

FIG. 12B

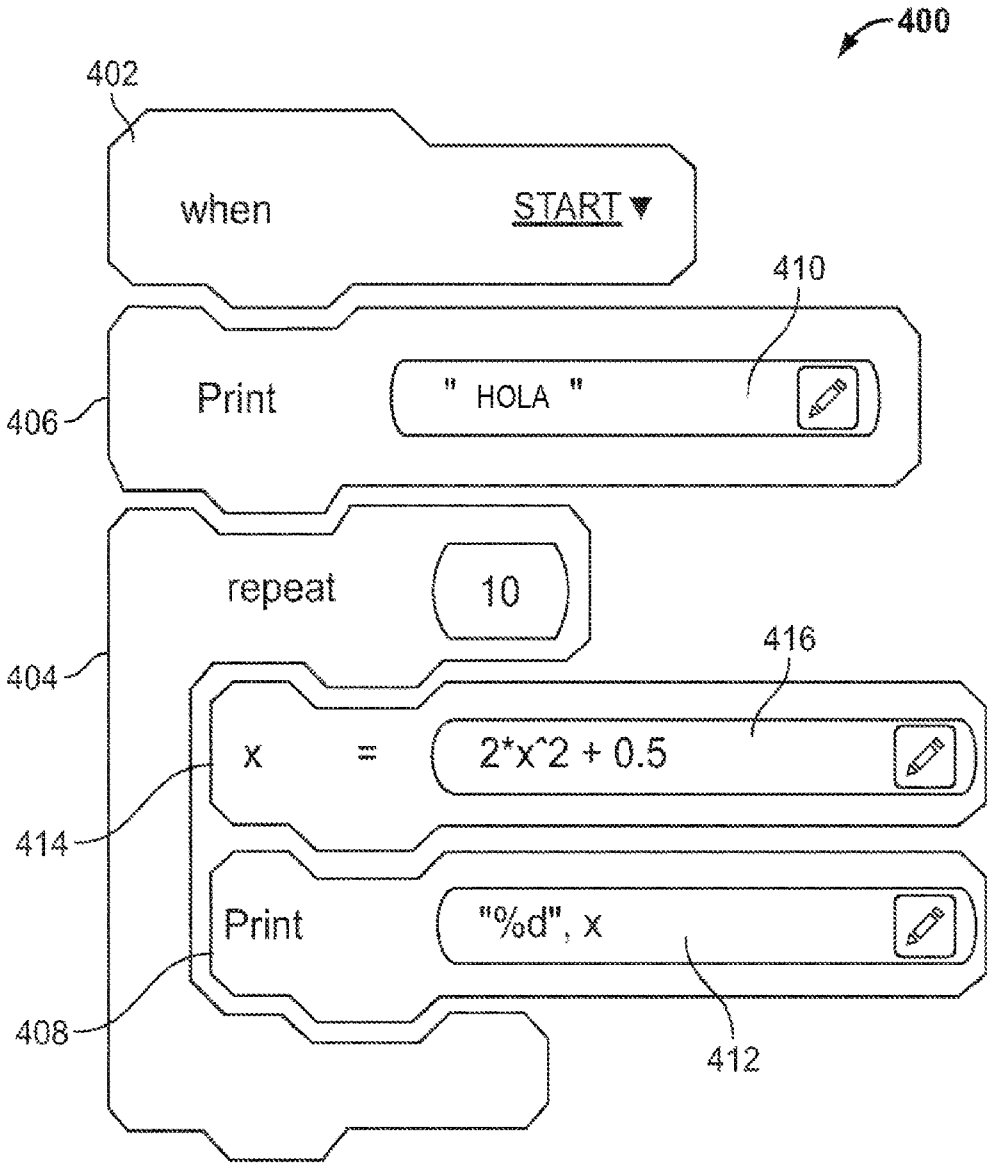
400
402
when
START
410
Print
" HOLA "
406
repeat
10
404
416
x
=
2*x^2 + 0.5
414
Print
"%d", x
408
412

FIG. 13

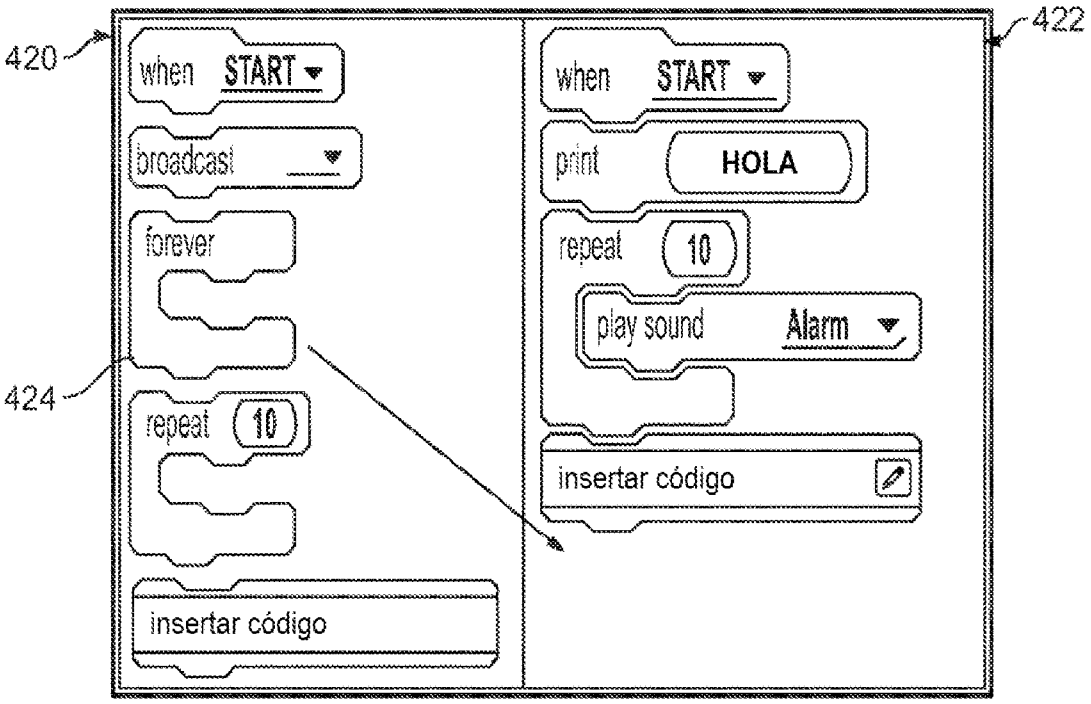

FIG. 14A

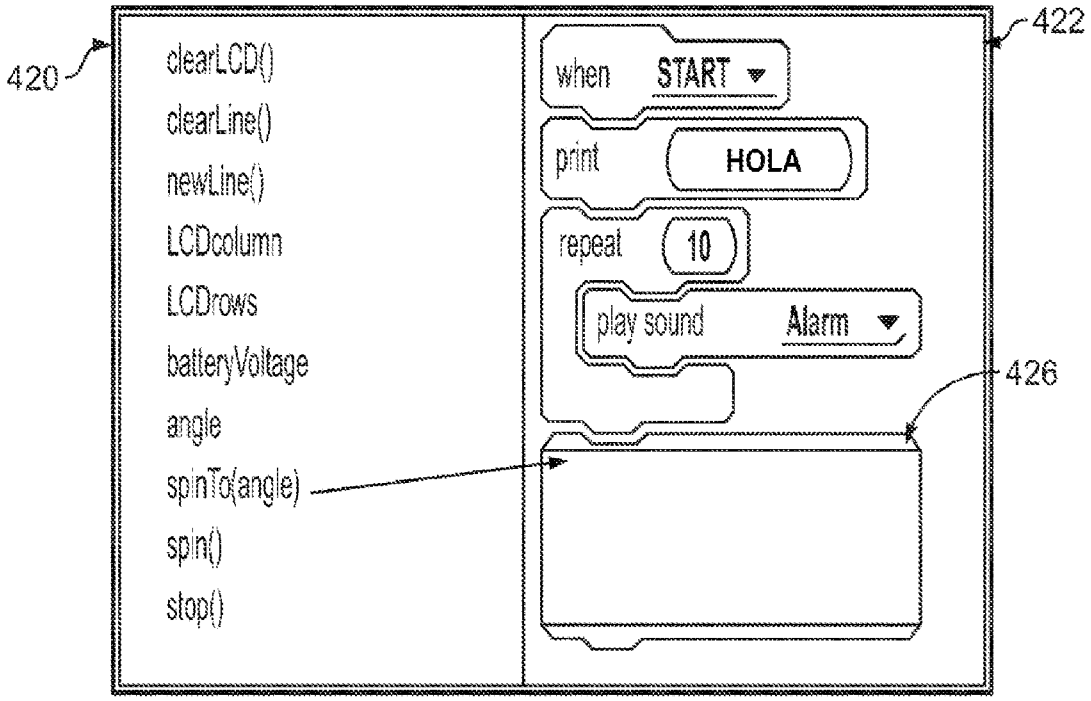

FIG. 14B

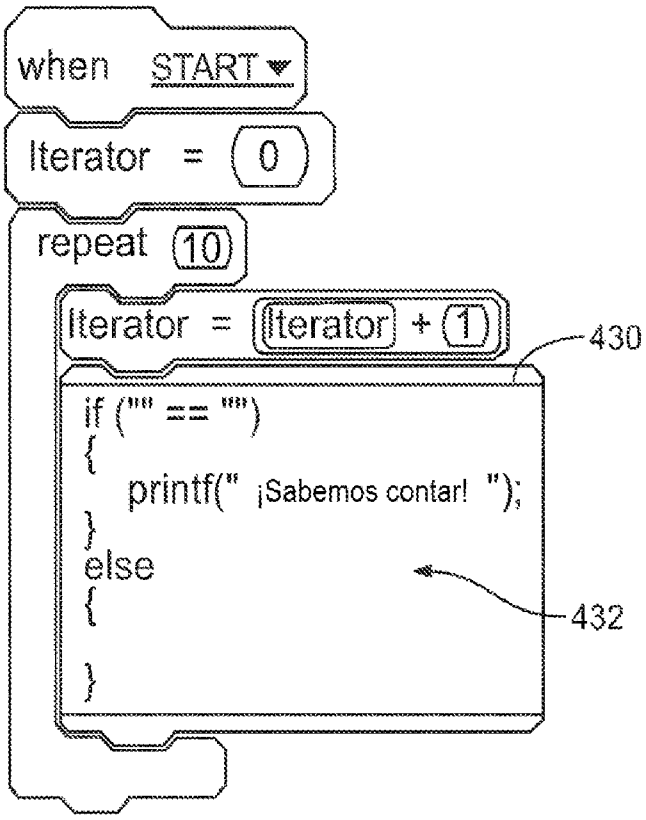

FIG. 15A

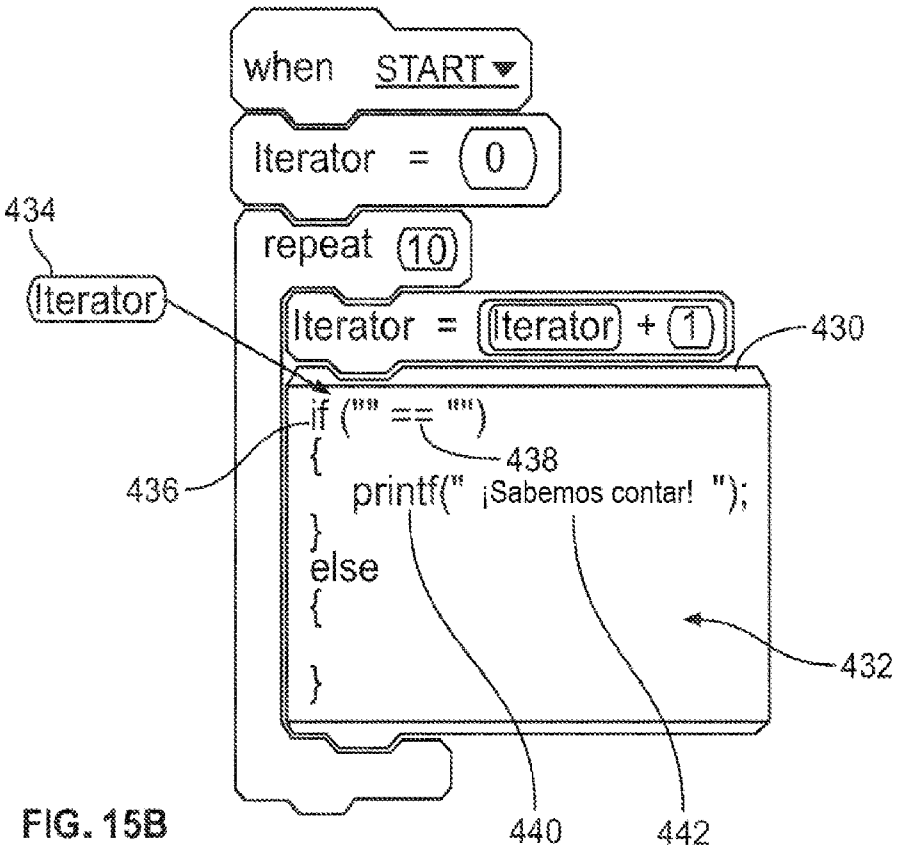

FIG. 15B

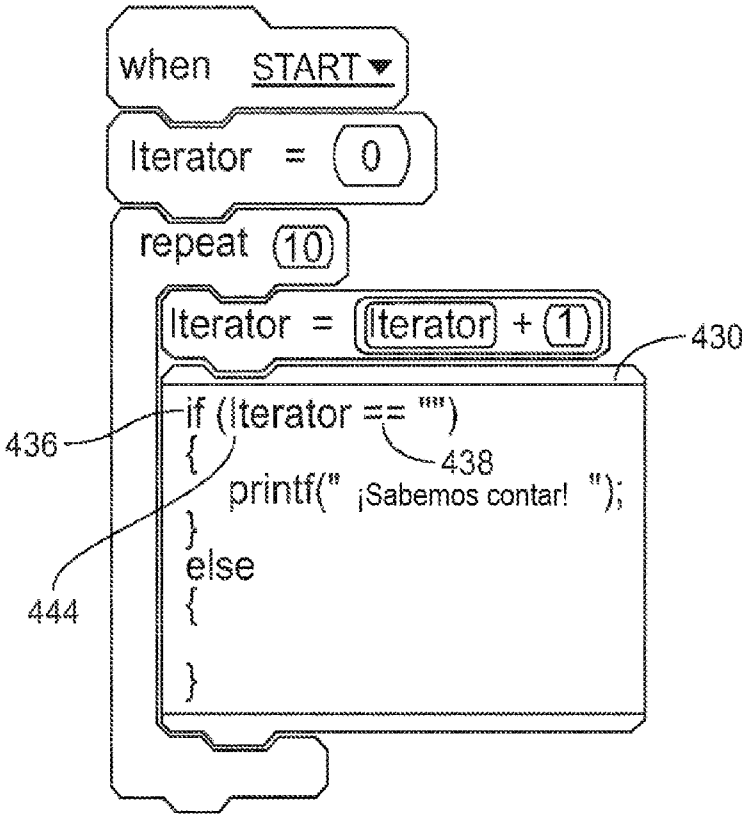

FIG. 15C

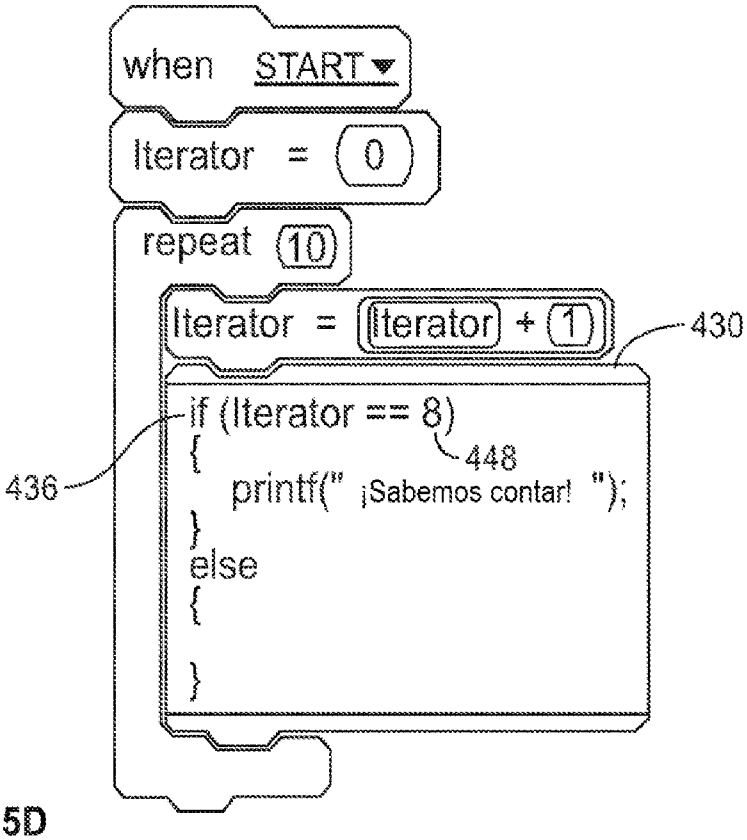

FIG. 15D

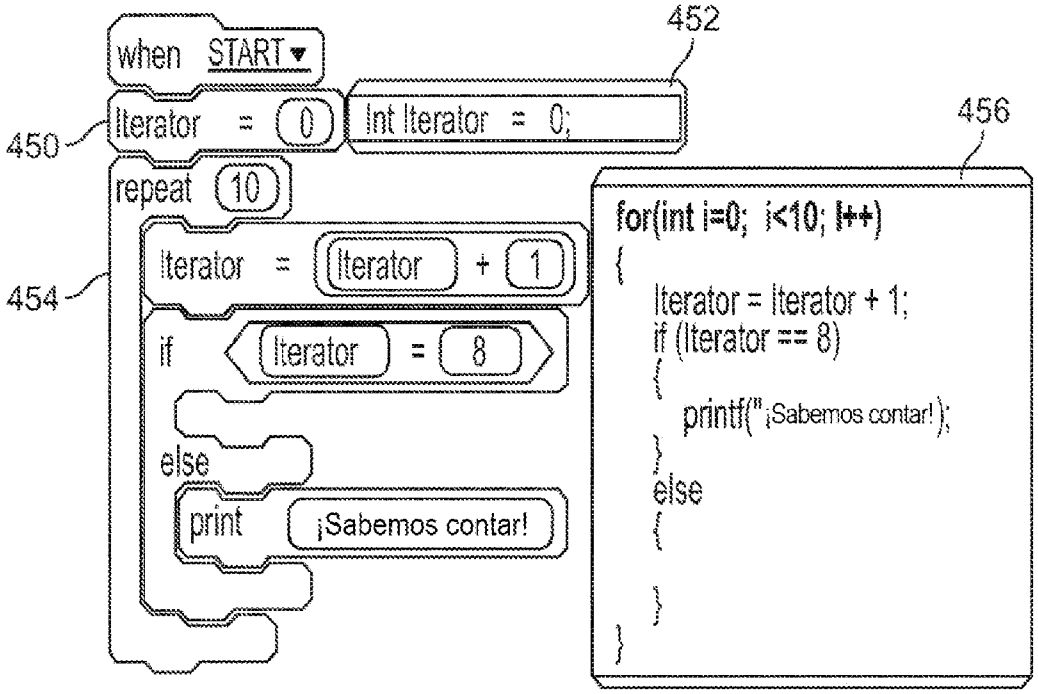
when START
Iterator = 0
int Iterator = 0;
repeat 10
Iterator = Iterator + 1
if Iterator = 8
else
print ¡Sabemos contar!
for(int i=0; i<10; i++)
{
Iterator = Iterator + 1;
if (Iterator == 8)
{
printf("¡Sabemos contar!");
}
else
{
}
}
450
452
454
456

FIG. 16

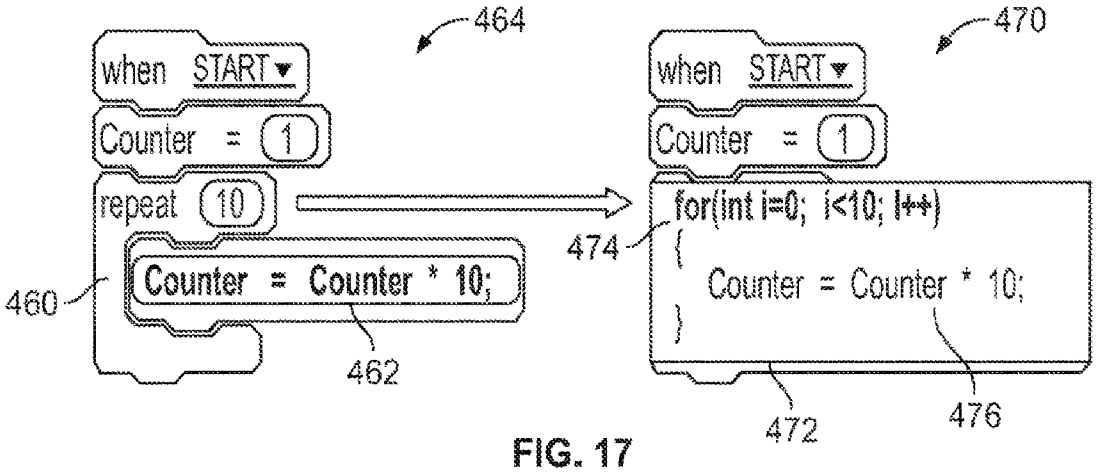
when START
Counter = 1
repeat 10
Counter = Counter * 10;
for(int i=0; i<10; i++)
{
Counter = Counter * 10;
}
460
462
464
470
472
474
476

FIG. 17

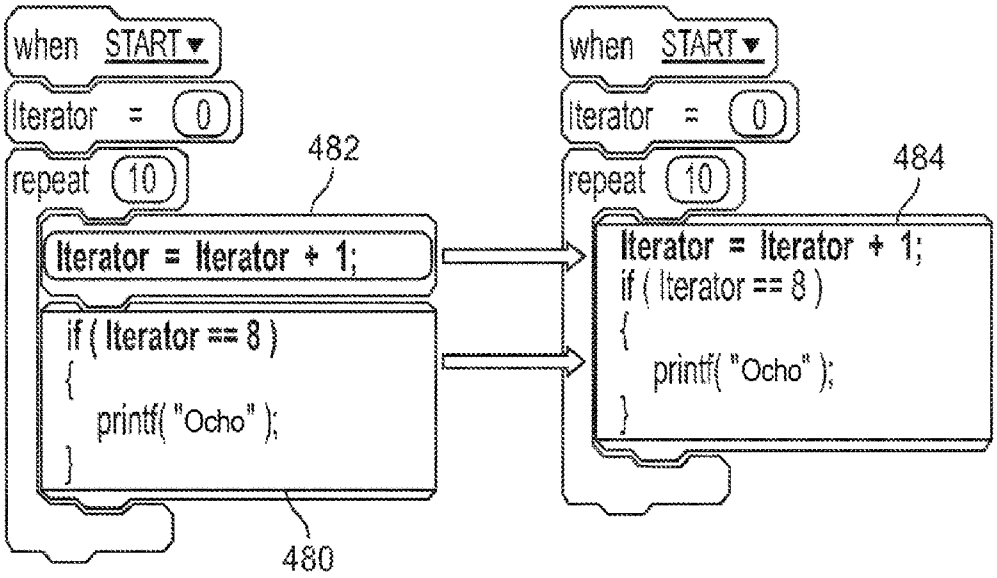
when START
Iterator = 0
repeat 10
Iterator = Iterator + 1;
if (Iterator == 8)
{
printf("Ocho");
}
482
480
when START
Iterator = 0
repeat 10
Iterator = Iterator + 1;
if (Iterator == 8)
{
printf("Ocho");
}
484

FIG. 18

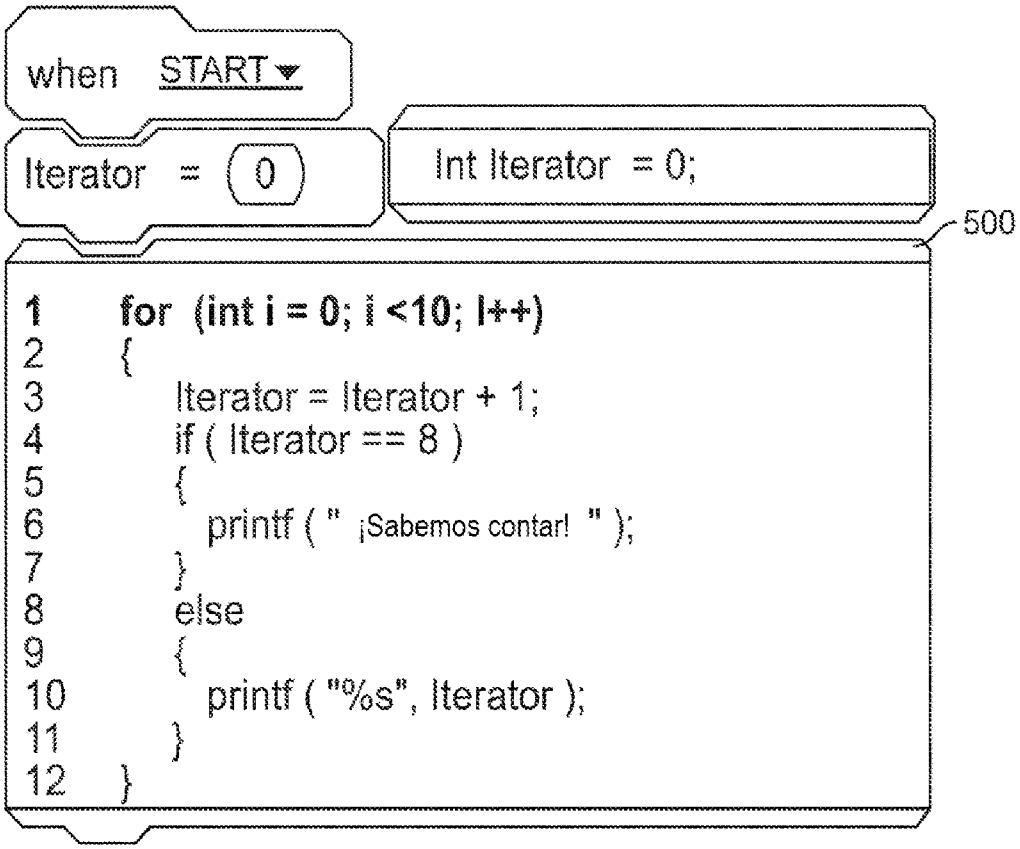

FIG. 19A

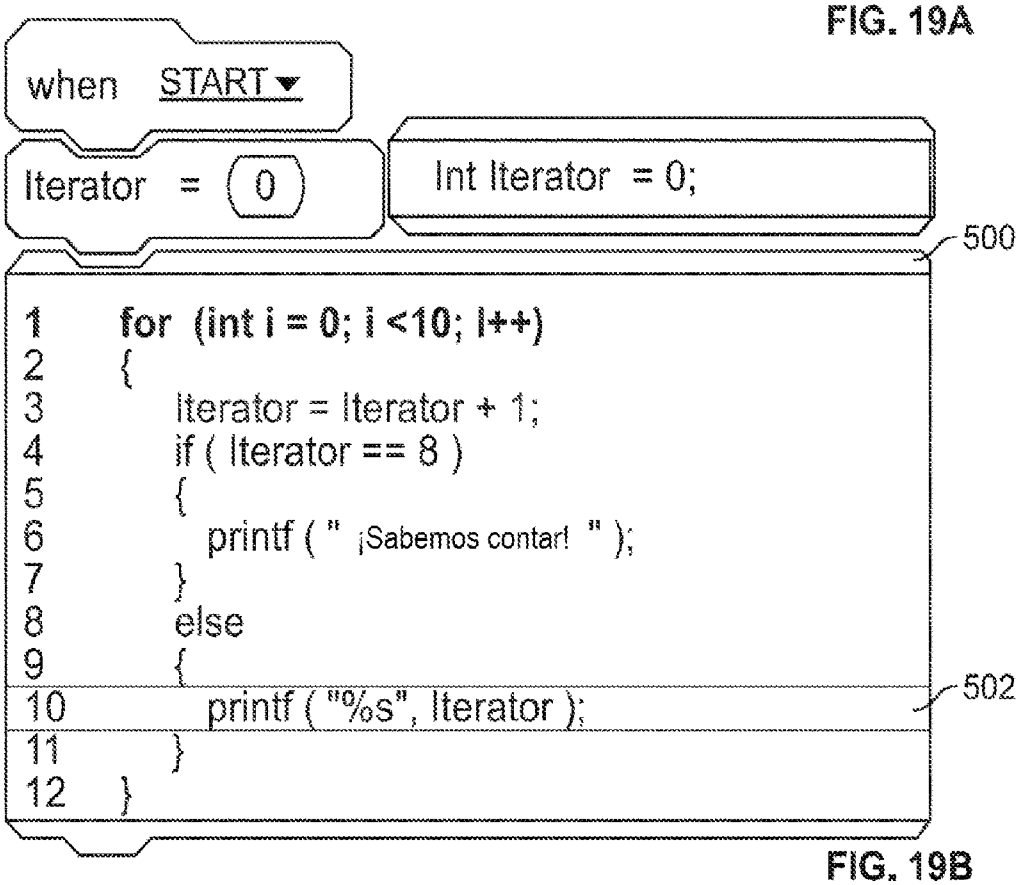

FIG. 19B

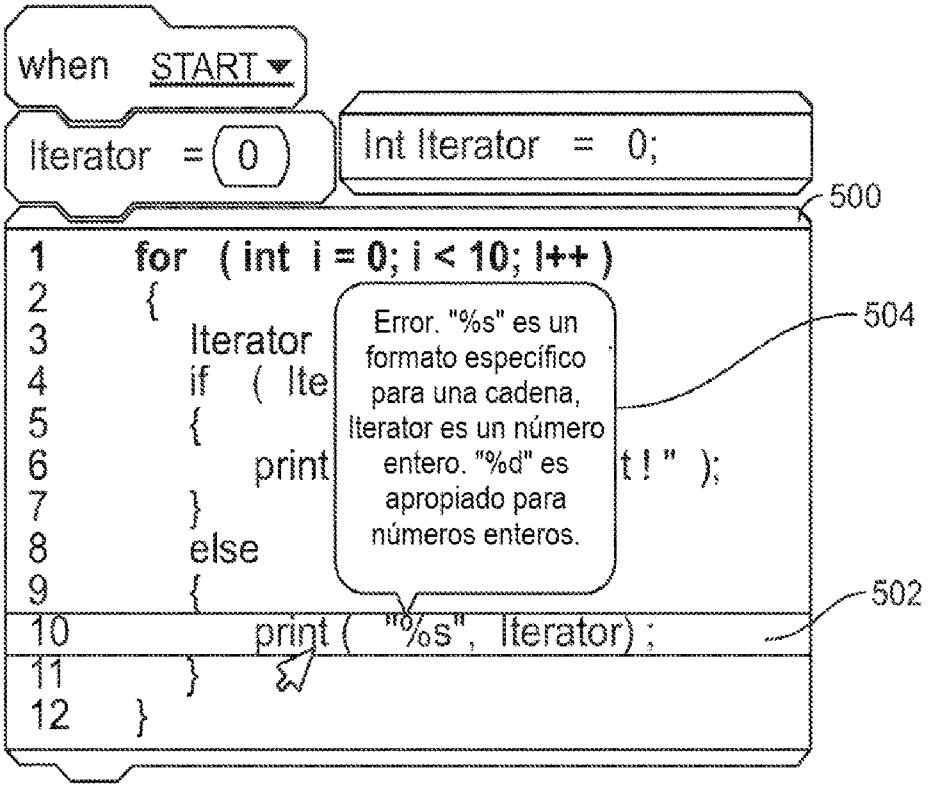

FIG. 19C

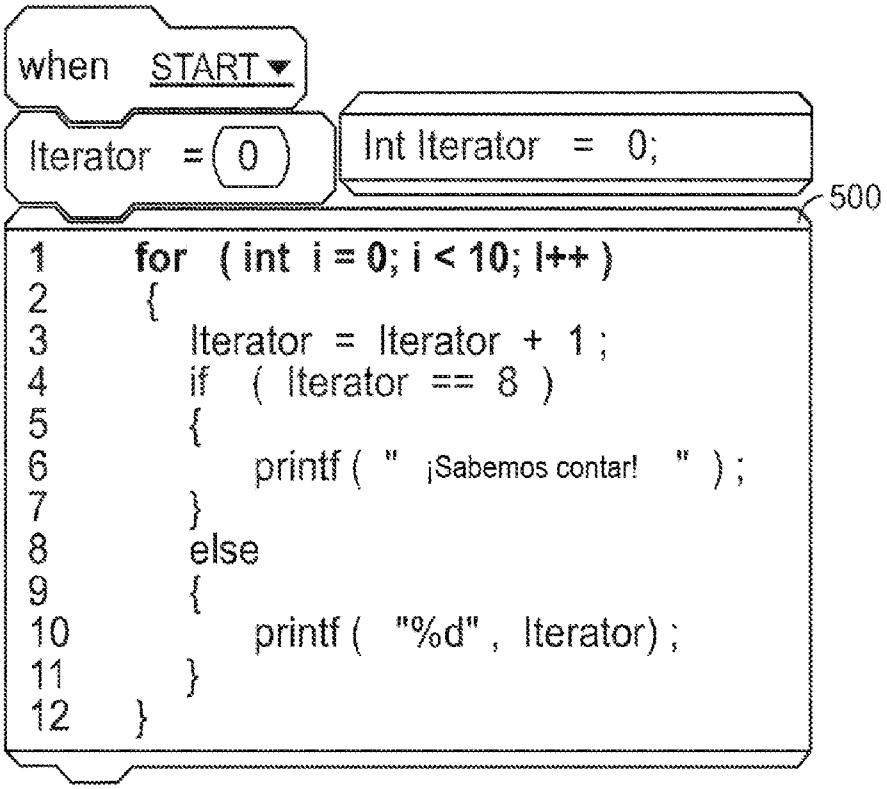

FIG. 19D

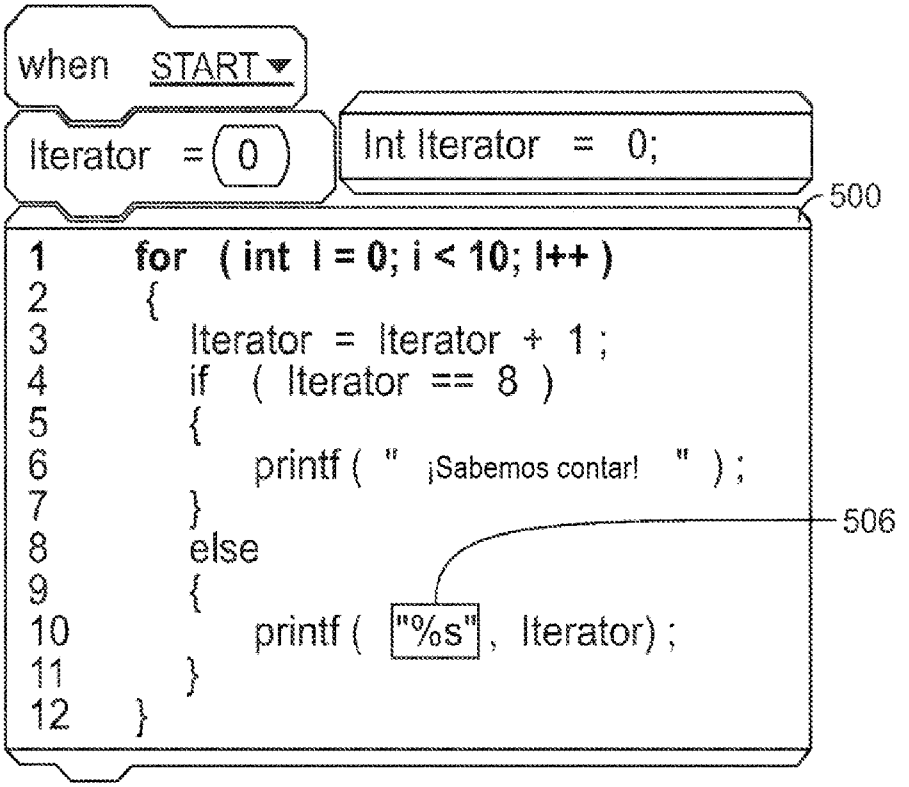

FIG. 19E

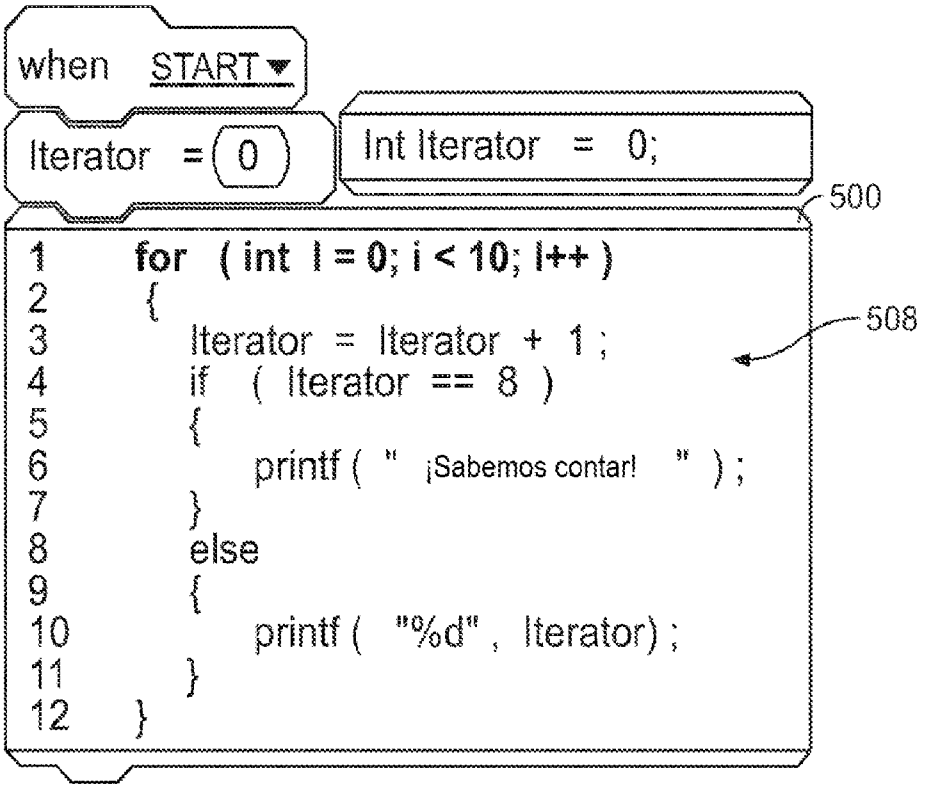

FIG. 19F

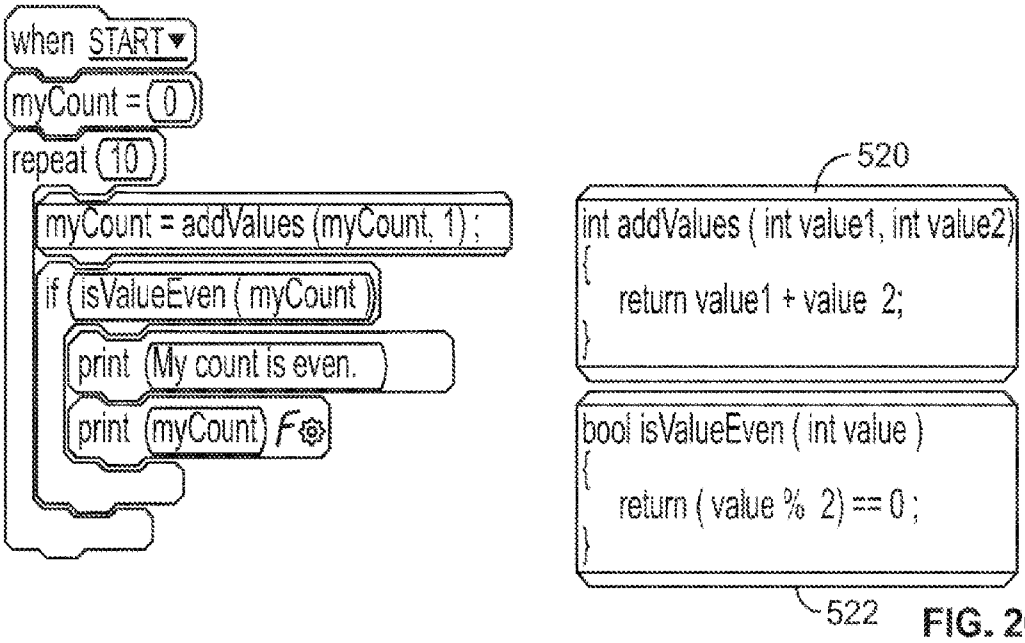

FIG. 20A

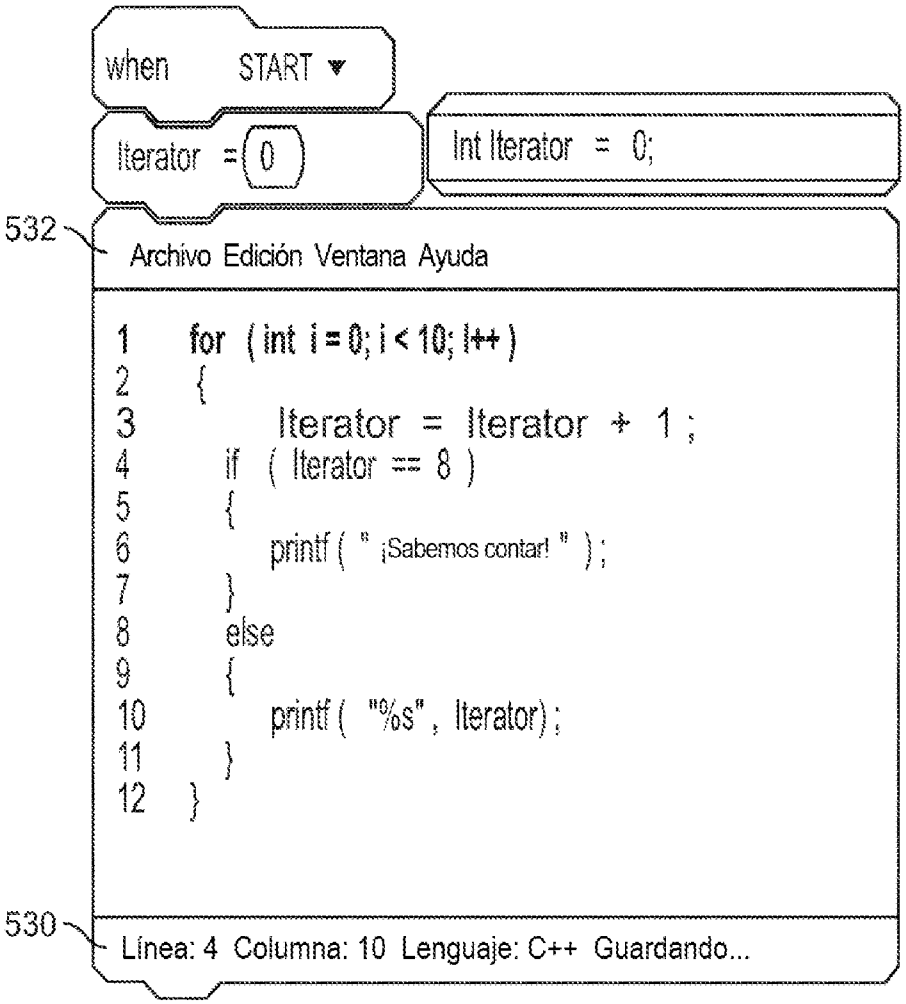

FIG. 20B

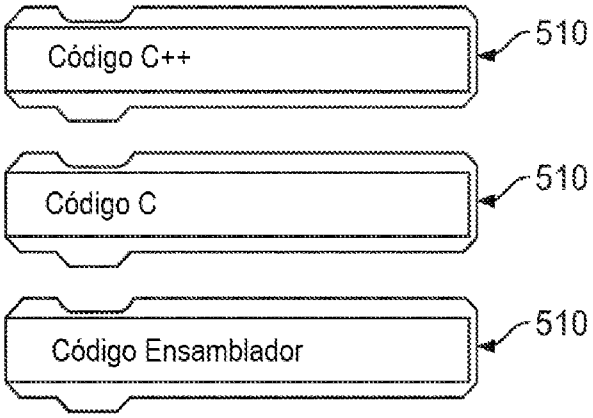

FIG. 21

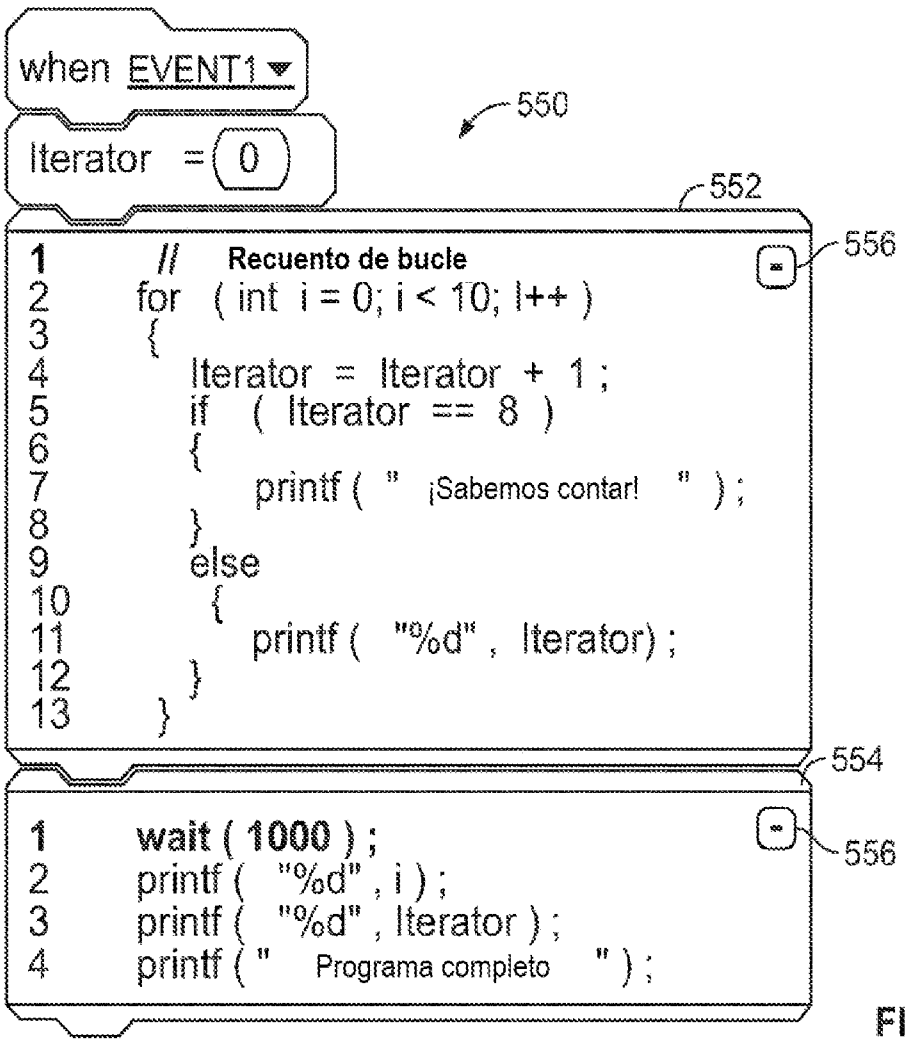

FIG. 22A

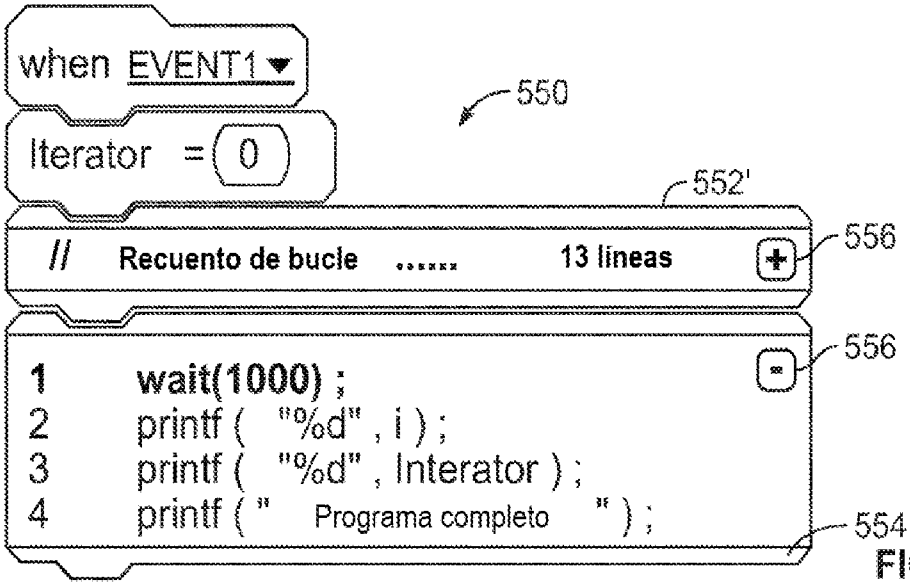

FIG. 22B

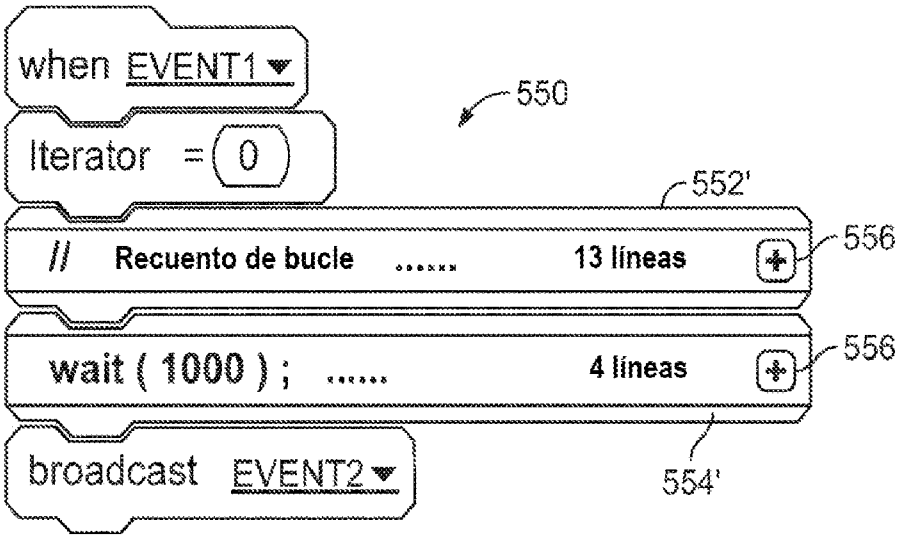

FIG. 22C

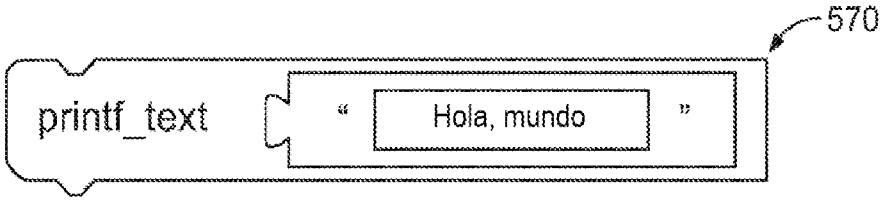

FIG. 23A

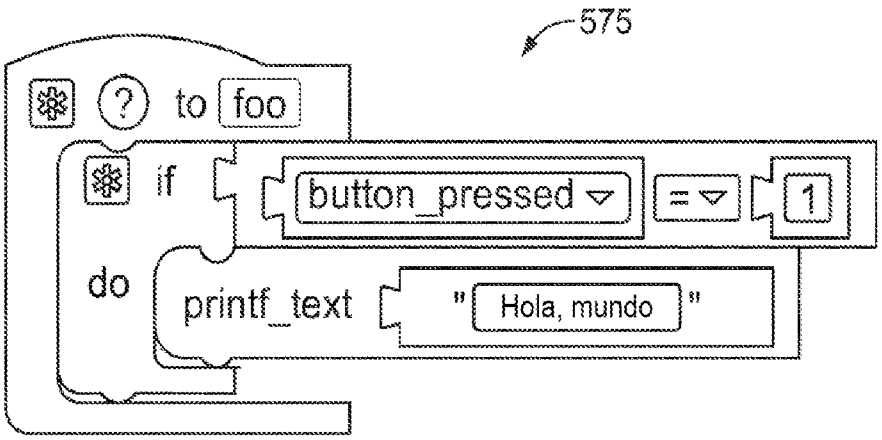

FIG. 23B

```
1    int main (void)
2    int Iterator = 0;
3    for (int i = 0; i < 10; I++)
4    {
5        Iterator = Iterator + 1;
6        if ( Iterator == 8 )
7        {
8            printf( "  ¡Sabemos contar!  " );
9        }
10       else
11       {
12           printf( "%d", Iterator);
13           printf(" Sigue intentándolo ");
14           wait(1000);
15       }
16   }
```

600

FIG. 24A

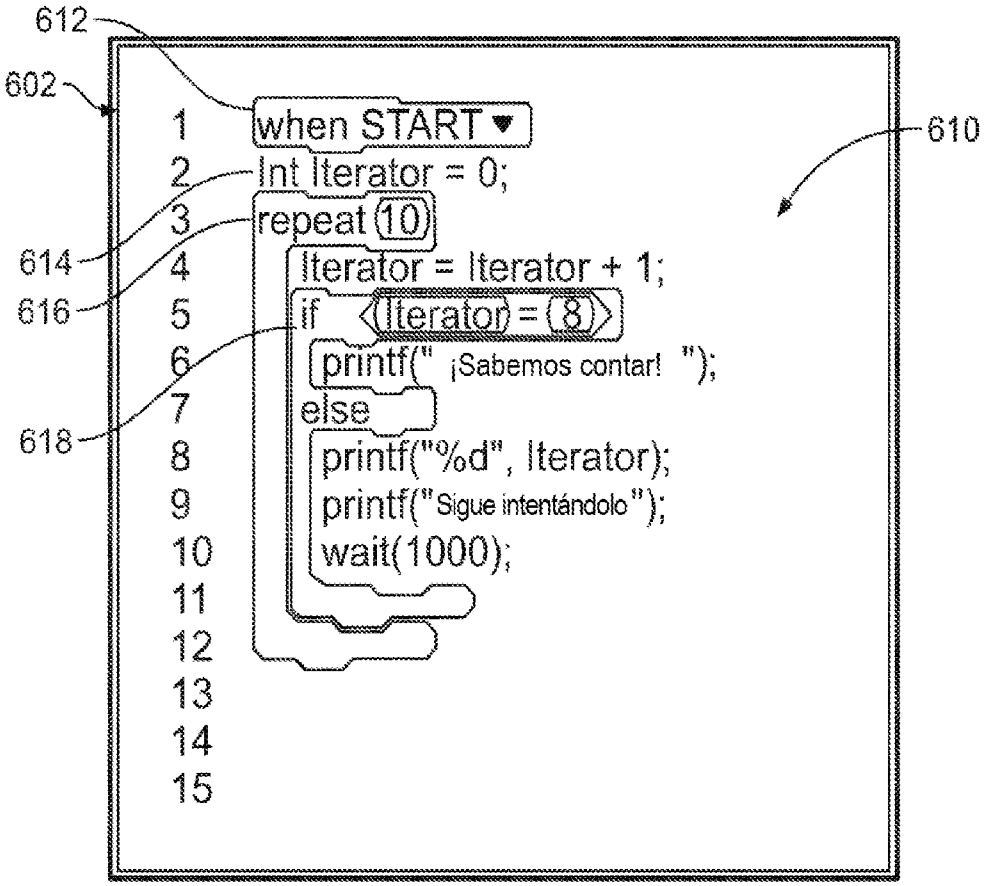

FIG. 24B

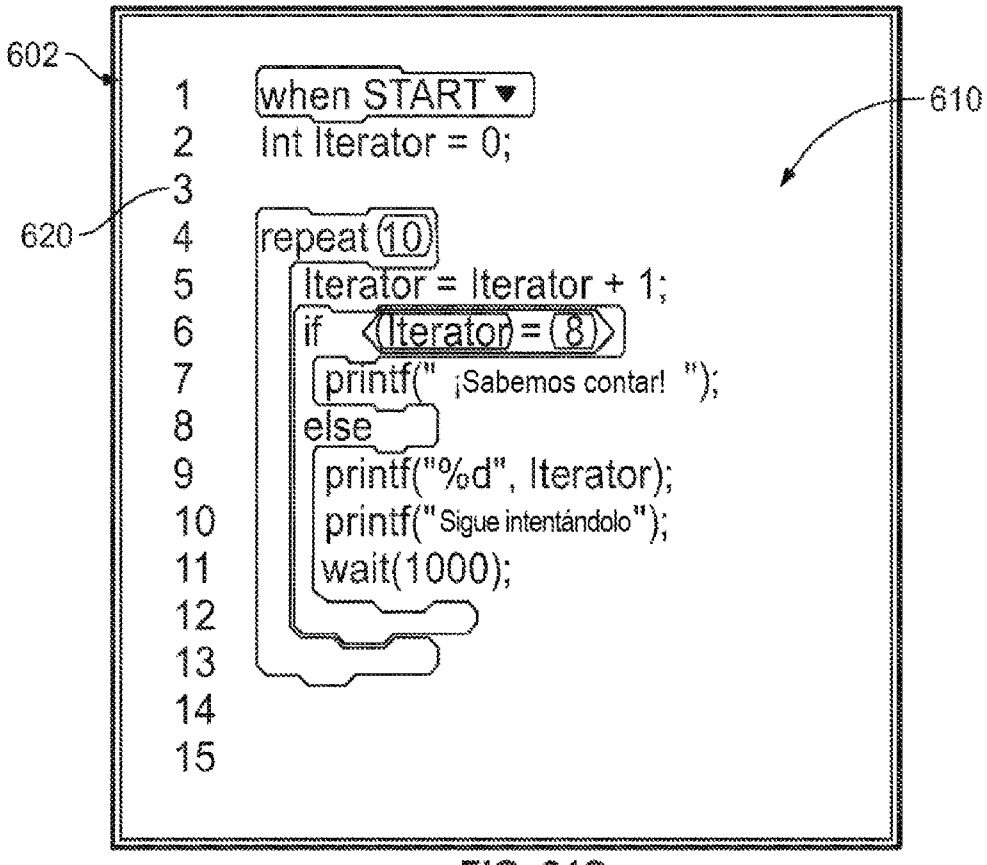
602
620
610
1 when START
2 Int Iterator = 0;
3
4 repeat (10)
5 Iterator = Iterator + 1;
6 if (Iterator) = (8)
7 printf(" ¡Sabemos contar! ");
8 else
9 printf("%d", Iterator);
10 printf("Sigue intentándolo");
11 wait(1000);
12
13
14
15

FIG. 24C

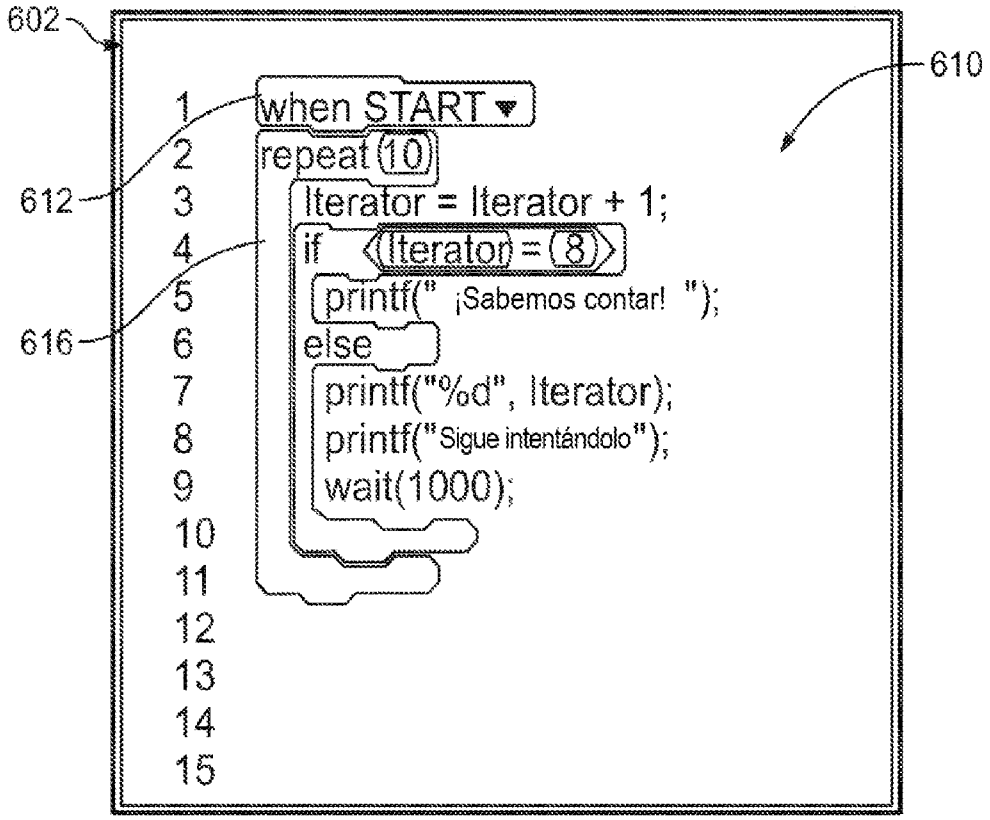

FIG. 24D

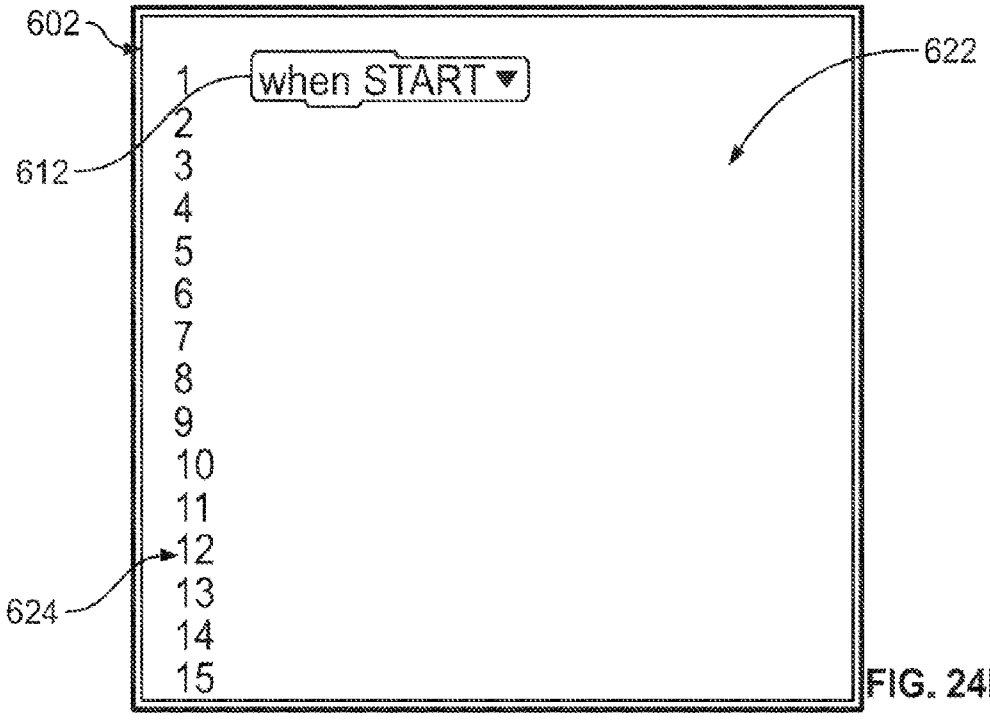

FIG. 24E

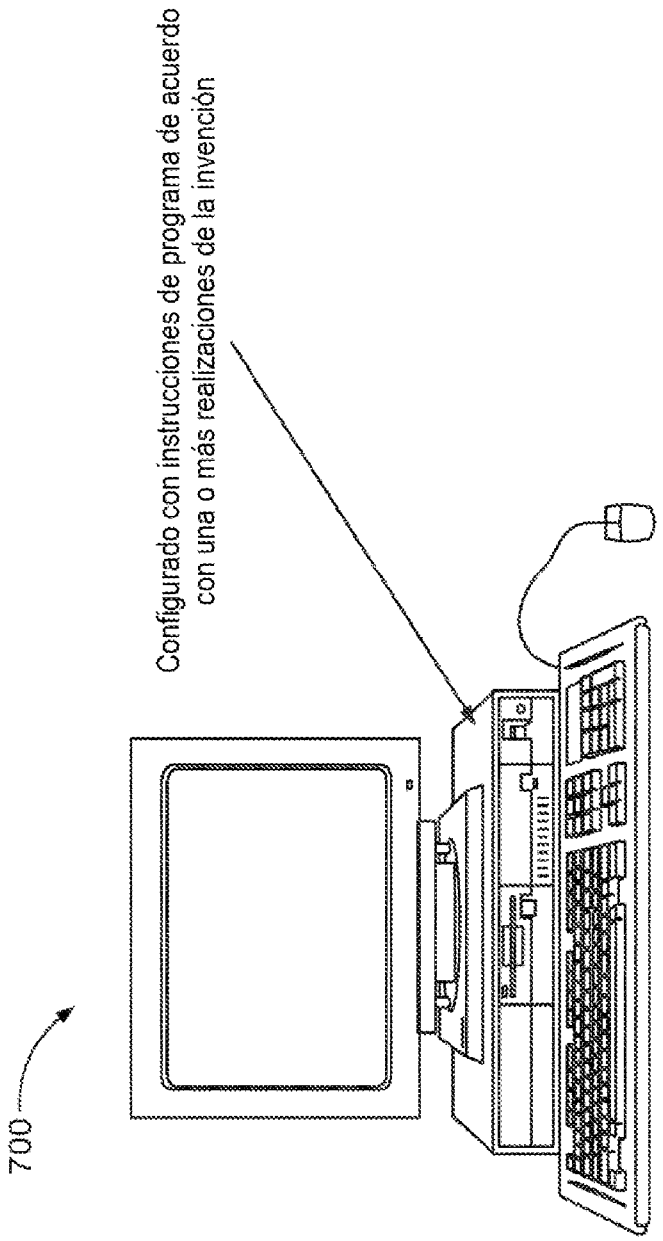
Configurado con instrucciones de programa de acuerdo con una o más realizaciones de la invención
700

FIG. 25

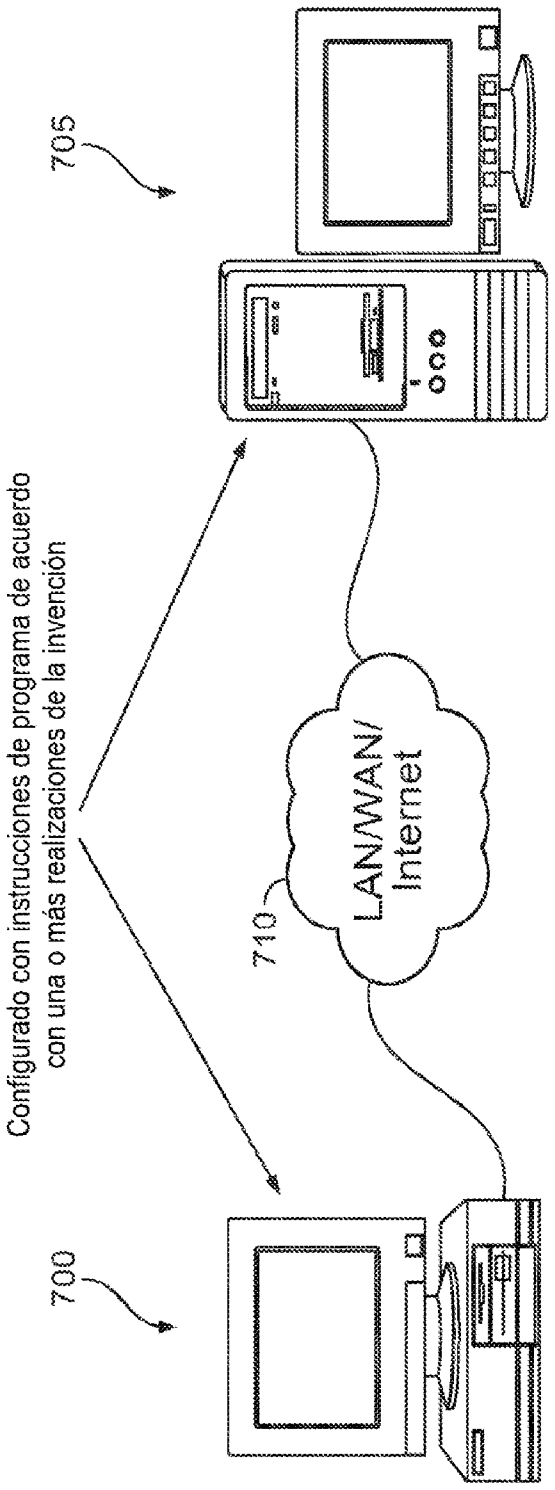
705
Configurado con instrucciones de programa de acuerdo con una o más realizaciones de la invención
LAN/WAN/ Internet
710
700

FIG. 26

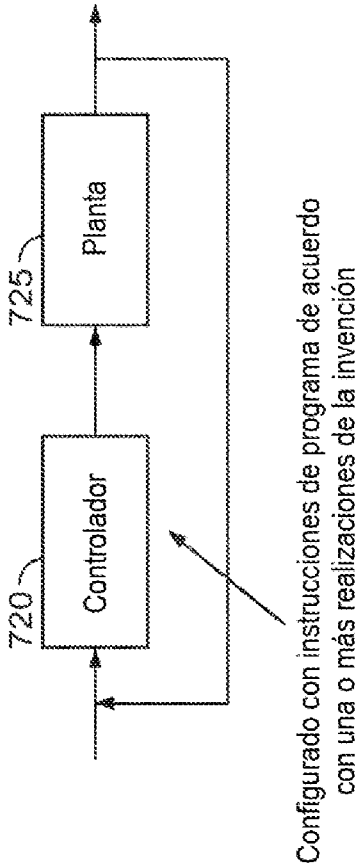
720
Controlador
725
Planta
Configurado con instrucciones de programa de acuerdo con una o más realizaciones de la invención

FIG. 27

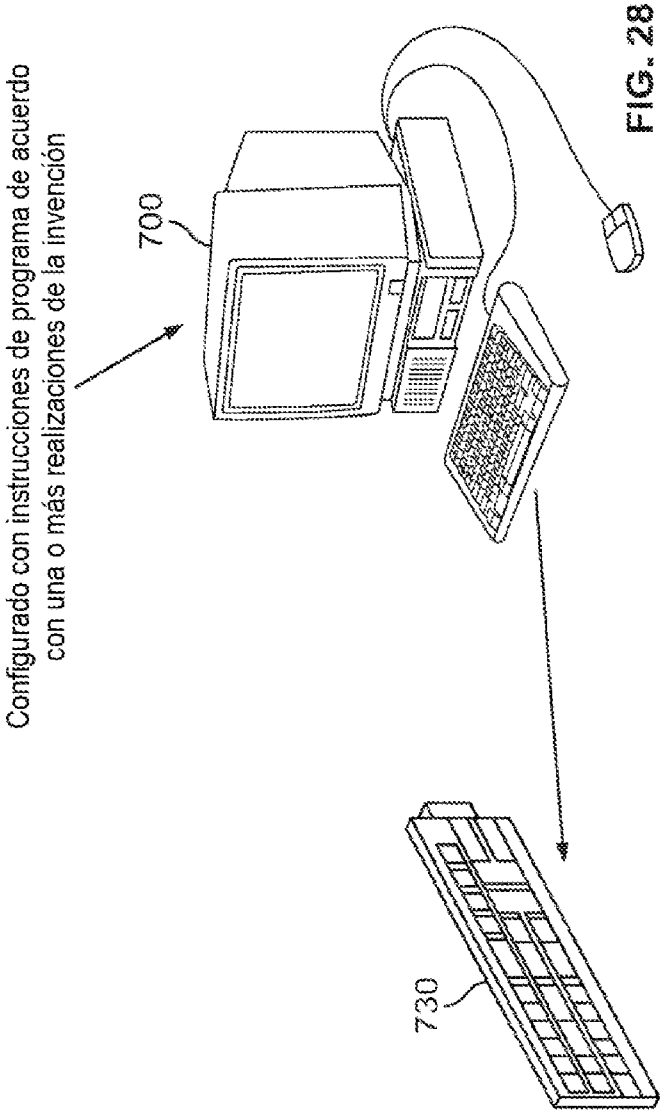
Configurado con instrucciones de programa de acuerdo con una o más realizaciones de la invención
700
730

FIG. 28

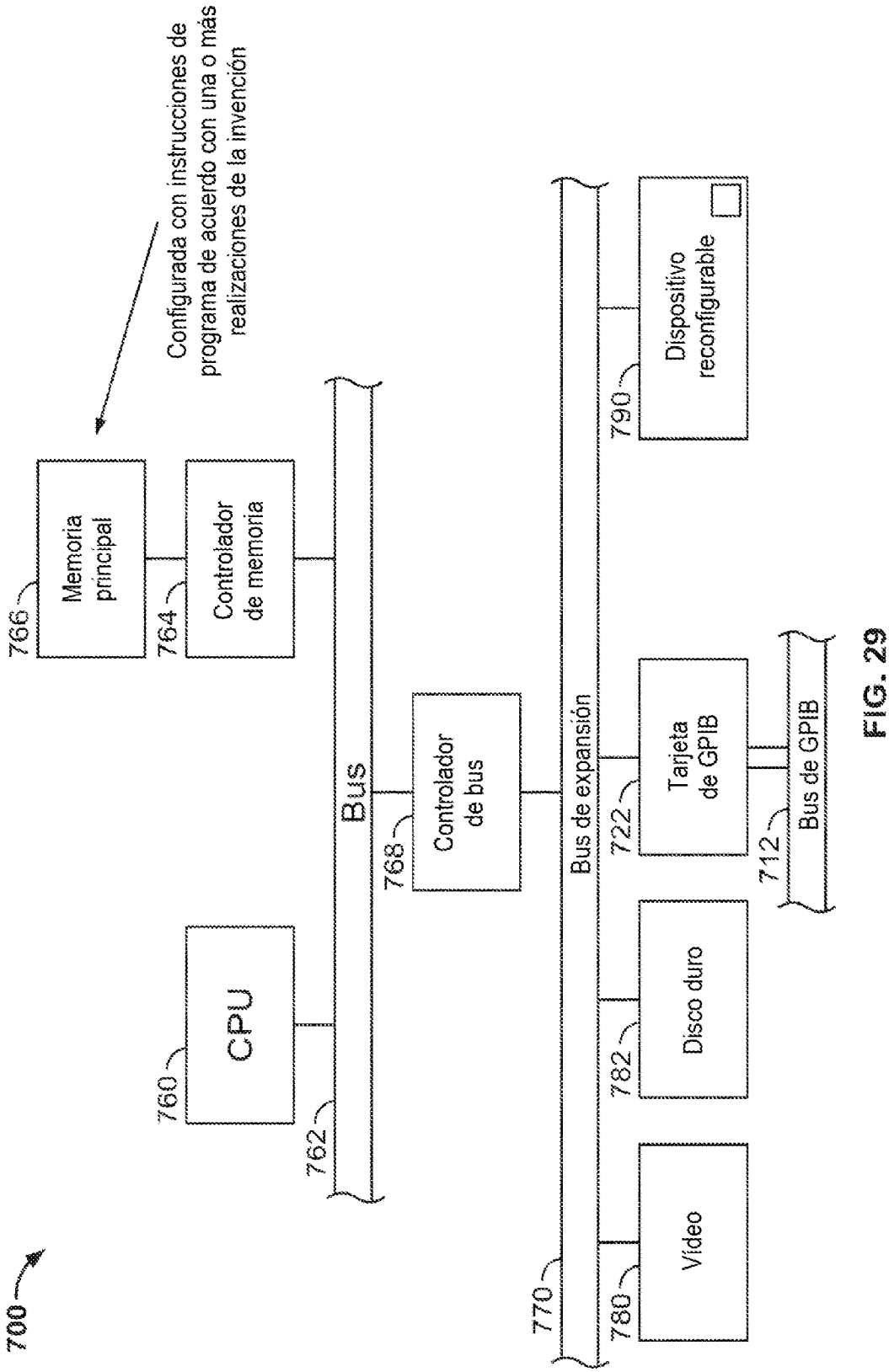
700
Configurada con instrucciones de programa de acuerdo con una o más realizaciones de la invención
766
Memoria principal
764
Controlador de memoria
760
CPU
762
Bus
768
Controlador de bus
770
Bus de expansión
790
Dispositivo reconfigurable
722
Tarjeta de GPIB
712
Bus de GPIB
782
Disco duro
780
Vídeo

FIG. 29