

UNIVERSIDAD AUTÓNOMA DE CIUDAD JUÁREZ

Instituto de Ingeniería y Tecnología

Departamento de Ingeniería Eléctrica y Computación



RUNAROUND: APLICACIÓN ORIENTADA A LA ENSEÑANZA Y APRENDIZAJE
DE LA PROGRAMACIÓN ESTRUCTURADA A TRAVÉS DE ANIMACIONES
GENERADAS A PARTIR DE CÓDIGO FUENTE

Reporte Técnico de Investigación presentado por:

Luis Eduardo Salazar Valles 86406

Requisito para la obtención del título de

INGENIERO EN SISTEMAS COMPUTACIONALES

Profesor Responsable: Ivonne Haydee Robledo Portillo

Mayo del 2013

Autorización de Impresión

Los abajo firmantes, miembros del comité evaluador autorizamos la impresión del proyecto de titulación

**RUNAROUND: APLICACIÓN ORIENTADA A LA ENSEÑANZA Y APRENDIZAJE
DE LA PROGRAMACIÓN ESTRUCTURADA A TRAVÉS DE ANIMACIONES
GENERADAS A PARTIR DE CÓDIGO FUENTE**

Elaborado por el alumno:

Luis Eduardo Salazar Valles 86406

Jorge Enrique Rodas Osollo

Profesor de la Materia

Ivonne Haydee Robledo Portillo

Asesor Técnico

Declaración de Originalidad

Yo Luis Eduardo Salazar Valles declaro que el material contenido en esta publicación fue generado con la revisión de los documentos que se mencionan en la sección de Referencias y que el Programa de Cómputo (Software) desarrollado es original y no ha sido copiado de ninguna otra fuente, ni ha sido usado para obtener otro título o reconocimiento en otra Institución de Educación Superior.

Luis Eduardo Salazar Valles

Dedicatoria

Para mis padres, hermanos y amigos.

Agradecimientos

El presente proyecto no hubiera sido posible sin la paciencia y apoyo constante de mi asesora, la maestra Ivonne Robledo. Gracias.

Le agradezco al profesor Jorge Rodas por su guía y consejos a lo largo de toda la carrera; a mi familia y amigos, por sus consejos y apoyo durante el desarrollo del proyecto; y también, a todos los compañeros que evaluaron la aplicación.

Gracias.

Índice

Autorización de Impresión	ii
Declaración de Originalidad	iii
Dedicatoria.....	iv
Agradecimientos	v
Lista de Figuras	ix
Lista de Tablas	xi
Introducción	1
Capítulo 1. Planteamiento del problema	2
1.1 Antecedentes	2
1.2 Definición del problema	7
1.3 Objetivo de la investigación.....	7
1.4 Preguntas de investigación.....	8
1.5 Justificación de la investigación.....	8
1.6 Limitaciones y delimitaciones de la investigación.....	8
Capítulo 2. Marco teórico	9
2.1 Enseñanza y aprendizaje de la programación estructurada.....	9
2.1.1 Programación estructurada	9
2.1.2 Enfoques en la enseñanza y aprendizaje de la programación.....	13
2.2. Análisis y traducción de código fuente.....	15
2.2.1 Compiladores.....	15
2.2.2 Análisis léxico	17
2.2.3 Análisis sintáctico	19
2.2.4 Análisis semántico y traducción dirigida por la sintaxis.....	21
2.2.5 Lex	23
2.2.6 Yacc	23
2.3 Animación de algoritmos.....	24
2.3.1 Técnicas de animación web 3D	25

Capítulo 3. Materiales y Método.....	28
3.1 Descripción del área de estudio	28
3.2 Materiales.....	29
3.2.1 Hardware	29
3.2.2 Software y servicios	30
3.3 Método	30
3.3.1 Tipo de investigación	30
3.3.2 Evaluación de aplicaciones educativas	31
3.3.3 Evaluación de herramientas de desarrollo.....	34
3.3.4 Desarrollo del analizador léxico	35
3.3.5 Desarrollo del analizador sintáctico.....	37
3.3.6 Desarrollo del intérprete.....	39
3.3.7 Diseño de metáforas y animaciones 3D	41
3.3.8 Integración de escenas e intérprete	49
Capítulo 4. Resultados de la investigación	52
4.1 Presentación de resultados	52
4.2 Análisis e interpretación de los resultados.....	54
Capítulo 5. Discusiones, conclusiones y recomendaciones.....	58
5.1 Discusiones	58
5.2 Conclusiones	59
5.3 Recomendaciones para futuras investigaciones	60
Referencias.....	61
Apéndices	65
Apéndice A: Protocolo	65
Apéndice B: Ejemplos de la codificación.....	68
B.1 Analizador léxico (fragmento en <i>Python</i>):	68
B.2 Analizador sintáctico (fragmento en <i>Python</i>):.....	70
B.3 Intérprete (fragmento en <i>Javascript</i>):.....	72

B.4 Animaciones (fragmento en <i>Javascript</i>):	74
Apéndice C: Encuesta de evaluación	76
Apéndice D: Resultados de las encuestas	79

Lista de Figuras

Figura 1. Secuencia.....	10
Figura 2. Selección simple.....	11
Figura 3. Selección múltiple.....	11
Figura 4. Repetición hacer hasta.....	12
Figura 5. Repetición hacer mientras.....	12
Figura 6. Autómata Finito Determinista.....	19
Figura 7. Árbol sintáctico $7 + (8 * 9)$	20
Figura 8. Árbol sintáctico $(7 + 8) * 9$	20
Figura 9. Interfaz de bloques en Scratch.....	32
Figura 10. Interfaz de Alice.....	32
Figura 11. Interfaz de Jeliot.....	33
Figura 12. Interfaz de Greenfoot.....	33
Figura 13. Definición del analizador léxico.....	36
Figura 14. Uso del analizador léxico.....	36
Figura 15. Definición del analizador sintáctico.....	38
Figura 16. Uso del analizador sintáctico.....	38
Figura 17. Recorrido iterativo del árbol sintáctico.....	40
Figura 18. Codificación JSON de los nodos.....	40
Figura 19. Contenedor de variable abierto.....	41
Figura 20. Contenedor de variable cerrado.....	41
Figura 21. Tipos de datos.....	41
Figura 22. La plataforma y sus elementos.....	42
Figura 23. Actor Runaround.....	43
Figura 24. Área de expresiones.....	43
Figura 25. Generación de plataformas por cada llamada a función.....	44
Figura 26. Editor de código fuente.....	45
Figura 27. Representación y recorrido del árbol de la sintaxis.....	46
Figura 28. Animación de correr.....	47
Figura 29. Animación de sujetar.....	47
Figura 30. Escena declaración de variable.....	49
Figura 31. Interpolación elástica utilizada para el comportamiento de un contenedor.....	50
Figura 32. Encuesta realizada al grupo B de programación II.....	53
Figura 33. Porcentaje de edades.....	54
Figura 34. Porcentaje de semestres.....	54
Figura 35. Total de respuestas: parte B pregunta 4.....	55

Figura 36. Total de respuestas: parte C pregunta 6.....	55
Figura 37. Total de respuestas: parte C pregunta 4.....	56

Lista de Tablas

Tabla 1. Hardware utilizado para el desarrollo del proyecto.	29
Tabla 2. Software y servicios utilizados para el desarrollo del proyecto.....	30

Introducción

La programación de computadoras es una habilidad que permite a los profesionistas de todas las áreas resolver problemas cada vez más complejos; el interés por estudiar programación ha crecido rápidamente en los últimos años.

La complejidad de los programas que se desarrollan actualmente requiere el uso de técnicas de programación efectivas, por lo que su enseñanza es de suma importancia; sin embargo, algunos cursos de programación para estudiantes con poca o ninguna experiencia enfrentan determinadas dificultades, ya que algunos estudiantes tienen problemas con la sintaxis y los conceptos abstractos como apuntadores o memoria. Estos problemas han influido en el desarrollo de diferentes aplicaciones cuyo objetivo es apoyar al aprendizaje y la enseñanza de temas específicos en la programación.

El presente proyecto aborda el desarrollo de una aplicación que apoye a los estudiantes en el aprendizaje y a los profesores en la enseñanza de los conceptos fundamentales de la programación estructurada a través de animaciones generadas a partir de código fuente.

En el primer capítulo se revisan los antecedentes, como las aplicaciones educativas ya existentes, y se definen el objetivo del proyecto, así como sus limitaciones y delimitaciones. Después, en el segundo capítulo se indagan las bases teóricas de la programación estructurada, los fundamentos necesarios para el desarrollo de los compiladores y las tecnologías disponibles para realizar animaciones 3D en un entorno web.

Posteriormente, en el tercer capítulo se describen los pasos realizados para el desarrollo de la aplicación, incluyendo el desarrollo del analizador léxico, del analizador sintáctico, el desarrollo del intérprete, y el diseño de las metáforas y las animaciones 3D. En el cuarto capítulo, la aplicación es evaluada por los estudiantes y profesores de la UACJ revelando resultados favorables, así como observaciones y recomendaciones.

Finalmente, en el quinto capítulo se exponen las conclusiones, indicando que los resultados son muy positivos e ilustrando que es viable el desarrollo de este tipo de aplicaciones. Se detallan las limitaciones encontradas, y para terminar, las recomendaciones para futuras investigaciones, como el buscar nuevas metáforas y enfoques, así como también el realizar investigaciones posteriores sobre el impacto en el aprendizaje y la enseñanza de la programación que este tipo de aplicaciones son capaces de ofrecer.

Capítulo 1. Planteamiento del problema

Este capítulo inicia con los antecedentes del proyecto llevando a cabo una revisión de la literatura de los problemas asociados con la enseñanza y aprendizaje de la programación, así como también de las aplicaciones educativas más relevantes, sus enfoques, objetivos y resultados. Después, en la definición del problema se identifican las dificultades tanto como para estudiantes como para profesores y, posteriormente, se establece el objetivo de desarrollar una aplicación que genere animaciones a partir del código fuente. Las preguntas de investigación discuten qué conceptos serán representados, cuáles metáforas se desarrollarán y qué tecnologías serán utilizadas. Finalmente, se establecen las limitaciones previstas que pueden afectar al desarrollo de la investigación y las delimitaciones en cuanto a lugar y temática abordados.

1.1 Antecedentes

La programación se relaciona con diversos campos de la tecnología, y el ser capaz de programar es una habilidad muy útil. En los últimos años la demanda de programadores y el interés por estudiar programación han crecido rápidamente [1]. Los cursos de programación son cada vez más populares y gran cantidad de estudiantes universitarios se encuentran estudiando los cursos básicos, sin embargo, sufren una variedad de deficiencias y dificultades.

Los cursos de programación generalmente son considerados difíciles, ya que requieren la correcta comprensión de conceptos abstractos y, muchos estudiantes sufren problemas de aprendizaje dada la naturaleza del tema. A menudo no hay suficientes recursos y los estudiantes sufren de una falta de atención personal [1]. Los grupos son grandes y heterogéneos, por lo que es complicado diseñar un curso efectivo para todos, lo cual provoca altos índices de abandono en los cursos básicos.

Una encuesta internacional a estudiantes y profesores [2] identificó como temas especialmente problemáticos los conceptos abstractos como apuntadores y memoria; pero aún más problemático es el aprender a aplicar estos conceptos y cómo combinarlos en programas más complejos. Como se describe en [1], los estudiantes principiantes utilizan el enfoque de programar “línea por línea” en lugar de utilizar “piezas” de estructuras

significativas de un programa; pueden tener una comprensión muy pobre de la naturaleza básica de la ejecución secuencial de un programa y tienden a olvidar que cada instrucción opera en el entorno creado por las instrucciones previas.

Estas dificultades han influido en el desarrollo de diferentes aplicaciones educativas. En la taxonomía descrita en [3] se identifican siete categorías principales: micro mundos de programación, entornos de programación educativos, herramientas para mejorar las capacidades de los entornos de programación, herramientas de software apoyando la enseñanza de “primero objetos”, entornos de videojuegos y herramientas específicas de un lenguaje de programación educativo. Una combinación apropiada de herramientas educativas y profesionales junto a una suave transición de las primeras a las segundas sería una situación ideal.

A continuación se presentan cuatro de las aplicaciones más populares según [4]: *Scratch* es una aplicación que permite crear programas al ensamblar bloques de programación visual, como bloques *Lego*, para controlar imágenes, música y sonido. *Alice* es otra aplicación que permite crear películas y videojuegos 3D al ensamblar cajas de instrucciones que corresponden más a las de la programación orientada a objetos. *Greenfoot* es un entorno de desarrollo *Java* para crear animaciones y juegos 2D editando clases de escenarios y actores, sin embargo el estudiante requiere conocimiento previo de programación. *App Inventor* es un lenguaje visual de bloques, muy similar a *Scratch*, que permite crear aplicaciones móviles para *Android*. Incluye bloques de programación y funcionalidad estándar, cómo manejo de sensores del teléfono, redes y bases de datos.

Se analizó el uso de *Scratch* en la *ORT Uruguay University* [5] durante las primeras semanas de un curso de introducción a la programación. La aplicación fue utilizada para estimular a los estudiantes en esta etapa crítica y familiarizarlos con los conceptos fundamentales de la programación, sin la distracción de la sintaxis. El uso de la *Scratch* promovió un alto nivel de motivación y se percibió una mejoría en el aprendizaje; sin embargo, no se encontraron diferencias estadísticas significativas en los resultados, así como tampoco en el nivel de retención. El instalar *Scratch* y familiarizarse con el entorno toma tiempo y, unas semanas después, el estudiante debe cambiar a un entorno de desarrollo tradicional por lo que las mejorías iniciales pueden verse afectadas por dichos cambios.

En 2008 se realizó un experimento con 166 estudiantes de preparatoria [6], se enseñó *Alice* a 81 estudiantes y al resto C++. Como una aplicación de “primeros objetos”, *Alice* brinda a los estudiantes un enfoque intuitivo mediante la creación de mundos 3D. Los estudiantes expresaron una respuesta positiva al curso de *Alice*, se desempeñaron significativamente mejor que los otros grupos e incrementaron su confianza al programar, comprendiendo los conceptos básicos de la programación así como la relación entre los algoritmos y las animaciones.

Greenfoot, descrita en [7], parece ser la aplicación tradicional de micro-mundo, sin embargo, es mucho más flexible, ya que separa la funcionalidad del escenario de la interfaz principal, permitiendo el desarrollo de una gran variedad de temas. A diferencia de *Alice* y *Scratch*, que son más amigables para los principiantes, *Greenfoot* requiere una mayor competencia para su operación, ya que el uso de *Java* trae consigo un mínimo de complejidad al ser un lenguaje orientado a la sintaxis; sin embargo, *Greenfoot* permite crear proyectos más ambiciosos, al contar con el ecosistema y eficiencia de una plataforma robusta como *Java* pero, al mismo tiempo, conserva una simplicidad en su uso. Al utilizar código estándar de *Java*, la aplicación *Greenfoot* facilita la transición a entornos de desarrollo profesionales en el futuro.

La aplicación *App Inventor* puede ser utilizada para una introducción amigable a los conceptos de programación necesarios para otras áreas, según [8]. Numerosos campos como Química, Aero acústica, Bioinformática, Finanzas, entre otros, utilizan actualmente la programación, muchos informalmente como parte de sus clases normales. No se puede pensar en alguna faceta humana que no se vea transformada por las aplicaciones móviles y, *App Inventor*, permite que cualquiera pueda desarrollarlas gracias a su programación por bloques y poderosas funcionalidades móviles, lo que la convierte en una herramienta capacitadora y facilitadora para otras ramas.

Los métodos de enseñanza tradicionales por lo regular son muy pasivos y no atraen el interés del estudiante, ya que no se relacionan con nada familiar para él. Un curso web desarrollado en las escuelas del Caribe [9] espera ayudar a los estudiantes a reducir la brecha entre las situaciones del mundo real y los conceptos de la programación. Utilizando visualizaciones, los conceptos son enseñados a través de juegos y metáforas, por lo que el aprendizaje se lleva a cabo en el contexto del mundo real. La respuesta fue positiva, los

estudiantes se identificaron con el contenido visual y se lograron reducir los problemas durante el aprendizaje de los conceptos abstractos.

El uso de metáforas por parte de los profesores para enseñar a los estudiantes los conceptos complejos es muy común. En [10] se describen tres objetos de aprendizaje para la enseñanza de la programación a través de metáforas animadas e interactivas. Las metáforas para el concepto de variable, estructura condicional y estructura de repetición consisten en cilindros rotatorios con dígitos que giran y sistemas de tubos con rutas alternas para una pelota que va cayendo. Un experimento piloto [10] con estos objetos de aprendizaje reveló que los estudiantes ya estaban familiarizados con las metáforas seleccionadas y estas les ayudaron a identificar características importantes de los conceptos que en cursos tradicionales no resultan obvios.

La aplicación *Hypermedia Visualization System (HalVis)*, descrita en [11], utiliza metáforas interactivas y animadas para la enseñanza de la programación. *HalVis* se compone de tres vistas: la vista conceptual presenta el programa en términos muy generales, utilizando una metáfora del mundo real; la vista detallada muestra los pasos individuales, incluyendo la modificación de los datos; y la vista poblada, el comportamiento del programa con conjuntos grandes de datos. Se realizaron múltiples experimentos con 133 estudiantes comparando el impacto en el aprendizaje de cada vista [11]. La vista detallada fue la más efectiva, seguida de la vista conceptual y por último la vista poblada. El mejor desempeño resultó al combinar las 3 vistas. El poder de las metáforas reside en el hecho de que estas proveen puentes conceptuales que unen a los escenarios familiares para los estudiantes, con los componentes abstractos de los algoritmos.

Uno de los problemas más graves en la enseñanza de la programación es la excesiva cantidad de tiempo requerido para conocer la sintaxis de algún lenguaje de programación, lo cual deja poco tiempo para desarrollar otras habilidades más importantes. Una aplicación para la enseñanza de la programación a través de plantillas [12] fue desarrollada para ayudar a reducir los problemas de sintaxis. El estudiante puede agregar una sentencia nueva con un botón y, a continuación, se le presenta una plantilla a rellenar; este enfoque evita el teclear la sentencia completa y hace imposible insertar sentencias no válidas en el programa principal. Un experimento realizado con 20 estudiantes durante 6 semanas [12] muestra que el uso de esta aplicación mejoró significativamente su desempeño.

La naturaleza textual de la mayoría de los entornos de desarrollo va en contra del estilo de aprendizaje de la mayoría de los estudiantes. La aplicación *Raptor*, descrita en [13], está diseñada específicamente para ayudar a los estudiantes a evitar los problemas de la sintaxis. Los programas son creados visualmente al combinar símbolos gráficos básicos y pueden ser ejecutados ya sea paso a paso o en un modo continuo. Durante la ejecución se muestra el contenido de las variables y el símbolo que está siendo evaluado actualmente. Enseñar programación visualmente ayudó a desarrollar más las habilidades para solucionar problemas que utilizando sólo un lenguaje tradicional.

La visualización de software es una opción más para apoyar el aprendizaje de los estudiantes, esta consiste en el uso de gráficos por computadora para facilitar el entendimiento de los algoritmos. La aplicación *Jeliot*, descrita en [14], muestra a los estudiantes representaciones gráficas de los aspectos de un algoritmo que generalmente son intangibles y ocultos, como el flujo de control y las dependencias de datos. El código *Java* a ser animado no requiere ningún tipo de modificación. La efectividad de *Jeliot* fue evaluada en un estudio con 45 estudiantes [14]. Los estudiantes que utilizaron *Jeliot* construyeron modelos mentales más detallados y concretos del algoritmo, sin embargo, los requisitos de los estudiantes cambian rápidamente y para otros más avanzados las animaciones pueden significar una pérdida de tiempo que los distraen del resultado.

Una vez que el estudiante puede crear programas de una complejidad mayor, otra dificultad es el aprender a descomponer los problemas apropiadamente y comprender como las partes llevan a un todo. Tal como se describe en [15], la aplicación *AnnAnn.Net* es un anotador de código animado que a través de las anotaciones explica el objetivo de los cambios realizados en el código y resalta las líneas modificadas. El estudiante puede navegar a través de los pequeños cambios realizados en el código, junto a sus anotaciones, para comprender la evolución paso a paso de un sistema simple a uno de mayor complejidad. El uso de la aplicación ayuda al profesor a hacer explícito su conocimiento tácito que utiliza rutinariamente al construir los programas.

La revisión de los trabajos anteriores revela la existencia de múltiples retos en la enseñanza y aprendizaje de la programación. Entre los problemas principales que se identificaron fueron la dificultad al comprender los conceptos fundamentales debido a su naturaleza abstracta. Otro problema radica en distraerse con la sintaxis y el entorno de

desarrollo, lo cual evita el desarrollo de las habilidades más importantes para solucionar problemas. Las metáforas son útiles para que los conceptos de la programación se asimilen de una forma más concreta, ya que los estudiantes pueden relacionarlos con situaciones de la vida real. Para evitar enfocarse demasiado en la sintaxis, se desarrollaron entornos de desarrollo visuales que facilitan la creación de programas al ensamblar piezas estilo rompecabezas; los entornos son simplificados reduciendo el número de opciones y además proporcionan funcionalidades llamativas listas para utilizar.

El enfoque principal de estas aplicaciones educativas es el de permitir que el estudiante obtenga resultados visuales rápidamente, ya sea en animaciones 2D o 3D, y de esta forma no es necesario que comience desde cero; esto motiva al estudiante y lo ayuda a enfocarse en como relacionar los conceptos del mundo real con los conceptos de la programación, en lugar de sólo dedicarse a aprender a utilizar una herramienta de desarrollo tradicional. Estas aplicaciones ayudan en la etapa inicial del aprendizaje y enseñanza de la programación motivando al estudiante a aprender más por sí mismo; sin embargo es necesaria una transición a herramientas cada vez más potentes hasta llegar a los entornos de desarrollo profesionales, que es uno de sus objetivos principales.

1.2 Definición del problema

La enseñanza de la programación estructurada a estudiantes con poca o ninguna experiencia involucra diversos retos educativos para los profesores. Por otra parte, los estudiantes requieren recordar y asimilar una variedad de ideas nuevas, así como manipular símbolos y reglas sin una analogía del mundo real evidente, lo cual dificulta al mismo tiempo el aprendizaje.

1.3 Objetivo de la investigación

Desarrollar una aplicación que apoye a los profesores en la enseñanza y a los estudiantes en el aprendizaje de los conceptos fundamentales de la programación estructurada a través de animaciones generadas a partir del código fuente.

1.4 Preguntas de investigación

¿Qué analogías y diseño de animaciones apoyarán a la enseñanza y aprendizaje de los conceptos fundamentales de la programación estructurada?

¿Cuáles conceptos fundamentales de la programación estructurada serán representados?

¿Qué tecnologías son adecuadas para el desarrollo de la aplicación?

1.5 Justificación de la investigación

La aplicación apoyará a la enseñanza y aprendizaje de los conceptos fundamentales de la programación estructurada, ya que:

- Los conceptos serán representados del código abstracto a las animaciones concretas.
- Será de fácil acceso para profesores y alumnos, debido a que podrá ejecutarse en línea a través de un navegador.
- Propiciará un ambiente para que el estudiante practique y aprenda por sí mismo.

1.6 Limitaciones y delimitaciones de la investigación

Entre las posibles limitaciones se incluye el desconocimiento de las técnicas de procesamiento del código fuente, lo cual involucra indagar las fases de análisis que realiza un compilador para relacionar las sentencias escritas en algún lenguaje de programación con la generación de gráficas 3D. El generar las animaciones correspondientes conlleva también sus dificultades, ya que si el código original cuenta con cierto nivel de complejidad, como anidación de diferentes estructuras de control, se complicará la representación gráfica dependiendo de las metáforas seleccionadas.

En cuanto a las delimitaciones del proyecto, este será desarrollado en la UACJ, en el IIT, para su uso por los estudiantes de las materias fundamentos de programación. La temática de la aplicación será la programación estructurada, centrándose en las estructuras de control básicas: secuencia, selección simple y ciclo mientras, así como también en la asignación y declaración de variables. La aplicación será accesible fácilmente a través de un navegador y utilizará la sintaxis del lenguaje C, por ser un lenguaje estructurado muy utilizado y además porque los estudiantes lo utilizarán en los cursos posteriores.

Capítulo 2. Marco teórico

En el siguiente capítulo se definen los conceptos y teorías fundamentales para el desarrollo de la aplicación. En la primera sección se definen las técnicas de enseñanza y aprendizaje de la programación estructurada incluyendo las estructuras de control; después, se lleva a cabo una revisión de los diferentes enfoques para la enseñanza y aprendizaje de la programación. En la segunda sección se introduce el concepto de compilador, sus fases de análisis léxico, sintáctico y semántico, así como una breve descripción de la técnica de la traducción dirigida por la sintaxis. En la última sección se revisa la animación de algoritmos, sus características, las técnicas utilizadas en proyectos similares de visualización y, finalmente, se describen las tecnologías existentes para la visualización de gráficas 3D a través del navegador, sus características, y el panorama a futuro.

2.1 Enseñanza y aprendizaje de la programación estructurada

La programación estructurada es una técnica que facilita la lectura y modificación de los programas al utilizar las estructuras de control: secuencia, selección e iteración. Es necesario que los estudiantes aprendan esta y otras de las técnicas más relevantes de la programación correctamente, por lo que es necesario el uso de diferentes enfoques para la enseñanza y aprendizaje de la programación que puedan aplicarse a la técnica de la programación estructurada.

2.1.1 Programación estructurada

Una de las características principales de un buen programa es que pueda ser leído y comprendido por cualquier otro programador. Si se mantiene una buena estructura, el programa será más fácil de comprender, modificar y depurar cuando sea necesario. Según [16], un programa no estructurado es aquel en el que no se ha hecho ningún esfuerzo, o muy poco, para ayudar a los otros programadores a leerlo y comprenderlo fácilmente. La estructura de un programa no supone ninguna diferencia para la computadora, solamente para las personas que trabajan con éste.

Como se explica en [17], la programación estructurada consiste en la escritura de programas que cumpla con las siguientes reglas: el programa deberá tener un diseño modular, los módulos serán diseñados descendentemente y cada módulo del programa se codificará usando las estructuras de control. Las estructuras de control se emplean para especificar el orden de ejecución de las instrucciones en un algoritmo; este orden determina el flujo del control de un programa. La programación estructurada ayuda a producir programas más fáciles de escribir, verificar, leer y mantener, ya que utilizando un número limitado de estructuras de control se logra minimizar la complejidad de los problemas.

El teorema del programa estructurado de Böhm y Jacopini [17] dice que todo programa propio puede ser escrito utilizando solamente tres tipos de estructuras de control básicas: secuencia, selección y repetición. Un programa se dice que es propio si tiene un solo punto de entrada y otro único punto de salida, si existen caminos desde la entrada hasta la salida que pasan por todas las partes del programa, todas las instrucciones son ejecutables y si no existen bucles infinitos.

La secuencia, descrita en [18], es una estructura de control que permite a la computadora ejecutar una acción, después otra, luego la que sigue y así sucesivamente hasta la última (Figura 1).

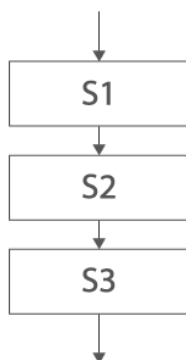


Figura 1. Secuencia.

Las acciones de una secuencia pueden consistir de operaciones elementales como la declaración de variables, lectura e impresión de datos o el cálculo de alguna expresión. En ocasiones se tienen operaciones que son excluyentes, es decir, que sólo tiene que ejecutarse una o la otra, pero no ambas de manera simultánea; también puede presentarse el caso en el

que se tenga una variedad de opciones de acción. En estos casos es necesario utilizar la estructura de control de selección.

La selección es una estructura de control, descrita en [18], que permite controlar la ejecución de las acciones que requieran de ciertas condiciones para su realización. De acuerdo con dichas condiciones se decide si las acciones correspondientes se ejecutarán o no. La selección presenta 3 formas: simple (Figura 2), doble, y múltiple (Figura 3). Se clasifican comúnmente por el número de alternativas de acción, es decir, si hay una, dos o más de dos respectivamente.

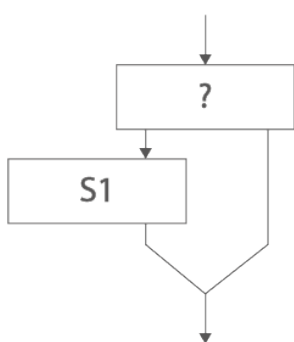


Figura 2. Selección simple.

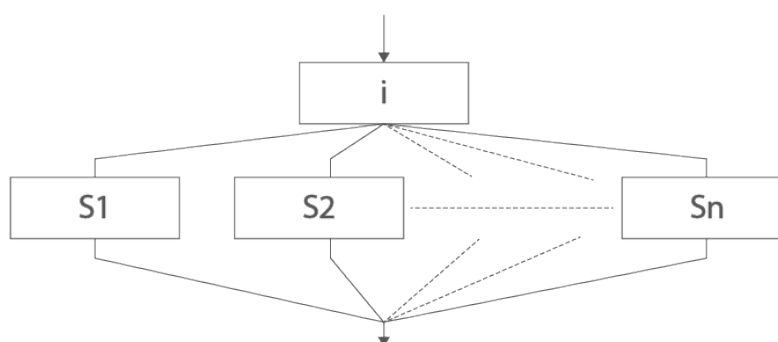


Figura 3. Selección múltiple.

La estructura de repetición, como se explica en [18], tiene como objetivo controlar que una acción o un grupo de acciones se ejecuten en más de una ocasión, es decir, permite formar ciclos repetitivos. La repetición consta de tres formas: “hacer hasta”, “para” y “hacer mientras”.

La repetición “hacer hasta” (Figura 4), permite controlar la repetición de una acción o grupo de acciones hasta que se cumpla la condición de terminación del ciclo. La repetición tipo “para” permite formar un ciclo controlado por un contador para el cual es necesario definir un valor inicial, un valor final y un incremento; esto significa que debe conocerse de antemano el número de veces que deberá de repetirse el ciclo.

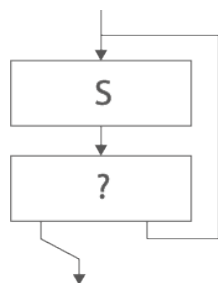


Figura 4. Repetición hacer hasta.

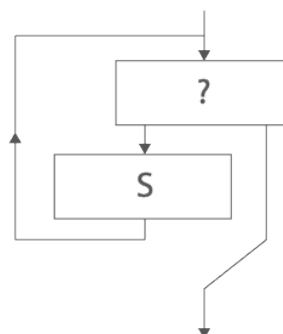


Figura 5. Repetición hacer mientras.

La repetición tipo “hacer mientras”, (Figura 5), permite realizar una repetición en un intervalo de 0 a N veces, esto se debe a que la condición de control se coloca al inicio de la estructura y solo se repetirá el ciclo mientras la condición sea verdadera, de lo contrario se terminará.

Al limitarse a utilizar solamente las tres estructuras de control antes mencionadas permite generar diagramas de flujo de una topología restringida, comparados con los diagramas que pueden crearse cuando el flujo de control puede dirigirse de cualquier punto a otro. Como se describe en [19], las estructuras de control comparten la propiedad de tener una sola entrada en la parte superior y una sola salida en la parte inferior, por lo que estas pueden ser interpretadas como una sola acción de una computación secuencial. La secuencia y selección pueden ser comprendidas mediante el razonamiento enumerativo y la repetición mediante la inducción matemática. Comparada con la mayor libertad de la programación no estructurada, el restringirse con toda humildad a las estructuras de control representa una disciplina necesaria que ha demostrado sus beneficios.

La principal desventaja de la programación estructurada afecta principalmente a aquellos que han aprendido a programar de forma no estructurada, ya que “los viejos hábitos son difíciles de eliminar” [16]. Otro inconveniente es que las estructuras producen programas más largos que los realizados de forma no estructurada. En el pasado la memoria era relativamente cara y limitada, por lo que se pagaba por hacer programas tan breves como fuera posible. Este no es el caso actualmente, ya no hay necesidad de ser breve programando. Hay, sin embargo, necesidad de ser claro programando para que los programas resulten fáciles de comprender, modificar y corregir.

2.1.2 Enfoques en la enseñanza y aprendizaje de la programación

La complejidad de los programas que se desarrollan actualmente requiere el uso de técnicas de programación efectivas, sin embargo, es necesario primero comenzar con estrategias de enseñanza y aprendizaje para ayudar a los estudiantes a utilizar correctamente dichas técnicas. Las estrategias de aprendizaje son procedimientos que presentan rasgos propios: su aplicación no es automática sino controlada, se componen de otros elementos más simples que constituyen técnicas o destrezas, y fomentan el uso selectivo de los propios recursos y capacidades disponibles.

La mayoría de los cursos de programación se imparten utilizando los enfoques tradicionales de la enseñanza, incluyendo una mezcla de exposiciones, lecturas y sesiones prácticas; desafortunadamente, este tipo de enfoques produce estudiantes pasivos que sólo reciben información.

La primera etapa de la enseñanza de la programación resulta un tanto tediosa para los estudiantes que están ávidos de utilizar la computadora por lo que utilizar diferentes enfoques, como la utilización de un editor de algoritmos [20], ayuda a disminuir la ansiedad por el uso de la computadora, facilita la nivelación entre los estudiantes que cuentan o no con conocimientos previos, y fomenta el uso posterior de las estrategias algorítmicas, aún después de aprender a desarrollar programas.

Un estudio para identificar los problemas referentes a la enseñanza y aprendizaje de la programación fue realizado en Malasia [21], los resultados identificaron cuatro problemas principales: la falta de habilidades para analizar problemas, el uso inefectivo de las técnicas de representación de problemas, uso inefectivo de las estrategias de enseñanza para la solución de problemas y codificación, y la dificultad para dominar la sintaxis y las funciones.

Se ha comprobado que el uso del método global, ver primero el problema como un todo y después las partes que lo componen, aplicado al aprendizaje de un lenguaje de programación ahorra tiempo y esfuerzos. Por consiguiente, se creó un ambiente de aprendizaje, descrito en [22], que incluye un editor interactivo de algoritmos, un constructor automático de trazas y un traductor a *Pascal*. Los resultados obtenidos muestran un avance considerable en el aprendizaje de los diferentes paradigmas de la programación.

El conocimiento que permite a los profesores transformar su entendimiento del tema de estudio en algo más accesible para sus alumnos se le conoce como el conocimiento del contenido pedagógico [23]. Algunas preguntas clave para revelar este conocimiento en el contexto de la programación son: ¿por qué enseñar programación?, ¿qué conceptos enseñar?, ¿cuáles son las dificultades en la enseñanza de estos conceptos?, y ¿cómo enseñar programación? Este conocimiento facilita la elección de los distintos enfoques en la enseñanza, como el de ofrecer un lenguaje de programación más simple para no enfocarse en la sintaxis, escoger cuidadosamente varios problemas a resolver para desarrollar el pensamiento algorítmico o enseñar con entornos de desarrollo adecuados.

Descrita en [24], se realizó una clasificación de los enfoques en la enseñanza de la programación organizada en: clase y laboratorios, visualización de software, robots, aprendizaje basado en problemas, aprendizaje cognitivo y otros. Los investigadores continúan con la búsqueda de enfoques alternativos o herramientas complementarias a las clases tradicionales para mejorar la enseñanza de la programación.

En [25] se identificaron varios enfoques pedagógicos en la enseñanza de la programación: el enfoque de programación estructurada que apoya el desarrollo de programas utilizando sólo las tres estructuras de control, enfoque de solución de problemas que enfatiza la importancia de solucionar los problemas antes de utilizar las herramientas de desarrollo, enfoque de desarrollo de software que incorpora el desarrollo de las habilidades de solución de problemas y las habilidades de programación en un único proceso, el enfoque de la programación pequeña que reduce la carga cognitiva a una escala menor a través de guías, enfoque de lenguajes de programación simples o control de actores, enfoque de enseñanza del lenguaje que está basado en una pedagogía de adquisición de un segundo lenguaje natural y, por último el enfoque de la teoría del aprendizaje que sugiere una evaluación referenciando criterios, como los objetivos de la taxonomía de Bloom.

Entre las estrategias de la enseñanza de la programación se consideran como los aspectos más importantes la manera de motivar al alumno y la forma de como se le hace llegar todo tipo de información, tanto de los profesores como de los propios alumnos. Según [26], no solo es importante instruir al alumno en el correcto uso de las nuevas tendencias tecnológicas, sino también enseñarle a crearlas de acuerdo a una necesidad en

particular. Para esto se debe trabajar mucho en el razonamiento lógico del alumno, puesto que de ahí surge la creatividad y destreza del cómo lograr sistematizar algún proceso que está funcionando de manera manual, es decir, sin el uso de ninguna herramienta informática.

Desafortunadamente, en la actualidad existe una falta de evidencia empírica sobre la efectividad de estos enfoques; sin embargo, los siguientes son considerados aspectos de un enfoque o herramienta efectiva: atraen al estudiante a un aprendizaje activo, permiten al estudiante cierto grado de control sobre su aprendizaje, cuenta con mecanismos de ayuda, retroalimentación justo a tiempo, uso de colaboración, incorporación de auto análisis y representaciones múltiples de los conceptos, como texto, imagen y sonido. Existen una variedad de interrogantes que necesitan ser respondidas, por lo que serán necesarios estudios futuros, ya que de otra manera, nuestros programadores de computadoras en el futuro no tendrán las habilidades necesarias para crear nuevas aplicaciones, sino que serán solamente usuarios de programas creados por otros. En la era de los trabajadores de la tecnología y conocimientos digitales, estas son habilidades inadecuadas que necesitan atención en el campo de la educación tecnológica.

2.2. Análisis y traducción de código fuente

El código fuente representa las instrucciones que ejecutará la computadora, sin embargo, es necesario traducirlo mediante un compilador al lenguaje que la máquina requiere para su ejecución. El compilador primero analizará que las palabras que conforman el código estén bien escritas, que el orden de estas palabras forme sentencias válidas, que estas sentencias tengan sentido en el contexto correspondiente y, si no hay errores, realizará finalmente la traducción necesaria.

2.2.1 Compiladores

Al igual que los lenguajes naturales, como el inglés, francés o ruso, los lenguajes de programación definen una manera de estructurar palabras en enunciados para comunicar información. Mientras los lenguajes naturales comunican sentimientos, hechos y preguntas del mundo real, un lenguaje de programación, como se define en [27], consta de notaciones para describir cálculos a personas y a máquinas. El mundo como lo conocemos depende

de los lenguajes de programación, ya que todo el software ejecutándose en todas las computadoras fue escrito en alguno de estos lenguajes; sin embargo, antes de que el programa pueda ejecutarse, primero debe ser traducido a alguna forma entendible para la computadora. Por ejemplo, cuando un estadounidense y un francés desean comunicarse y ninguno de los dos conoce el lenguaje del otro, es necesaria la ayuda de algún tercero, de un traductor.

Un compilador es un programa que actúa como un traductor, recibe las sentencias escritas en algún lenguaje de programación y, si tienen sentido en ese lenguaje, las traducirá a sentencias con el mismo significado en otro lenguaje de computadora [28]. Una secuencia de enunciados en un lenguaje de programación es un programa. El compilador traduce el programa de un lenguaje de programación, llamado el lenguaje fuente a otro programa, esto es otra secuencia de enunciados, a otro lenguaje llamado lenguaje objetivo [28]. Por lo regular, el lenguaje fuente es un lenguaje de alto nivel, como C o C++, mientras que el lenguaje objetivo es código máquina, es decir, código escrito en las instrucciones de máquina correspondientes a la computadora en la cual se ejecutará [29].

De acuerdo a los diferentes tipos de código y las diversas formas de funcionamiento, podemos distinguir otros tipos de traductores. Ensamblador: Un ensamblador es un compilador donde el lenguaje fuente es un lenguaje ensamblador y el lenguaje objetivo es el código máquina [30]. Intérprete: Un intérprete no genera código objetivo, sino que analiza y ejecuta directamente cada sentencia del código fuente [31]. Preprocesador: Entre las tareas de un preprocesador se encuentran la sustitución de macros, la inclusión de archivos y la extensión del lenguaje.

Existen reglas para definir que tiene sentido en cada lenguaje, el compilador aplica estas reglas para determinar si la entrada tiene sentido y para confirmar que la salida también tenga sentido [27]. Estos principios y técnicas del diseño de compiladores son aplicables a muchos otros dominios que, seguramente, serán reutilizados continuamente durante la carrera de un profesional de la computación. El estudio de compiladores abarca los lenguajes de programación, las arquitecturas de las máquinas, la teoría del lenguaje, la algoritmia y la ingeniería de software.

El proceso de compilación se divide principalmente en el análisis y la síntesis [27]. La parte de análisis, también conocida como *front end*, separa el programa fuente en sus piezas constitutivas para crear una representación intermedia de este. La parte de análisis también recolecta información acerca del programa y la almacena en una tabla de símbolos. Si en el análisis se detecta que el programa está mal formado, entonces es necesario proveer mensajes informativos para que el usuario pueda tomar acciones correctivas. La parte de síntesis, o *back end*, construye el programa objetivo a partir de la representación intermedia y de la información en la tabla de símbolos.

2.2.2 Análisis léxico

El análisis léxico es la primera fase de un compilador, como se describe en [31]. Al iniciar el proceso, el código fuente de un programa no es más que un flujo de caracteres, así que la tarea del analizador léxico es reconocer símbolos en este flujo de caracteres y representarlos de una forma más útil para la siguiente fase. El analizador léxico lee los caracteres del código, carácter por carácter, los agrupa en lexemas, y produce como salida una secuencia de *tokens* por cada lexema en el programa fuente [27].

En la mayoría de los lenguajes de programación se consideran *tokens* a ciertos caracteres relacionados entre sí [30], como las palabras clave, operadores, identificadores, constantes, y signos de puntuación (paréntesis, coma y punto y coma) [27]. El agrupamiento de caracteres en *tokens* depende del lenguaje, es decir, un lenguaje generalmente agrupará caracteres en *tokens* diferentes a los de otro lenguaje.

Según [28], un lenguaje de programación típico puede ser definido con alrededor de 50 a 100 *tokens* diferentes, en donde cada *token* es representado en el archivo fuente por una cadena de varios caracteres. Los *tokens* se deben distinguir claramente de las cadenas de caracteres que representan. Por ejemplo, el *token* de la palabra reservada *if* se debe distinguir de la cadena de caracteres “*if*” que representa. En [29] se indica que para hacer clara esta distinción a la cadena de caracteres representada por un *token* se le denomina lexema. Algunos *tokens* tienen sólo un lexema, las palabras reservadas tienen esta propiedad, no obstante, un *token* puede representar un número infinito de lexemas.

Los *token* pueden tener valores asociados a estos, llamados atributos del *token*, como pueden ser el tipo, los lexemas, o un valor numérico [29]. Por ejemplo, si tenemos una variable con nombre *x*, su atributo tipo será identificador y su lexema “*x*”. En el caso de un token de símbolo especial, como *+*, no sólo se tiene el atributo de lexema “*+*” y el tipo operador, sino también la operación aritmética real asociada como atributo a este símbolo. Los *tokens* también pueden ser de dos tipos [30]: cadenas específicas como palabras reservadas o puntos y comas, y no específicas, como identificadores o constantes, la diferencia entre ambos tipos radica en si ya son conocidos previamente o no. Por ejemplo, la palabra reservada *while* es conocida previamente en un lenguaje que la utilice, pero el nombre de una variable no es conocido, ya que es el programador será quien le da el nombre en cada programa.

Ya que el analizador léxico es la parte del compilador que lee el texto fuente, este puede efectuar ciertas tareas adicionales a la identificación de *tokens*, como eliminar las partes no esenciales para la siguiente fase, es decir, realiza la función de preprocesador en cierta medida [30]. El preprocesador, descrito en [27], lleva a cabo procesos simples a la entrada que no requieren el reconocimiento de *tokens*, como la eliminación de comentarios, la compactación de espacios en blanco consecutivos a uno solo, y el registro de los caracteres de salto de línea detectados para, de esta forma, poder asociar un número de línea a cada mensaje de error.

Es común para el analizador léxico el interactuar con la tabla de símbolos [28], como al momento de detectar un lexema de tipo identificador será necesario ingresarlo a la tabla de símbolos. En algunos casos, la información acerca del tipo del identificador puede ser leída de la tabla de símbolos por el analizador léxico para determinar el *token* adecuado que se enviará a la siguiente fase, al análisis sintáctico. Cada lexema es reconocido y convertido a un solo *token* en el flujo de salida, por lo que un flujo de varios *tokens* nunca es realmente recolectado. En cualquier instante existe un solo *token*, en el cual el analizador sintáctico estará trabajando después.

Los autómatas finitos deterministas (Figura 6) son utilizados para la construcción de los analizadores léxicos [29], ya que permiten describir el proceso de reconocimiento de patrones de caracteres. Es posible describir los patrones de los *tokens* al enumerar una

secuencia de estados y transiciones del autómata o por medio de una expresión regular. Es de suma importancia diseñar un autómata eficiente ya que la mayoría del tiempo de la compilación se ocupa en el análisis léxico.

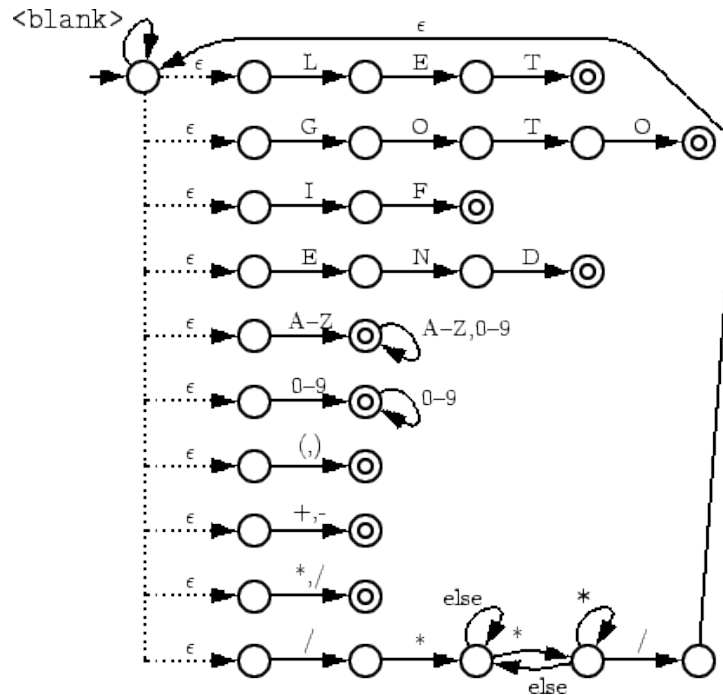


Figura 6. Autómata Finito Determinista.

2.2.3 Análisis sintáctico

El analizador sintáctico, o *parser*, recibe como entrada los *tokens* que le suministra el analizador léxico y comprueba que estén ordenados de forma correcta, dependiendo del lenguaje [30]. La mayoría de los lenguajes de programación permiten la construcción de sentencias con paréntesis anidados y balanceados, estos lenguajes son libres de contexto [28]. La sintaxis de un lenguaje libre de contexto se determina a través de reglas de una gramática libre de contexto; aplicando estas reglas, el *parser* comprobará si la cadena puede ser generada.

Una gramática libre de contexto [29] es una especificación para la estructura sintáctica de un lenguaje de programación que utiliza convenciones y operaciones muy

similares a las correspondientes en las expresiones regulares, con la única diferencia de que sus reglas son recursivas. Este cambio aparentemente elemental para el poder de la representación tiene enormes consecuencias.

El analizador sintáctico, ya sea de forma explícita o implícita, construye un árbol que representa la estructura sintáctica del programa [29]. La estructura del árbol sintáctico depende en gran medida de la estructura sintáctica particular de cada lenguaje. Este árbol se define como una estructura de datos dinámica, en la cual cada nodo se compone de un registro cuyos campos incluyen los atributos necesarios para el resto del proceso de compilación.

La construcción del árbol hace posible el diferenciar entre aplicar algunos operadores antes que otros en la evaluación de expresiones [30]. Por ejemplo, en la expresión $7 + 8 * 9$, el resultado de la evaluación dependerá de si aplicamos antes el operador de multiplicación (Figura 7) que el operador suma (Figura 8). Una manera adecuada de saber qué operador se debe aplicar antes, en este caso, es elegir qué árbol sintáctico generar de entre los dos posibles.

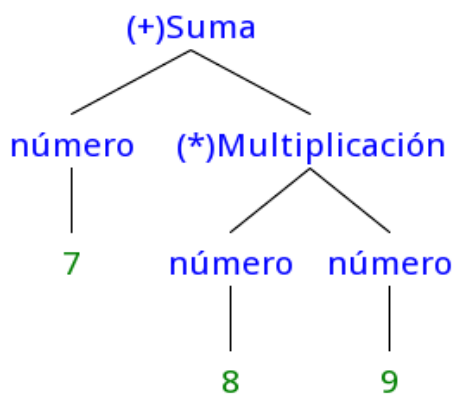


Figura 7. Árbol sintáctico $7 + (8 * 9)$

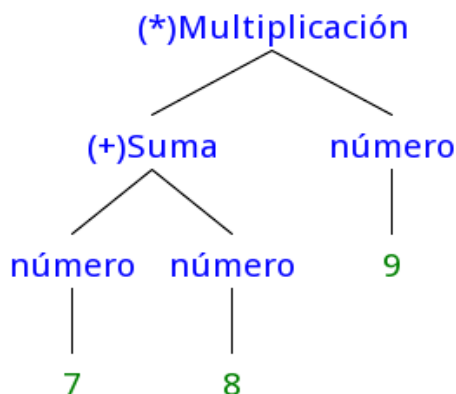


Figura 8. Árbol sintáctico $(7 + 8) * 9$

Existen tres tipos generales de *parsers*: universales, descendentes (*top-down*) y ascendentes (*bottom-up*) [27]. Los métodos universales, como el algoritmo *Cocke-Younger-Kasami* y el algoritmo de *Earley* pueden procesar cualquier gramática, sin embargo son

muy ineficientes para emplearlos en los compiladores de uso para la producción de aplicaciones complejas. Como implica su nombre, el método *top-down* construye el árbol de la sintaxis desde la raíz hasta las hojas y se basa en gramáticas LL [31], esto es que la entrada será leída de izquierda a derecha y que las derivaciones serán también por la izquierda. El método *bottom-up* inicia por las hojas y trabaja hasta la raíz, este se basa en gramáticas LR [31], donde la entrada se lee de izquierda a derecha y las derivaciones serán por la derecha. Los métodos más eficientes de *top-down* y de *bottom-up*, trabajan solamente con ciertas subclases de gramáticas LL y LR, sin embargo, estas gramáticas son lo suficientemente expresivas para describir la mayoría de los lenguajes de programación modernos; de cualquier forma, la entrada del *parser* siempre se escanea de izquierda a derecha, un símbolo a la vez.

El analizador sintáctico es el motor controlador del compilador [28], controla al analizador léxico y al generador de código; reconoce la estructura de las sentencias, sintaxis, del programa de entrada representado por *tokens*. Normalmente la salida de un *parser* es un árbol de la sintaxis, pero no siempre es necesaria su construcción, puede que en cualquier instante sólo se requiera la existencia de un fragmento del árbol.

2.2.4 Análisis semántico y traducción dirigida por la sintaxis

Un compilador no sólo tiene que revisar la estructura sintáctica del código fuente, también necesita inspeccionar si la semántica corresponde a la del lenguaje de programación. La semántica de un programa es su significado, por lo que determina su comportamiento durante el tiempo de ejecución [29]. El analizador semántico toma el árbol sintáctico de la fase anterior y lleva a cabo una serie de comprobaciones complejas. Como se describe en [30], es necesario comprobar que los operadores trabajan sobre tipos compatibles, si se obtienen como resultado elementos con tipos apropiados y si las llamadas a subprogramas tienen los parámetros adecuados, tanto en número como en tipo. En resumen, su tarea es revisar el significado del código para descubrir si tiene sentido.

Estas comprobaciones de consistencia que se efectúan antes de la ejecución del programa fuente se dominan comprobaciones estáticas, mientras que aquellas realizadas

durante la ejecución son las comprobaciones dinámicas [31]. La revisión de la sintaxis es un ejemplo de comprobación estática; mientras que la comprobación de tipos puede efectuarse de forma estática o dinámica. La mayoría de los lenguajes de programación tienen características que se pueden detectar antes de la ejecución y no pueden descubrirse mediante el análisis sintáctico; se conoce a tales características como semántica estática, y su revisión es la tarea del analizador semántico [29].

Esto implica que el análisis semántico debe garantizar que se consideren todas las reglas dependientes del contexto del lenguaje de programación [31], como declarar los identificadores antes de utilizarse o la comprobación de tipos, que es una forma de garantizar que los identificadores relacionados sean de tipos compatibles. Dos identificadores se relacionan al formar el lado izquierdo o derecho de un operador, al formar el lado izquierdo o derecho de una sentencia de asignación o al ser parámetros reales o formales de una función.

Los nodos del árbol de la sintaxis pueden describir ciertos atributos como su valor numérico, si es una expresión, o la ubicación en memoria de una variable, si es un identificador. Estos atributos no son definidos en la gramática libre de contexto, pero son parte del lenguaje. A los árboles de la sintaxis en los que sus nodos guardan atributos se les suele llamar árboles decorados. Los atributos pueden ser de dos tipos, sintetizados y heredados [30]; los atributos sintetizados se calculan a partir de los valores de los atributos de sus nodos hijos en el árbol decorado y los atributos heredados se calculan a partir de los atributos de los nodos hermanos o padres.

El análisis semántico puede efectuarse en paralelo con el análisis sintáctico, es decir, puede efectuarse el análisis semántico cada vez que el analizador sintáctico reconoce una estructura sintáctica o producción. Como se explica en [31], las acciones del análisis semántico relacionadas con las producciones se incorporan al proceso del análisis sintáctico. Estas acciones semánticas generan código intermedio cada vez que se ejecutan; de esta forma, la estructura sintáctica del código fuente dirige a la traducción, por lo cual hablamos de una traducción que es dirigida por la sintaxis.

Es llamada traducción dirigida por la sintaxis debido a que las secuencias de generación de código son definidas en la gramática que especifica la sintaxis del lenguaje

[28]. En otras palabras, el *parser* es capaz de elegir las secuencias apropiadas de código máquina estrictamente en base a la sintaxis, sin depender de la semántica, excepto para valores constantes y direcciones máquina de las variables o funciones.

Cuando se realiza una traducción dirigida por la sintaxis se dota a cada nodo del árbol de la sintaxis de una serie de atributos al aplicar las reglas de una definición dirigida por la sintaxis, que es una gramática libre de contexto con atributos asociados a los símbolos de la gramática y reglas asociadas a las producciones [27]. Al recorrer el árbol se van ejecutando las acciones semánticas correspondientes, al reconocer la producción apropiada, para generar una representación interna del código fuente a compilar [30].

2.2.5 Lex

Lex es una herramienta para la creación de analizadores léxicos. Para su utilización es necesario especificar una tabla de expresiones regulares que corresponden a los lexemas de los *tokens*. La tabla es traducida a un programa que lee un flujo de entrada y lo divide en cadenas que concuerdan con las expresiones dadas, segmentando la entrada y preparándola para un análisis sintáctico [29]. El reconocimiento de las expresiones es realizado por un autómata finito determinista generado a partir de las expresiones regulares [32].

Los programas de análisis léxico generados con *Lex* aceptan especificaciones ambiguas y por defecto eligen la cadena más larga posible que concuerde en cada punto de la entrada, esto es el principio de *Maximum Munch*. La versión más popular de *Lex* se conoce como *Flex*, por *fast lex*; se distribuye como parte del paquete compilador *Gnu* que produce la *Free Software Foundation* y se encuentra disponible gratuitamente en Internet.

2.2.6 Yacc

Yacc, *Yet Another Compiler Compiler*, es una herramienta que facilita la creación de analizadores sintácticos [29]. Toma como entrada una especificación de la sintaxis de un lenguaje y produce como salida un procedimiento de análisis sintáctico para ese lenguaje. Históricamente a los generadores de analizadores sintácticos se les ha conocido como compiladores de compiladores [33].

El usuario de *Yacc* especifica las estructuras que serán detectas junto con el código a ser invocado cada vez que alguna sea reconocida. *Yacc* transforma estas especificaciones en una subrutina que maneja el proceso de la entrada utilizando un algoritmo *bottom-up* simplificado, en lugar de los algoritmos *top-down* más restrictivos [28]. Un lenguaje de entrada puede ser tan complejo como un lenguaje de programación o tan simple como una secuencia de números. *Yacc* es uno de los generadores de *parsers* más populares y de amplio uso.

2.3 Animación de algoritmos

El diseño de animaciones ilustrativas de los algoritmos es un difícil reto psicológico y perceptual. ¿Qué información debería presentarse y cómo? Actualmente, la creación de animaciones de algoritmos efectivas es un arte y no una ciencia; las técnicas de color y sonido son herramientas de este arte. Como se describe en [34], la animación de algoritmos busca ilustrar el comportamiento de un programa visualizando las operaciones fundamentales que el programa ejecuta; han demostrado ser muy útiles para la educación y la investigación en el diseño y análisis de algoritmos.

La animación de algoritmos permite visualizar el comportamiento de un algoritmo al producir una abstracción tanto como de los datos como de las operaciones del mismo [35]. Inicialmente se mapea el estado actual de un algoritmo a una imagen que después es animada basada en las operaciones entre estados exitosos de la ejecución. El animar un algoritmo permite una mejor comprensión del funcionamiento interno y hace evidente sus deficiencias y ventajas para una futura optimización.

El primer sistema interactivo de animación de algoritmos fue *BALSA*, el cual introdujo el concepto de separar el algoritmo de su animación y el marcar eventos interesantes que controlarán la ejecución [34]. *Animus* fue un pionero de las animaciones fluidas, *TANGO* introdujo el paradigma de transición de caminos para especificar las animaciones, *Zeus* agregó color y sonido, y finalmente junto a *Polka-3D* exploraron el uso de gráficas 3D para la visualización de algoritmos.

La animación de algoritmos a través del sistema *Jeliot* [36] se realiza mediante un intérprete visual que transforma los objetos de datos del algoritmo a sus contrapartes visuales en la animación. La presentación se basa en una metáfora de teatro; se considera a

la animación completa como una presentación teatral. El guion de la obra es el algoritmo visualizado y el escenario es la ventana donde se muestra la animación. La obra es realizada por los actores, que son entidades gráficas con atributos visuales como tamaño, forma, color y ubicación. Cada actor tiene un rol, que corresponde a cada objeto de datos del algoritmo; la apariencia de cada actor es determinada por el director, es decir, el usuario que diseña la animación. Esto es importante, ya que cada programador visualiza una variable entera de forma muy diferente, sin mencionar los tipos de datos más complejos.

Algunas técnicas desarrolladas para el sistema de animación Zeus [37] incluyen las vistas múltiples, que son conceptualmente más simples y fáciles de implementar; las señales del estado, que muestran gráficamente los cambios en las estructuras de datos del algoritmo; el historial estático, que es una vista para resaltar eventos cruciales a través de la ejecución; las animaciones continuas, que representan cambios con transiciones suaves contra las discretas; la comparación de múltiples algoritmos al mismo tiempo; y la selección de los datos de entrada, ya que influye en el mensaje contenido en la animación.

La utilización de colores tiene el potencial de resolver diversos problemas en la animación de algoritmos por la virtud misma del color para comunicar una gran cantidad de información efectivamente. Como se describe en [37], se utiliza el color para codificar el estado de las estructuras de datos, resaltar la actividad, unir las vistas, enfatizar los patrones y hacer el historial visible. La utilización de sonidos se emplea para reforzar lo que está siendo mostrado visualmente y para identificar patrones. Al ser los sonidos intrínsecamente dependientes del tiempo, son muy efectivos para representar fenómenos dinámicos como la ejecución de los algoritmos. Una de las tareas más importantes en la animación de algoritmos es el diseño de la apariencia de las visualizaciones [35], ya que se debe tomar en cuenta diferentes problemas, como la información que deberá ser mostrada, cómo se representará y qué aspectos deberán resaltarse.

2.3.1 Técnicas de animación web 3D

Desde sus humildes inicios de un entorno limitado y sencillo, actualmente la web es una sorprendente colección de páginas con todo tipo de aplicaciones para el entretenimiento y la productividad. Los usuarios pueden realizar varias tareas en la web, como comprar

productos e interactuar en tiempo real con otros usuarios por todo el mundo; sin embargo, falta un elemento clave en la web, el 3D. Como se menciona en [38], en la actualidad el 3D es utilizado en línea principalmente en aplicaciones como juegos y mundos virtuales, los cuales son procesados utilizando computadoras poderosas y software especializado; no obstante, las grandes firmas de negocios e ingeniería, entre otros usuarios, también desean el realismo y el detalle adicional que agrega el 3D.

La representación 3D en la web puede proveer una herramienta útil y valiosa para ilustraciones atractivas en diversas áreas [39], incluyendo las aplicaciones médicas, museos, publicidad, entretenimiento y educación. Las interfaces visuales 3D son efectivas, atractivas y consideradas como un factor clave para agregar valor a las aplicaciones y mejorar la experiencia global del usuario [40].

La primera aparición de contenido 3D en la web comienza en 1994 con *VRML*, *Virtual Reality Markup Language* [41], un formato basado en texto para especificar escenas 3D en términos de geometría y propiedades de materiales; sin embargo, esta tecnología no tuvo gran éxito debido a que requería la instalación de *plug-ins* específicos para cada plataforma, además de su bajo rendimiento a la hora de procesar modelos y escenas complejas en tiempo real. El uso del 3D en la Web continuó en los noventa con tecnologías como *Java3D* y *Shockwave*, pero diversos obstáculos limitaron también a su amplia aceptación.

Antes de que el 3D en la web pueda realmente popularizarse, primero es necesario superar numerosos obstáculos. Por ejemplo, el uso de *plug-ins* en algunas tecnologías produce problemas ocasionales en los navegadores, por lo que se requiere que los navegadores puedan procesar el contenido 3D nativamente. La falta de estandarización también es un problema, ya que si la estandarización no ocurre, la web se saturará con numerosos formatos incompatibles, forzando a los desarrolladores a crear múltiples versiones del mismo contenido para ejecutarse en diferentes navegadores [38]. Además, aunque las capacidades del hardware y el ancho de banda se han incrementado significativamente, aún pueden no ser suficientes para desplegar modelos 3D altamente complejos.

En 2011, el *Kronos Group* liberó la especificación para *WebGL 1.0* que fue adoptada por *Mozilla Firefox*, *Google Chrome* y otros. *WebGL* [40] es una librería de

software basada en *OpenGL ES 2.0* que puede ser invocada por medio de *JavaScript* para crear gráficas 3D en el elemento *canvas* de *HTML5*; no obstante, todavía es muy laborioso para los desarrolladores el crear aún una simple escena 3D.

Debido al estado primitivo de *WebGL*, se han desarrollado múltiples librerías de alto nivel en *JavaScript* para facilitar el desarrollo web 3D, como *GLGE*, *Three.js* y *Angle*, que fueron rápidamente adoptadas por un gran número de desarrolladores de juegos, fans del 3D y diseñadores web. Estas herramientas están diseñadas para superar el entorno poco amigable de ejecución bajo el cual las aplicaciones gráficas web son ejecutadas, como lo son un pobre poder de cómputo, acceso limitado al sistema de archivos y una falta de soporte multihilo. Como se presenta en [41], *NetGL* es otra herramienta que simplifica el desarrollo en *WebGL*, ya que ofrece un marco de trabajo que incorpora múltiples herramientas auxiliares de desarrollo, es amigable con la web actual y está lista para la siguiente generación web.

Los usuarios se están acostumbrando cada vez más al contenido 3D gracias a las tecnologías de las películas y videojuegos, por lo que la demanda de contenido 3D en la web es urgente [41]. La tecnología *WebGL* es actualmente el desarrollo más prometedor para el 3D en la web, ya que ofrece gráficas 3D con aceleración por hardware sin necesidad de *plug-ins*; además, el avance en dispositivos móviles permite la aceleración de las gráficas, y los navegadores móviles soportan la tecnología *HTML5* [39], por lo que la visualización de gráficas 3D puede ser implementada en las aplicaciones web móviles. Se pronostica un incremento en el uso del contenido 3D en la web debido a los cada vez más populares dispositivos móviles y que, eventualmente, el 3D en la web se desempeñará tan bien como el video en la web lo ha hecho [38].

Capítulo 3. Materiales y Método

En el siguiente capítulo se exponen los recursos utilizados y los pasos realizados para el desarrollo del proyecto. Comenzando por una descripción del área de estudio, se revisan las áreas de las ciencias de la computación que fueron de interés para el desarrollo, como lo son la teoría de lenguajes de programación, teoría de autómatas, teoría de grafos y compiladores. Después, se indica el tipo de investigación y se listan todos los recursos y materiales que fueron utilizados durante todo o parte de las etapas del proyecto; estos incluyen una variedad de herramientas de software, como las de diseño y animación, generadores de analizadores léxicos y sintácticos, así como también librerías de gráficas 3D. En seguida, se detalla la secuencia de pasos que fueron llevados a cabo para la realización del proyecto, comenzando con una indagación de los trabajos similares, así como de los conceptos y técnicas requeridas; continuando con el proceso llevado a cabo para la realización del analizador léxico, analizador sintáctico y el intérprete. Posteriormente, se describe la realización de los modelos y escenas 3D, los retos relacionados al diseño e iluminación, así como también las técnicas utilizadas para la animación y, finalmente, la integración de las animaciones desarrolladas con el ambiente dinámico del intérprete.

3.1 Descripción del área de estudio

La teoría de compiladores y la teoría de lenguajes de programación fueron fundamentales para el desarrollo del proyecto, ya que debido al diseño de la aplicación, fue necesario analizar el código fuente ingresado por los estudiantes. Iniciando con el análisis léxico, basado en la teoría de autómatas, se reconocieron los *tokens* o componentes significativos del lenguaje. La teoría de lenguajes de programación fue necesaria para el diseño de la aplicación, que abarcó la definición de los *tokens* y la gramática del lenguaje.

La gramática libre de contexto fue diseñada definiendo las reglas de producción; esta describe la estructura deseada de los *tokens*. El análisis sintáctico basado en una gramática determinada, generó otra representación más del programa, el árbol de la sintaxis; los conceptos básicos de teoría de la computación, algoritmia y estructuras de

datos fueron necesarios para realizar el recorrido recursivo de los nodos del árbol e interpretar el programa.

El diseño de un lenguaje de programación es una tarea compleja que requirió la toma de decisiones difíciles; por ejemplo, el seleccionar la inclusión de alguna característica de entre otras incompatibles y elegir entre expresividad o eficiencia. Como se describe en [42], si el lenguaje se enfoca en una sola área de aplicación se puede minimizar la mayor dificultad en el diseño de un lenguaje: el sacrificar buenas ideas. Para el proyecto actual, se diseñó un lenguaje con una sintaxis muy parecida a la del lenguaje C, sin embargo, éste representó sólo un subconjunto de C.

La animación de algoritmos, descrita en [35], forma parte del campo de estudio de la visualización de software; esta estudia el comportamiento de un algoritmo produciendo una abstracción tanto de los datos como de las operaciones. Como apoyo para la enseñanza y aprendizaje, las visualizaciones de algoritmos pueden ofrecer una alternativa llamativa a otros tipos de enseñanza, como presentaciones con pseudocódigo o código real escrito. El estado actual del campo de visualizaciones de algoritmos, descrito en el reporte [43], indica que no se ha concluido necesariamente que la visualización de algoritmos sea efectiva en la práctica; algunos hallazgos no muestran diferencias significativas en la efectividad pedagógica, sin embargo, la mayoría de los investigadores siguen positivos. Las visualizaciones tienen potencial para la enseñanza, pero crearlas y utilizarlas efectivamente es difícil.

3.2 Materiales

A continuación, se presentan los recursos de hardware, software y servicios que fueron utilizados durante todo o parte del desarrollo del proyecto.

3.2.1 Hardware

Título y características	Descripción
Laptop ASUS K50IJ: Intel Pentium Dual Core, 3GB RAM, 320 GB HDD	Equipo utilizado para el desarrollo de la aplicación

Tabla 1. Hardware utilizado para el desarrollo del proyecto.

3.2.2 Software y servicios

Título y versión	Descripción
Google App Engine SDK for Python v1.7.5	Kit de desarrollo para aplicaciones web
Google AppSpot Cloud Service	Servicio de alojamiento para aplicaciones web
Mercurial hg v2.5.4	Sistema de control de versiones
BitBucket.org	Servicio en línea compatible con Mercurial hg
Tortoise hg v2.3.4	Interfaz gráfica para el sistema Mercurial hg
PLY(Python Lex-Yacc) v3.4 por David M. Beazley	Generador de analizadores léxicos y sintácticos
jQuery JavaScript Library v1.9.0	Librería para el desarrollo de aplicaciones web
jQuery UI v1.10.1	Librería de controles para la interfaz gráfica
CodeMirror v3.1 por Marijin Haverbeke	Editor web de código con resaltado de sintaxis
Three.js r56 por Ricardo Cabello y Larry Battle	Librería para el desarrollo de aplicaciones WebGL
Tween.js por Soledad Penadés	Librería que implementa las ecuaciones de interpolación de Robert Penner
Google Chrome Browser v25	Navegador web que incluye herramientas de desarrollo y depuración de aplicaciones web
Sublime Text 2 v2.0.1	Editor de código fuente
Blender v2.66	Aplicación de modelado y animación 3D

Tabla 2. Software y servicios utilizados para el desarrollo del proyecto.

3.3 Método

En la siguiente sección se describe la secuencia de pasos realizada para el desarrollo del proyecto, comenzando por el tipo de investigación y la indagación de proyectos similares, así como de la evaluación de los antecedentes y las técnicas necesarias para el desarrollo; enseguida, se detalla el diseño del analizador léxico y sintáctico, el desarrollo del intérprete y, finalmente, se describe el proceso de diseño en 3D, de las animaciones y su integración con la aplicación.

3.3.1 Tipo de investigación

La investigación y el desarrollo engloba tres actividades: investigación básica, investigación aplicada y desarrollo experimental; el actual proyecto corresponde al tipo de investigación conocida como desarrollo experimental.

El manual de frascati [44] define el desarrollo experimental como “el trabajo sistemático, basado en el conocimiento existente obtenido de la investigación y la experiencia práctica, que va dirigido a la producción de nuevos materiales, productos o dispositivos, a la puesta en marcha de nuevos procesos, sistemas y servicios o a la mejora sustancial de los ya existentes”.

Es difícil diferenciar con precisión entre el desarrollo experimental y otros tipos de actividades. La Fundación Americana de la Ciencia (NSF) de Estados Unidos [44] proporciona una base práctica para juzgar estos casos: “Si el objetivo principal es introducir mejoras técnicas en el producto o en el proceso, la actividad se puede definir como de I+D. Si, por el contrario, el producto, el proceso o la metodología ya están sustancialmente establecidos y el objetivo principal es abrir mercados, realizar la planificación previa a la producción o conseguir que los sistemas de producción o de control funcionen armónicamente, la actividad ya no es de I+D”. Existe también otro tipo de investigación y desarrollo en la informática; este es el desarrollo experimental cuyo fin sea resolver la falta de conocimientos tecnológicos necesarios para desarrollar un sistema o programa informático.

3.3.2 Evaluación de aplicaciones educativas

Partiendo de los antecedentes se encontraron que la mayoría de las aplicaciones que ofrecen animaciones llamativas, como *Scratch* (Figura 9) y *Alice* (Figura 10), están orientadas hacia los niños pequeños o de secundaria; además son dependientes de un estilo de programación completamente visual. Por otra parte, otro tipo de aplicaciones orientadas un nivel intermedio, incluyen un editor de texto para ingresar sentencias utilizando un lenguaje de programación real, como las aplicaciones de *Greenfoot* y *Jeliot*, aunque despliegan animaciones más sencillas.

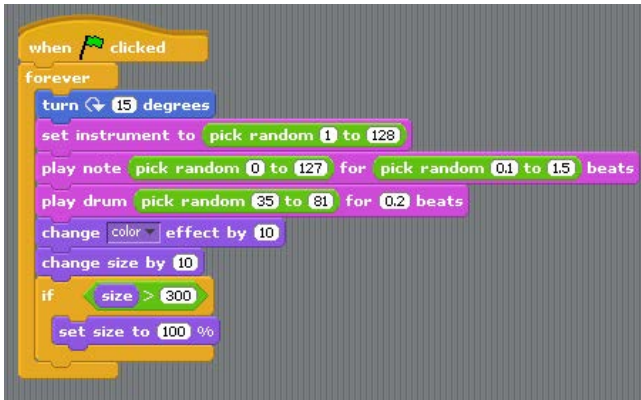


Figura 9. Interfaz de bloques en Scratch.

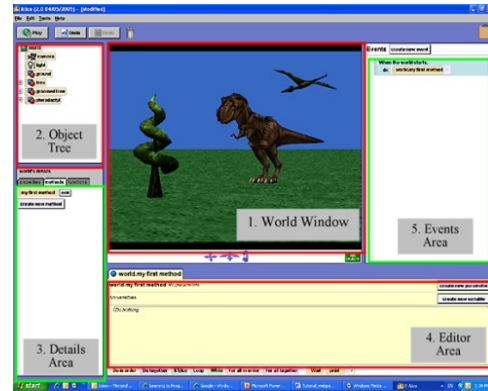


Figura 10. Interfaz de Alice.

El enfoque de actor utilizado por algunas aplicaciones permite que mediante las instrucciones que el alumno ingresa en el editor de código, sea posible visualizar el comportamiento de dicho actor en un mundo virtual, cómo es el caso de *Greenfoot* y *Turtle Graphics*. Sin embargo, aunque este enfoque ayuda a comprender los conceptos de control, deja fuera los demás conceptos de la programación estructurada.

De acuerdo a las anteriores delimitaciones propuestas para el proyecto actual, la aplicación se orientó a estudiantes universitarios, por lo que se seleccionó el diseño de nivel intermedio que incluyó un editor de texto convencional para ingresar el código fuente.

La mayoría de las aplicaciones revisadas requirieron la instalación de software adicional, como la máquina virtual de Java; en las instituciones públicas se restringe con regularidad la instalación y ejecución de software no autorizado, dificultando el acceso a las aplicaciones, por lo que se optó por crear una aplicación Web; ya que actualmente es una plataforma importante que permite ejecutar la aplicación en cualquier sistema operativo, así como en dispositivos móviles; además, no requerirá de la *Java Virtual Machine* o instalación alguna, como la mayoría de las aplicaciones actuales, facilitando el acceso y uso tanto a los estudiantes como a los profesores.

Las gráficas 3D son llamativas para los estudiantes, como lo ha demostrado la aplicación de *Alice*, ya que ofrecen una retroalimentación espacial para cierto tipo de metáforas; se seleccionó hacer uso de gráficas 3D en el navegador que gracias a las nuevas tecnologías web, ahora fue posible procesar gráficas detalladas en tiempo real.

Entre las metáforas más relevantes, la aplicación *Jeliot* (Figura 11) maneja animaciones de conceptos más abstractos como las variables, evaluación de expresiones y

entornos de funciones. Por otra parte, se encontró la metáfora de un actor o robot que es afectado por las instrucciones del usuario, como el caso de *Greenfoot* (Figura 12) o *Turtle graphics*, mostrando animaciones de personajes muy concretos.

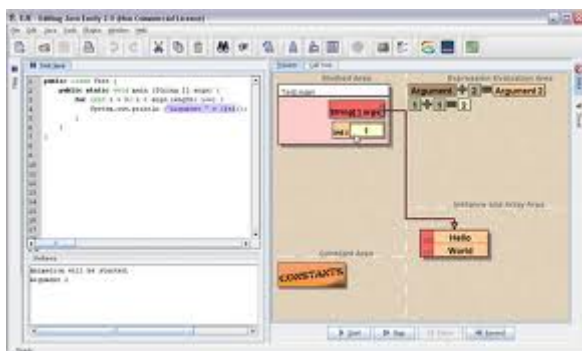


Figura 11. Interfaz de Jeliot.

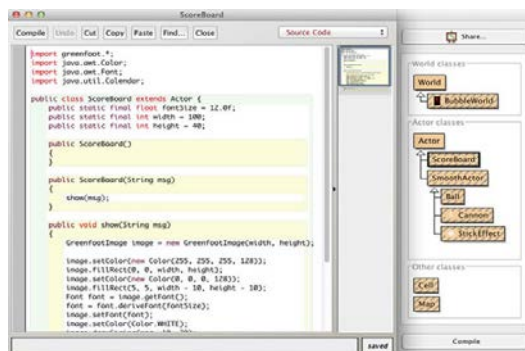


Figura 12. Interfaz de Greenfoot.

Se decidió optar por representar los conceptos manejados por *Jeliot*, pero a su vez contar con un actor principal que cambie el estado del proceso; así como también representar las variables de una forma más concreta, en forma de contenedores. La aplicación *Jeliot* fue de gran inspiración por su forma de manejar los entornos de variables al anidar una colección de rectángulos; para el proyecto actual, se eligió ir más allá y representar los entornos de variables de una forma más concreta: como plataformas en 3D que contendrán tanto a actores como a contenedores de variables.

Adicionalmente, se integró un actor principal a la aplicación, como en las aplicaciones de *Greenfoot* y *Turtle Graphics*. A diferencia de los otros actores, que obedecen a comandos limitados como caminar o girar, este nuevo actor efectúa acciones de forma indirecta a las sentencias del usuario. Por ejemplo, al asignar un valor a alguna variable, el actor realiza el movimiento del valor hacia el contenedor y lo deposita dentro de éste.

La importancia de las múltiples vistas, las detalladas y las simples, permiten observar al mismo algoritmo desde otras perspectivas. Como demuestra la aplicación de *Hypermedia Visualization System (HalVis)*, el cambiar de un punto de vista detallado a uno general puede ofrecer ventajas para ciertos estudiantes. Se eligió incluir dos vistas en la aplicación, la principal que contiene a las metáforas antes descritas, y otra más avanzada que muestra el árbol de la sintaxis, junto a su recorrido y evaluación, nodo a nodo.

Finalmente, el diseño combinó las técnicas de diversas aplicaciones, como lo fueron el enfoque de actor, los conceptos más abstractos de *Jeliot*, los gráficos llamativos de *Alice*, las múltiples vistas de *HalVis*, y el editor textual de código utilizado en las aplicaciones de nivel intermedio como *Greenfoot* y *Jeliot*.

3.3.3 Evaluación de herramientas de desarrollo

Para desarrollar una aplicación web fue necesario contar con un servicio de alojamiento en línea, así como con un nombre de dominio. Se comenzó revisando los servicios de alojamiento web que son gratuitos y más populares. Entre los servicios gratuitos se evaluó a 000WebHost, sin embargo el servicio gratuito resultó muy inestable y poco confiable. Se procedió a probar la tecnología de *Google App Engine*, y demostró ser una solución muy robusta. El servicio básico incluyó la creación de un subdominio. *Google App Engine* permitió desarrollar en *Python*, *Java* y *Go*. Se eligió el lenguaje *Python* por contar con las librerías necesarias para la generación de analizadores léxicos y sintácticos.

Se seleccionó el editor de código fuente para web *CodeMirror*, ya que contó con la funcionalidad requerida para resaltar las palabras clave, por líneas o rangos, y permitió colocar un marcador en el número de línea actual de una forma completamente programable. Además, posee una extensa documentación en línea, y se contaba ya con experiencia previa en su utilización.

Para la generación de gráficas 3D en el navegador, se indagó sobre las herramientas más populares, evaluando librerías como *GLGE*, *Three.js* y *Angle*. La búsqueda fue enfocada en una solución sin *plug-ins* para facilitar el acceso; finalmente, se optó por utilizar la librería de código abierto *Three.js*, debido a su rendimiento, facilidad de uso y amplia documentación en línea. Por consiguiente, fue requerida también una aplicación de diseño 3D, por lo que se evaluaron las aplicaciones de *3dStudioMax* y *Blender*. Se optó por utilizar la aplicación *Blender*, ya que no tiene costo alguno y proporciona una gran funcionalidad, además de contar con documentación y comunidades de ayuda en línea.

Para la realización de diversas animaciones dinámicas que sólo pudieron diseñarse durante la ejecución del programa, se eligió a la librería *Tween.js*, ya que ofrece una gran variedad de funciones de suavizado para la interpolación de valores, permitiendo realizar

animaciones con diversos comportamientos de aceleración y desaceleración para el recorrido de los actores y las transformaciones de los modelos. La librería es utilizada comúnmente en conjunto con *three.js*, por lo que existe una amplia documentación sobre el uso de las dos herramientas en conjunto, siendo un factor muy importante para su elección.

3.3.4 Desarrollo del analizador léxico

Comenzando por el analizador léxico, se investigó su propósito en el contexto de un compilador y las técnicas necesarias para su desarrollo. El analizador léxico, o *lexer*, reconoce patrones de caracteres al examinar todo el flujo del código y es implementado por medio de una máquina de estados. El analizador consume un carácter a la vez, y al detectar un patrón válido, o *token*, emite una estructura de datos para representarlo.

El analizador léxico del proyecto (ver apéndice B.1) fue desarrollado utilizando la librería *PLY*, *Python Lex Yacc*, que facilitó el desarrollo y evitó errores de programación al no ser necesario implementar de forma manual la máquina de estado y sus transiciones. Para crear el *lexer* se requirió definir un conjunto de símbolos, o *tokens*, a través de expresiones regulares (Figura 13). En este caso se definieron los *tokens* más utilizados en el lenguaje C, como los valores literales numéricos, caracteres, operadores aritméticos, relaciones y lógicos; además, fue necesario definir reglas especiales de precedencia para los operadores, reglas para el manejo de errores y otras más para los caracteres en blanco y saltos de línea para llevar un registro del número de línea y posición del token correspondiente.

Una vez definido el léxico o palabras del lenguaje, la librería *PLY* cuenta con las clases y métodos necesarios para utilizar las funcionalidades del analizador recién definido (Figura 14). Al momento de crear una nueva instancia del analizador léxico, es posible pasar al constructor el modulo en el que se encuentran las definiciones de los *tokens*; opcionalmente, los *tokens* pueden ser definidos en el mismo archivo. Después, se inicializó el analizador llamando al método *input*, este recibió como parámetro el flujo de caracteres. Una vez que se terminó la construcción e inicialización del analizador, fue posible llamar al método *token*. Este método devuelve el *token* que fue detectado de acuerdo a las reglas y prioridad definidas previamente.

```

63
64 def t_IDENTIFIER(t):
65     r'[A-Za-z][0-9A-Za-z_]*'
66     if t.value in reserved:
67         t.type = t.value.upper()
68     return t
69
70 def t_NUMBER(t):
71     r'--[0-9]+(\.[0-9]*)?'
72     return t
73
74 def t_STRING(t):
75     r'"([^\\"|\\.)*)"'
76     return t
77

```

Figura 13. Definición del analizador léxico.

```

9 import ctokens
10 import cgrammar
11
12 def lexer(text):
13     cllexer = lex.lex(module=ctokens)
14     cllexer.input(text)
15     result = []
16     tok = cllexer.token()
17     while tok != None:
18         result.append(tok.value)
19         tok = cllexer.token()
20     return result
21

```

Figura 14. Uso del analizador léxico.

La detección del *token* siguió el principio de *Maximum Munch*. Esto significa que el analizador devuelve el *token* con el mayor número de caracteres que sea posible por las reglas del léxico definido. Por ejemplo, si en el flujo de caracteres aparece el número flotante 3.1416, el analizador puede consumir 3.1 y detectarlo como un flotante; sin embargo, es indispensable que consuma todos los caracteres posibles mientras la regla del número flotante siga siendo válida; de lo contrario, sólo sería posible detectar patrones de longitud fija. Al llegar al final del flujo de caracteres, el método *token* devolvió el valor *None*, indicando que ha terminado de analizar el texto.

Existe también la posibilidad de definir reglas para el manejo de errores. Si el flujo de caracteres contiene patrones no aceptados por las reglas del léxico, fue posible tomar acciones como mostrar mensajes o ignorar los caracteres correspondientes. Además, se cuenta con la funcionalidad de los estados del analizador; éstos fueron utilizados para el manejo de los comentarios definiendo un estado adicional *comment*. Al encontrar la secuencia de inicio de un comentario en el lenguaje C, como *//* o */**, el analizador cambió al estado *comment*; durante este modo los caracteres leídos fueron ignorados hasta que se encontró el cierre de comentario apropiado, ya sea el carácter de nueva línea *\n* o **/*.

La inclusión de archivos de cabecera fue implementada como un comentario, esto debido a que por el momento la funcionalidad de librerías externas reales no estaba contemplada dentro de las limitaciones del proyecto.

3.3.5 Desarrollo del analizador sintáctico

Se continuó con la indagación del propósito y funcionamiento del analizador sintáctico o *lexer*. Como se mencionó en el capítulo anterior, la función del *lexer* fue detectar la estructura de un programa; esto es, la posición y el tipo de cada uno de los *tokens* en el programa. Debido a que una gramática de un lenguaje de programación, como el lenguaje C, puede tener estructuras de un lenguaje no regular, como los paréntesis balanceados, es necesario un mecanismo que vaya más allá de las máquinas de estados finitos, esto es, que tenga memoria.

Así como el analizador léxico identificó los fragmentos del texto significativos, tales como nombres de variables, o palabras clave, el analizador sintáctico permitió detectar las estructuras válidas formadas con estas palabras, es decir, las sentencias válidas del lenguaje. La estructura de estas sentencias se definió a través de una gramática libre de contexto; este tipo de gramáticas expresan aún más patrones que las expresiones regulares. Utilizando nuevamente la librería *PLY*, se definieron las sentencias del lenguaje a través de las reglas de reescritura, o producción, que indican los patrones requeridos para reemplazar una secuencia de tokens por un símbolo no terminal.

La definición de la gramática libre de contexto se realizó mediante un conjunto de definiciones recursivas (Figura 15). Las reglas de reescritura se codificaron en una cadena de caracteres que ocupa la primera línea de la función correspondiente, aprovechando así una funcionalidad de *Python* para realizar documentación. Las reglas se definen en dos partes separadas por los dos puntos; del lado izquierdo tenemos la regla a ser expandida, y del lado derecho el patrón que la representa. El patrón consta de terminales y no terminales, los terminales o *tokens* son representados por caracteres en mayúsculas. Esta forma de aplicar una gramática se conoce como análisis recursivo descendente, ya que se realiza una evaluación recursiva entre las funciones definidas. Cuando se identifica un patrón válido, fue posible regresar cualquier estructura de datos. En este caso se optó por representar cada patrón como un nodo de un árbol, definiendo una clase, y de esta forma se continuó formando el árbol durante el recorrido recursivo.

Una vez definido el analizador sintáctico, (ver apéndice B.2), fue necesario crear una instancia de su clase representativa (Figura 16); el constructor requirió como parámetro el nombre del módulo en el cual se definieron las reglas de la gramática; opcionalmente,

también es posible definir las reglas en el mismo archivo, como sucede con el analizador léxico. Antes de comenzar a utilizar el *parser*, fue preciso haber instanciado un *lexer*. Si se planea utilizar un *lexer* en conjunto con un *parser*, no es necesario inicializarlo con la entrada. A continuación, para utilizar el *parser* se procedió a llamar al método *parse*, pasando como parámetros el flujo de caracteres a analizar, y el *lexer* (Figura 16).

```

140 def p_stmt_if(p):
141     'estmt : IF exp stmt_or_compound optsemi'
142     p[0] = Tree(Node('if-then'),
143                 p[2], p[3])
144
145 def p_stmt_while(p):
146     'estmt : WHILE exp compoundstmt optsemi'
147     p[0] = Tree(Node('while'),
148                 p[2], p[3])
149
150 def p_stmt_declaration(p):
151     'estmt : type identifier EQUAL exp optsemi'
152     p[0] = Tree(Node('declaration', span(p,
153

```

Figura 15. Definición del analizador sintáctico.

```

23 def drawtree(text):
24     global count
25     count = 0
26     text = '\n'.join(text.splitlines())
27     cparser = yacc.yacc(module=cgrammar, write_tables=0)
28     clexer = lex.lex(module=ctokens)
29     parse_tree = cparser.parse(text, lexer=clex)
30     draw_tree = buchheim_layout(parse_tree)

```

Figura 16. Uso del analizador sintáctico.

El analizador sintáctico utilizó, de forma interna, el método *token* del analizador léxico cada vez que lo necesitó, de forma automática. Al terminar el análisis sintáctico, se produjo la estructura de datos definida en la regla más general; si esta contiene reglas no terminales, como en este caso, es posible construir de esta forma el árbol de la sintaxis. Por lo tanto, el resultado final del análisis léxico y sintáctico, es un árbol que representa una estructura válida del lenguaje definido. Si el flujo de caracteres no cumple con la gramática que se definió, no es posible construir el árbol.

El análisis sintáctico consta de operaciones de expansión y reducción de términos, y consumo de los terminales. El proceso de reducción es implementado por la librería a través del método LALR. Se utilizó la opción de depuración que genera las tablas de análisis para encontrar errores en las reglas. La gramática definida fue un subconjunto del lenguaje C, para evitar entrar en todos los detalles sutiles de la gramática ANSI C, y para mejorar el rendimiento del análisis.

Finalmente, fue preciso definir un conjunto de reglas especiales para alterar la precedencia de los operadores y el manejo de errores. De igual manera, fue indispensable calcular la posición de los caracteres *ASCII* originales y asignarlos en cada parte del árbol,

para de esta forma llevar un registro y resaltar en el editor de texto los caracteres originales que fueron procesados.

3.3.6 Desarrollo del intérprete

Mientras que el análisis léxico y sintáctico que conforman al *frontend* fueron realizados para el servidor, el intérprete o *backend* fue desarrollado para la ejecución del cliente. La decisión se justifica por la necesidad de tener disponibles, en todo momento, los valores de las variables actuales, el ámbito de la función actual y por la ventaja de ejecutar paso a paso el algoritmo sin requerir ninguna comunicación adicional con el servidor.

Para el primer diseño del intérprete se planteó utilizar el método de evaluación y aplicación recursiva, ya que éste aprovechaba las llamadas a función para recorrer de forma recursiva el árbol y, de esta forma, almacenar implícitamente la información de la evaluación en cada llamada recursiva, entre otras ventajas. Sin embargo, al utilizar implícitamente las funciones del lenguaje interpretador este método dificultaba el reproducir la ejecución paso a paso, por lo que fue necesario simular más a una máquina real; esto creando una lista de instrucciones, un apuntador a la instrucción actual, un sistema de almacenamiento de variables y una pila de entornos de función.

El recorrido por el árbol de la sintaxis fue modificado del método recursivo a un método iterativo (Figura 17). Para esta forma iterativa, se utilizó una pila, para almacenar los nodos a los que fue necesario volver en los recorridos de profundidad o anchura, también se crearon diccionarios para actualizar los entornos de variables cada vez que se interpretaba una llamada o retorno de función. Se optó por dirigir el recorrido mediante un ritmo, esto ya que la interpretación será apreciada por un ser humano y no una máquina. El ritmo mínimo es de 0.5 segundos, y puede multiplicarse para alentar el proceso. Cada vez que pasa el tiempo requerido el intérprete avanza un paso y recorre el siguiente nodo del árbol para evaluarlo e interpretarlo. Para habilitar la funcionalidad de pausa, se optó por detener el temporizador que llama a la función en el próximo paso.

La estructura del árbol se optimizó para la transferencia del servidor al cliente, reduciendo el tamaño al acotar el nodo a sus componentes necesarios para la interpretación y codificando toda la estructura a formato *JSON* (Figura 18). Cada nodo es un objeto literal

y cuenta con la propiedad *children*, la cual es una lista de los hijos correspondientes al siguiente nivel en el árbol. Además, cada nodo cuenta con un identificador, índices de desplazamiento en el flujo de caracteres, y un par de coordenadas que fueron calculadas en el servidor después del análisis sintáctico, pero antes de comenzar la interpretación.

```
78 var breadthWalk = function(ast) {
79   var result = [];
80   var nodes = [];
81   nodes.push(ast);
82   while (nodes.length > 0) {
83     var cur = nodes.pop();
84     var lchild = cur.children[0];
85     var rchild = cur.children[1];
86     if (lchild && lchild.node.type) {
87       result.push(lchild);
88     }
89     if (rchild) {
90       nodes.push(rchild);
91     }
92   }
93   return result;
94 };
```

Figura 17. Recorrido iterativo del árbol sintáctico.

```
1  {
2    "node": {
3      "span": [24, 28],
4      "type": "identifier",
5      "value": null
6    },
7    "children": [{
8      "node": {
9        "span": [24, 28],
10       "type": "main",
11       "value": null
12     },
13     "children": [],
14     "id": "6",
15     "x": 1.0,
16     "y": 3
17   }]
18   "id": "5",
19   "x": 1.0,
20   "y": 2.0
21 }
```

Figura 18. Codificación JSON de los nodos.

El intérprete desarrollado, (ver apéndice B.3), recorrió el árbol de la sintaxis, paso a paso y a la velocidad indicada evaluando cada nodo en ese mismo instante. En cualquier momento, se contó con acceso a los valores de las variables locales y globales, así como a los entornos de función. El intérprete reconoció las instrucciones básicas como las operaciones aritméticas y lógicas, la declaración de variables y funciones, las estructuras de control condicionales y repetitivas, entre otras. Dependiendo del nodo reconocido, se llevó a cabo una operación equivalente en el lenguaje interpretador, en este caso se simulaban las llamadas a función utilizando una pila propia para llevar el registro de las llamadas, además de consultar en mapas del entorno el valor de los identificadores.

El entorno global es inicializado antes de la interpretación y cada nuevo entorno que fue creado al interpretar una llamada a función tiene como propiedad de *parent* una referencia al entorno anterior, comenzando por el entorno global. Si un identificador no se encuentra en el entorno actual se comienza una búsqueda de su valor en el entorno padre de este, y si no se llega a encontrar, se continua con el padre del padre, así recursivamente hasta llegar al entorno global.

3.3.7 Diseño de metáforas y animaciones 3D

Las metáforas creadas son análogas a los conceptos fundamentales de un lenguaje de programación estructurada, como la declaración y la asignación de variables, las operaciones básicas aritméticas y lógicas, las estructuras de control selectivas e iterativas y las funciones.

Las variables se representaron como contenedores tridimensionales que incluyen el nombre de la misma escrito en las cubiertas; estos contenedores pueden abrirse (Figura 19) o cerrarse (Figura 20) por una tapa en la parte superior, haciendo alusión a que los valores son guardados dentro de éstos.

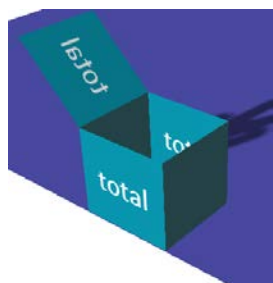


Figura 19. Contenedor de variable abierto.



Figura 20. Contenedor de variable cerrado.

Los tipos de datos se representaron utilizando diferentes colores en los contenedores de variables (Figura 21), ya que permitieron diferenciar de una forma rápida unos de los otros y, con el tiempo ayudan a familiarizarse con el concepto de tipo de datos en programación.

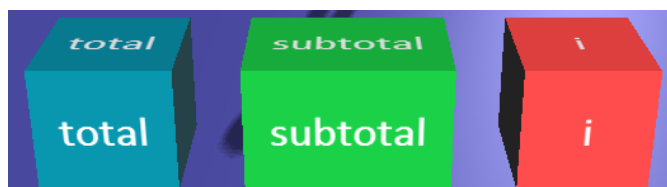


Figura 21. Tipos de datos.

Las plataformas fueron la metáfora elegida para representar a un entorno de variables (Figura 22); cada vez que se llama a una función se creó un nuevo entorno y, por lo tanto, una nueva plataforma correspondiente. Cada plataforma contó con un letrero que

indicó para que llamada de función y para que parámetros se creó este entorno, la parte frontal es el área de las variables en donde aparecerán los contenedores, la parte central superior es el área de expresiones la cual muestra paso a paso la evaluación de cada expresión y, finalmente, en la parte posterior se encuentran los contenedores especiales: una caja de literales que representa una fuente inagotable de valores literales utilizados en las expresiones, una caja de salida, y otra de entrada que simularon la interacción con el usuario.

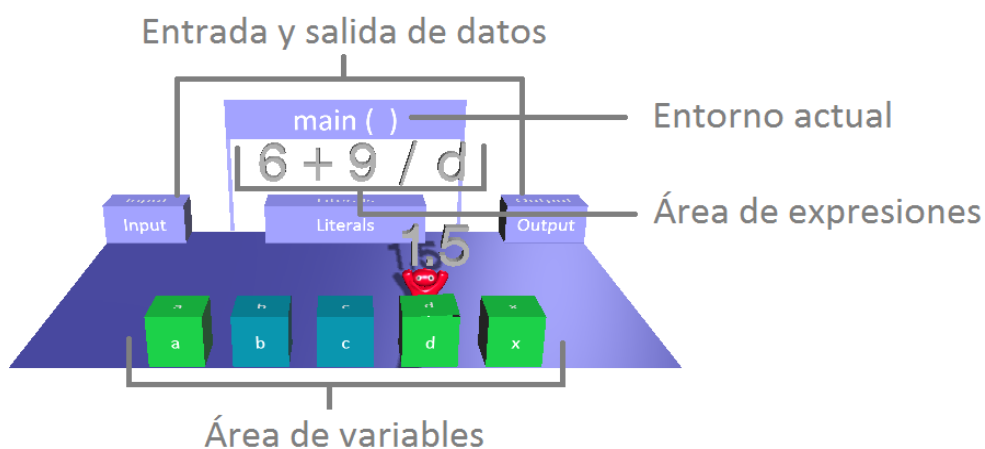


Figura 22. La plataforma y sus elementos.

Cuando una función recibe parámetros o se declara alguna variable, se creó un nuevo contenedor de variable en la plataforma actual; y cuando se regresa de una función esta plataforma y sus contenedores fueron destruidos, simbolizando la liberación de memoria al regresar de una función. La decisión de incluir una entrada y salida de datos fue debido a que éstos son conceptos muy importantes en la teoría de la computación; al contar con entradas y salidas fue posible considerar a un algoritmo como una caja negra que recibe una entrada de datos, realiza un proceso y genera una salida de datos. De esta forma se diferencia el mundo virtual en el visor de gráficas 3D del mundo real, en el cual el usuario puede interactuar fuera del visor de gráficas con los controles de interfaz gráficos.

A pesar de que el analizador léxico no reconoce los archivos de cabecera contenidos en el código, sino que los trata simplemente como a un comentario, fueron implementadas las funciones de la librería estándar de C, *printf* y *scanf*. La sintaxis es muy similar y ofrecen una funcionalidad básica con respecto a sus correspondientes versiones oficiales,

como la cadena de formato en el primer argumento. Sin embargo, se trató de una simulación muy básica y primitiva que, para los propósitos de la aplicación, cumplieron su funcionalidad satisfactoriamente. De esta forma, se pudo interactuar con el intérprete al recibir la entrada del usuario y almacenar la salida; esto es muy importante para lograr apreciar mejor el comportamiento del algoritmo, depurar ciertos errores y, en general, comprender mejor los programas.

El actor principal, llamado *Runaround* (Figura 23), es un personaje antropomórfico que recorre y salta entre las diferentes plataformas de la aplicación. Este personaje representa al espíritu de la máquina que sigue las instrucciones ciegamente; dependiendo de la sentencia que está siendo interpretada, el actor realiza diferentes acciones. En el área especial de evaluación (Figura 24) que se localiza en el centro de cada plataforma, se pueden visualizar las expresiones y como, paso a paso, se fueron construyendo, evaluando y reduciendo. Cuando se realiza una llamada a función, la evaluación del término requerido se pospone hasta que se retorne el valor requerido.



Figura 23. Actor Runaround.

`int area = 17500`

Figura 24. Área de expresiones.

Cuando se llama a una función, se crea una nueva plataforma que representa a un nuevo entorno de variables (Figura 25); luego, el actor saltará hacia el nuevo entorno, dejando pendiente la evaluación del entorno anterior hasta que éste regrese con algún valor. Cuando se retorna de una función, el personaje toma en sus manos el valor de retorno y salta hacia el entorno anterior, justo antes de que el nuevo entorno sea destruido, representando la pérdida de las variables una vez que terminó la función.

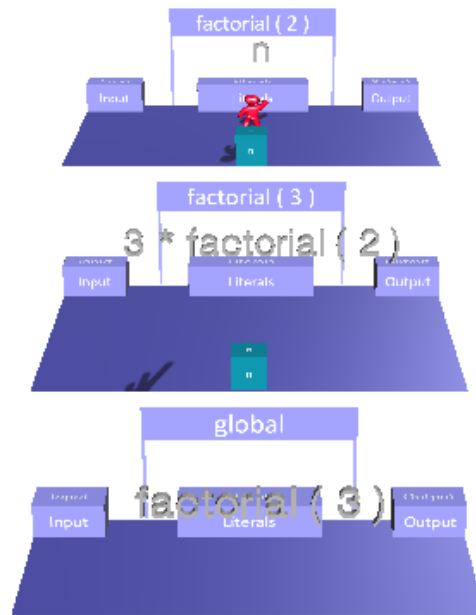


Figura 25. Generación de plataformas por cada llamada a función.

En el caso de una declaración de variable, el actor corre hacia el contenedor de literales, lo abre y toma el valor que necesitó, cargándolo con sus manos sobre su cabeza; después, un contenedor con el nombre de la variable declarada aparece en la plataforma y el actor corre hacia éste, cargando la literal, para finalmente depositar este valor en el contenedor recién creado.

Cuando en una evaluación de alguna expresión que incluya variables se necesitó consultar el valor actual de alguna de éstas, el actor corrió a encontrar el contenedor que representa a la variable, lo abrió, tomó el valor correspondiente y, después, corrió al centro de la plataforma para reemplazar el nombre de la variable en la expresión por el valor que tuvo en sus manos. Si se trata de una variable global o de otro entorno, el actor necesita buscarla entre las plataformas, por lo que salta hacia la próxima, y si no lo encuentra salta a la siguiente, y así sucesivamente hasta que encuentre el contenedor indicado; una vez encontrado, toma el valor en sus manos y recorre el mismo camino, en reversa, para reemplazar en la expresión la variable con su valor actual.

La aplicación más influyente para el diseño del proyecto fue *Jeliot*, siendo la principal fuente de inspiración, ya que los conceptos de nivel intermedio cumplen con los

objetivos. También se combinó la metáfora de actor de *Greenfoot*, junto a las gráficas 3D de *Alice* y además se diseñaron nuevas metáforas.

El concepto de constantes o literales de *Jeliot* fue muy importante, ya que al utilizar literales, resultó sencillo diferenciarlas de una variable. Las literales tienen un valor constante y básicamente sólo representan un valor; no son variables que puedan ser asignadas. Utilizando la metáfora de los contenedores para las variables, se logran diferenciar fácilmente de los valores literales, que no tienen apariencia de cajas.

El editor de código fuente también fue animado (Figura 26), ya que durante cada paso fue preciso detectar en qué posición del flujo de caracteres se encontraba la expresión actual y luego se resaltaba el texto correspondiente. Utilizando un triángulo gris en la parte izquierda del editor, se indicaba el número de línea actual, además, la línea fue coloreada con un tono gris. Durante la interpretación y recorrido del árbol de la sintaxis, cada nodo contaba ya con la información de posición de renglón y columna que fue calculada durante el análisis léxico y sintáctico para lograr resaltar las expresiones y sub expresiones en el editor de código fuente; este resaltado principal fue de un color amarillo, simulando a un marca textos, para diferenciar rápidamente cual fue la expresión que estuvo siendo evaluada.

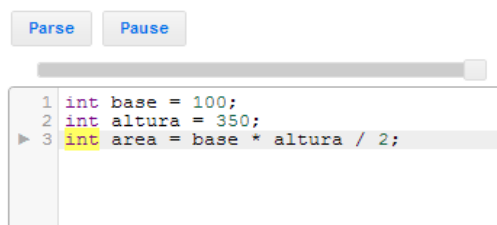


Figura 26. Editor de código fuente.

Fue primordial recordar la importancia de tener múltiples vistas, ya sean las detalladas y abstractas, o las más simples y concretas, para contar con más perspectivas que pudieran ser de apoyo para algunos estudiantes. Se incluyó una vista adicional a la principal: la vista de árbol (Figura 27); en esta vista fue posible observar de una forma detallada el resultado de los análisis léxico y sintáctico. La animación indica cual es el nodo que está activo mostrándolo con un color diferente en ese momento, además se observa su valor actual. En cada paso del recorrido, la cámara se traslada al siguiente nodo a ser

evaluado, esta transición mostró el orden de evaluación de las sentencias y expresiones, la estructura que el analizador sintáctico generó a partir del código, así como las acciones del intérprete para cada uno de los tipos del nodo. Fue muy ilustrativo codificar y observar la animación del árbol, ya que aunque se trató de un punto de vista muy abstracto, resumió de una forma muy simple la interpretación de un programa.

El dibujado del árbol (Figura 27) se llevó a cabo mediante el algoritmo de Buchheim que se rige por 5 principios: 1) las aristas del árbol no deben tocarse, 2) todos los nodos de la misma profundidad deben ser dibujados en la misma línea horizontal, 3) Los árboles deben ser dibujados lo más angosto posible, 4) Un padre debe estar centrado sobre sus hijos, y 5) un subárbol debe ser dibujado con la misma forma sin importar en cual posición del árbol se encuentre. Se utilizó una luz ambiental y una direccional que siguió al nodo activo, iluminando el área visible.

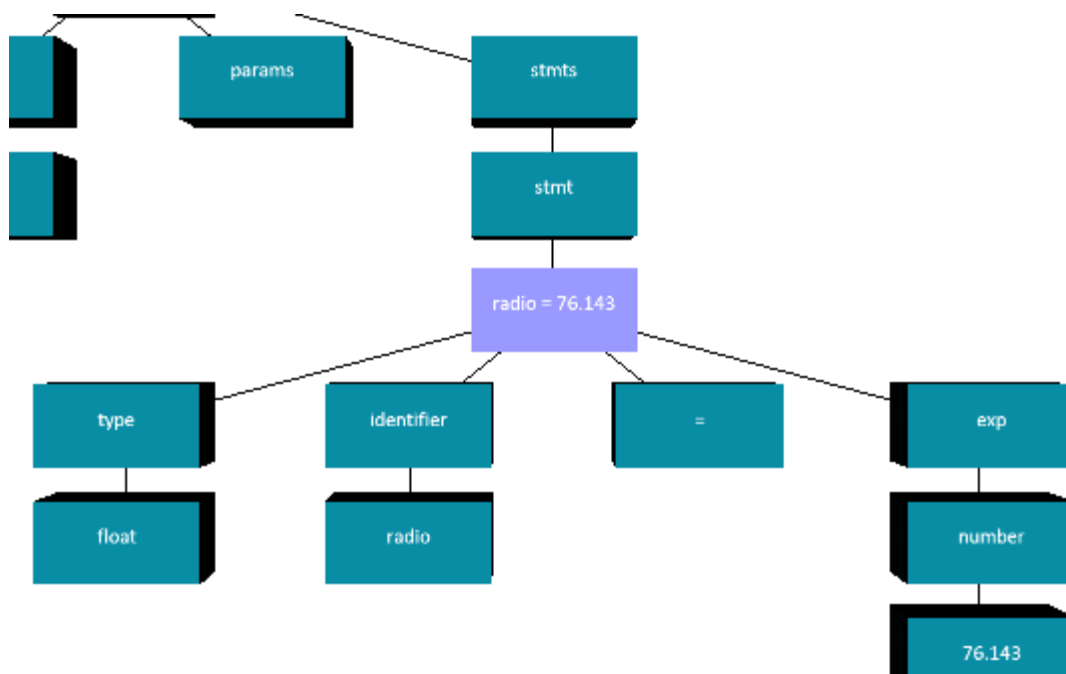


Figura 27. Representación y recorrido del árbol de la sintaxis.

El modelado de los personajes y objetos fue realizado utilizando el software de diseño 3D *Blender*, mediante la técnica de modelado por mallas. Una malla fue definida por un conjunto de vértices relacionados entre sí, en esta técnica se comienza el diseño con un modelo simple, una primitiva como un cubo y, posteriormente, se procedió a subdividir el

objeto en partes más pequeñas, para manipular cada una de estas secciones por separado mediante la escala, rotación y traslación. A través de este proceso de diseño se fue construyendo el modelo hasta llegar a un nivel de detalle aceptable.

Cuando se construyó el modelo en un ambiente tridimensional, fue necesario verificar que los vértices seleccionados realmente correspondieran con los vértices deseados, observando el modelo desde diferentes perspectivas, ya que, algunas vistas pueden crear una ilusión óptica. Para ahorrar tiempo y esfuerzo, se utilizaron técnicas como el diseño simétrico, en el cual solo fue necesario diseñar una parte del modelo y mediante la herramienta de espejo fue posible generar el resto.

Para la animación del actor, se procedió a utilizar la técnica de animación a través de esqueletos (Figura 28). Se comenzó definiendo esqueletos en el interior de la malla; éste se conformó de varios huesos independientes que a su vez se componen de una raíz y punta. Fue necesario asociar el esqueleto a la malla para permitir rotar y trasladar cada uno de los huesos y lograr deformar el modelo. La malla fue subdividida en grupos de vértices, de esta forma cada hueso solo deformó a su grupo correspondiente de vértices y se evitó una deformación excesiva en el modelo principal. Una vez definidos los huesos y sus grupos de vértices correspondientes, se pudo animar al modelo como si fuera un muñeco (Figura 29).

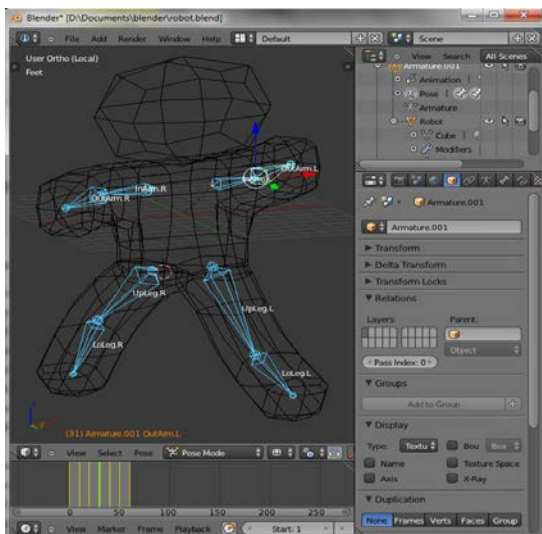


Figura 28. Animación de correr.

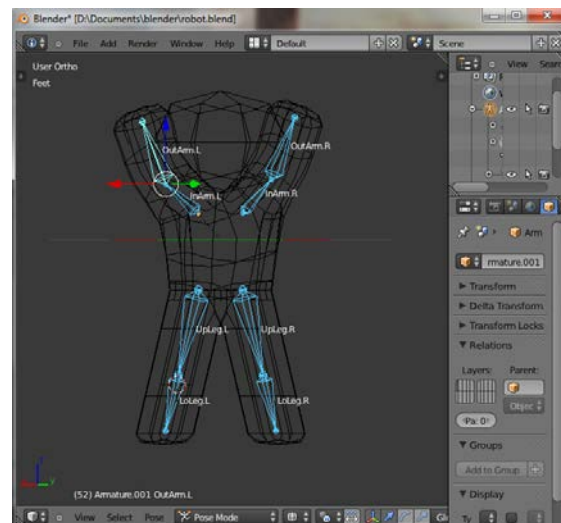


Figura 29. Animación de sujetar.

Utilizando cinemática inversa, fue posible definir la posición del último hueso en la cadena, todos los demás huesos asumieron una posición automáticamente, calculada por el motor de cinemática inversa, para mantener la fluidez del movimiento y evitar movimientos bruscos. El motor resolvió matemáticamente la cadena de posiciones, permitiendo un posicionamiento más sencillo y preciso.

Después de definir el esqueleto y los grupos de vértices en la malla, se procedió a definir diferentes poses en cuadros de animación clave. Al crear una pose, los huesos se comportaron de forma rígida, y al girar o trasladarse, la malla se deformó. Finalmente, al terminar cada pose se guardaron las posiciones, rotaciones y escala de todos los huesos para cada pose. La animación en este caso consta de 24 cuadros en total, de los cuales sólo 5 se definieron como cuadros claves. Al guardar estas 5 poses el motor de Cinemática Inversa calculó los valores intermedios necesarios para producir una animación fluida.

Los modelos animados fueron exportados a formato *JSON* para una carga más eficiente en la plataforma web, sin embargo, se decidió reducir el número de cuadros de 60 a 20, lo cual disminuyó considerablemente el tamaño del archivo. El modelo del actor fue el archivo que más memoria ocupó en la aplicación, con un tamaño de 3.76 MB y una textura de 114KB. El tiempo de carga inicial de la aplicación resultó afectado principalmente por el modelo del actor, ya que los demás elementos constan de figuras primitivas que se generan dinámicamente.

Para finalizar, fue necesario integrar todos los modelos y animaciones creando una escena, de forma muy similar a la escena de una película, gracias a las abstracciones de la librería *three.js*. Comenzando con un objeto escena, fue necesario agregar a esta una serie de elementos mínimos como una cámara, luces ambientales y luces direccionales, luego los modelos, las texturas y especificar los parámetros de brillo y sombras. La iluminación constó de 2 luces de tipo direccional, una en la parte trasera de la plataforma y la otra en frente. Cada objeto en la escena consta de una geometría y un material: la geometría es el conjunto de los vértices con sus posiciones correspondientes, y el material indica la superficie entre los vértices, tipo de brillo y sombreado, así como un mapa de textura.

Las texturas con texto para los contenedores de variables fueron generadas dinámicamente utilizando el elemento *canvas* de *HTML5*. Este elemento del *DOM* permitió generar un mapa de bits que pudo ser utilizado como textura para un material. Calculando

el tamaño del texto a desplegar, se generó una textura con el ancho requerido de forma completamente dinámica, por lo que no fue necesario almacenar ninguna imagen.

3.3.8 Integración de escenas e intérprete

Una vez realizadas las animaciones, (ver apéndice B.4), y el intérprete, fue necesario integrar todas las partes en un todo. Algunas animaciones resultaron ser dinámicas y no fue posible conocer de antemano su representación visual, por lo que fue necesario generarlas en tiempo de ejecución, como la creación de texturas para los nombres de variables declaradas, así como la posición de los elementos en el escenario.

La integración de las animaciones con el intérprete se llevó a cabo definiendo escenas para cada *token* que se evalúa durante el recorrido por el árbol de la sintaxis.

Una escena se conforma de múltiples actores efectuando una secuencia de comportamientos. Por ejemplo, en una declaración de variable (Figura 30), el actor *Robot* realiza el comportamiento de *grabLiteral()*, tomando en sus manos el valor indicado, luego efectúa *gotoBox()*, activando la animación de correr, hasta llegar al otro actor *Box* que reacciona con *open()*, debido al *openBox()* de *Robot*. Después, se ejecuta *dropLiteral()* y ahora *Box* responde con *close()*. Finalmente, el actor realiza la acción *gotoCenter()*, regresando al centro de plataforma, listo para la siguiente sentencia.

```
224     });
225     break;
226     case 'declaration':
227         var identifier = ast.children[1];
228         var type = ast.children[0].children[0].node.type;
229         var boxColor = getBoxColor(type);
230         var box = robot.scopePlatform.addBox(identifier, boxColor);
231         var declarationVal = identifier.node.value();
232         if (box !== undefined) {
233             robot.scopePlatform.expClear();
234             robot.grabLiteral(declarationVal);
235             robot.gotoBox(box, function() {
236                 robot.openBox(function() {
237                     robot.dropLiteral();
238                     robot.closeBox(function() {
239                         robot.gotoCenter(function() {
240                             delayedStep();
241                         });});});});});
242         }
243     break;
244     case 'while':
```

Figura 30. Escena declaración de variable.

Existen escenas, con actores y sus comportamientos, para la asignación y declaración de variables, las llamadas y retornos de funciones, las operaciones aritméticas o lógicas, las estructuras de control selectivas e iterativas, y las funciones *printf* y *scanf*.

Al crear las escenas fue necesario definir una secuencia de comportamientos activando la animación indicada y actualizando las posiciones de los actores. Las posiciones fueron calculadas utilizando la librería *tween.js* que implementó las ecuaciones de suavizado necesarias. Al interpolar los valores, se indicó la posición inicial y final del actor en el espacio y el tiempo deseado. Automáticamente, los valores intermedios fueron calculados a través de una interpolación de los valores dirigida por una ecuación de suavizado, que acelera o desacelera los cambios de los valores para crear transiciones fluidas y naturales. Por ejemplo, la animación que se reproduce al abrir un contenedor de variable (Figura 31) se realizó mediante una interpolación de los valores de la matriz de rotación en la tapa superior, tomando como eje de rotación un punto ubicado en la parte superior del mismo contenedor y utilizando una ecuación de suavizado de tipo elástica.

```
22 var BoxBehaviours = { Opening: 0, Open: 1, Closing: 2, Closed: 3 };
23 function Box(identifier, background, scopePlatform, minWidth, fontSize) {
24     this.open = function(callback) {
25         new TWEEN.Tween(lidRotator.rotation)
26             .to({
27                 z: Math.PI * 3/4,
28             }, stepWait * 2
29             ).easing(
30                 TWEEN.Easing.Elastic.Out
31             ).onStart(function() {
32                 self.behaviour = BoxBehaviours.Opening;
33             }).onComplete(function() {
34                 self.behaviour = BoxBehaviours.Open;
35                 if (callback !== undefined) {
36                     callback();
37                 }
38             }).start(animationElapsed);
39     }
40 }
```

Figura 31. Interpolación elástica utilizada para el comportamiento de un contenedor.

Las animaciones de desplazamiento del actor se definieron a través de interpolaciones de los valores de la matriz de posición y de rotación, con un suavizado que acelera en el inicio y frena al final. Cuando el actor recorre la plataforma de un punto a otro, fue necesario calcular el ángulo de rotación adecuado en base a la posición actual del

actor en el plano XZ y la posición final del contenedor, para lograr girar el modelo 3D y apuntarlo hacia el contenedor indicado, y finalmente iniciar una traslación hacia su objetivo, en el tiempo indicado y sin importar la distancia.

Durante el salto entre plataformas, el actor se desplaza en el plano YZ aplicando dos interpolaciones cuadráticas a su vector de posición que se asemejan a un tiro parabólico. Este tipo de animaciones basadas en interpolaciones fueron multiplicadas por la velocidad del ritmo empleado para recorrer al árbol; de esta forma, se vinculó a la velocidad de ejecución con la velocidad de las animaciones. Además, al pausar la ejecución, la animación se detiene completamente, siendo posible reanudarla desde la posición anterior.

En todo momento, fue importante considerar que la posición y orientación de la cámara fueran las indicadas, así como también ajustar la iluminación; en este caso se utilizaron una luz ambiental y dos luces direccionales, en el frente y detrás de la plataforma, que siguen al actor en todo momento. Se tomó la decisión de que la cámara siguiera al actor sólo cuando este realizara un salto entre las plataformas, esto para evitar perderlo de vista fácilmente.

Fue imprescindible el llevar una correspondencia con el árbol de la sintaxis, la evaluación de sus nodos y las animaciones, ya que una escena está constituida por una secuencia de comportamientos, por lo que al evaluar cada nodo del árbol de la sintaxis fue necesario en algunos casos reproducir varias animaciones por un solo *token*. Debido a la naturaleza asíncrona de las animaciones, se encadenaron los comportamientos en una secuencia a través de una lista de apuntadores a funciones o *callbacks*, por lo que el inicio de una animación dependió de que la otra terminara.

Las animaciones que son completamente dinámicas dependieron de múltiples variables, como la longitud del nombre de una variable, que provoca un ajuste en el ancho de un contenedor, o la posición de los elementos. Por ejemplo, fue necesario recalcular la posición de los contenedores de variables cada vez que una nueva variable era declarada, esto para que los contenedores de variables siempre estuvieran ubicados de forma centrada al frente de la plataforma. También es posible que la plataforma cambie de tamaño dinámicamente para poder alojar a la cantidad de variables necesaria y, gracias a que la iluminación sigue al actor siempre, es posible apreciar la escena sin importar si la plataforma es demasiado grande.

Capítulo 4. Resultados de la investigación

A continuación, en el presente capítulo se exponen los resultados obtenidos al llevar a cabo una encuesta a tres grupos de programación de la UACJ. En la encuesta se solicitó a los estudiantes su opinión con respecto al contenido, funcionalidad y diseño de la aplicación *Runaround*. Los datos arrojados por los instrumentos son de naturaleza cualitativa al tratarse de opiniones subjetivas de cada estudiante. El análisis de este tipo de datos se realizó a través de conteos para determinar la frecuencia de cada una de las respuestas, además fue necesario analizar cuidadosamente las preguntas abiertas y los comentarios o sugerencias de los estudiantes. Finalmente, se plantea una posible interpretación de estos resultados, sugiriendo qué características de la aplicación fueron exitosas entre los estudiantes y cuales necesitan de un nuevo enfoque.

4.1 Presentación de resultados

Se realizó una encuesta a alumnos y profesores (ver apéndice C) de la UACJ con el propósito de mostrar, de forma meramente ilustrativa, si el proyecto cumple con los objetivos propuestos inicialmente y, al mismo tiempo, considerar posibles mejoras o detectar problemas con la aplicación actual.

La encuesta se integró de 5 partes y las preguntas del instrumento son de tipo escala de Likert y abiertas, por lo que los datos a recolectar fueron en su mayoría de naturaleza cualitativa. La parte A se conformó por los datos generales del encuestado, como nombre, género y otros más específicos, si es estudiante o profesor; la parte B se integró por las preguntas de diagnóstico, estas intentan capturar la dificultad percibida por los estudiantes entre los distintos temas de la programación, si conocen las diferentes aplicaciones didácticas ya existentes, y su opinión sobre este tipo de herramientas.

Las partes C, D y E, constituyeron la evaluación de la aplicación *Runaround*. En la sección C se valoró el contenido de la aplicación con respecto a las metáforas utilizadas y a los temas de la programación representados, evaluando si la analogía percibida representó apropiadamente al concepto indicado. La sección D examinó la funcionalidad actual de la aplicación, revisando las opciones que se ofrecen al usuario y su utilidad. Y para terminar, la sección E evaluó el diseño de la aplicación, tomando en cuenta tanto el diseño de la

interfaz como el de las metáforas, para determinar si ayudó a cumplir el objetivo de la aplicación.

La primera encuesta fue aplicada el día martes 30 de abril del 2013 a las 11:00 horas en el centro de cómputo de IIT, salón W-102 (Figura 32). La clase encuestada correspondió al grupo B de programación II, perteneciente a la profesora Vianey Cruz; el grupo contó con la presencia de 9 estudiantes solamente.



Figura 32. Encuesta realizada al grupo B de programación II.

Para comenzar, fue presentado el objetivo de la aplicación, luego se distribuyeron las encuestas a los estudiantes, y se procedió inmediatamente a contestar la parte A de datos generales y la parte B de diagnóstico. Posteriormente, se continuó con una breve introducción a la interfaz, las metáforas y la operación de la aplicación, accesible en el sitio <http://irunaround.appspot.com>; después, los estudiantes ingresaron a la dirección web y utilizaron la aplicación durante un lapso de 15 minutos y, finalmente, se prosiguió con las secciones restantes de la encuesta.

El día 2 de Mayo del 2013 se realizaron las siguientes dos encuestas, en el centro de cómputo de IIT, de 10:00 a 11:00 con el grupo D de 16 estudiantes, y de 12:00 a 13:00 con el grupo E de 8 estudiantes; ambos de la profesora Patricia Parroquín. Se siguió el mismo procedimiento descrito para el grupo B, y se recolectaron los resultados. En total se encuestaron a 33 estudiantes de 3 grupos diferentes.

4.2 Análisis e interpretación de los resultados

Los resultados de las encuestas (ver apéndice D) fueron capturados y analizados cuidadosamente. Los datos obtenidos de las escalas tipo Likert se analizaron midiendo la frecuencia con la que los alumnos eligieron cada uno de los diferentes valores posibles. Realizando un conteo para cada uno de los valores, se compararon las frecuencias de las opciones seleccionadas para cada una de las preguntas; esta comparación fue realizada por cada grupo. Por último, se llevó a cabo un conteo total de las respuestas de todos los grupos.

Entre los datos globales, se determinaron los rangos de edades (Figura 33) y los semestres que se encuentran cursando los alumnos actualmente (Figura 34). Las edades predominantes son de 18 y 19 años, y el semestre predominante es el segundo, seguido del cuarto.

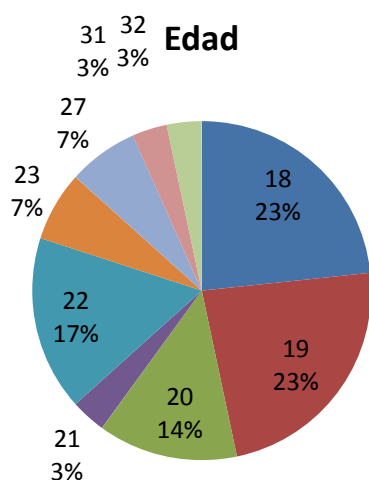


Figura 33. Porcentaje de edades.

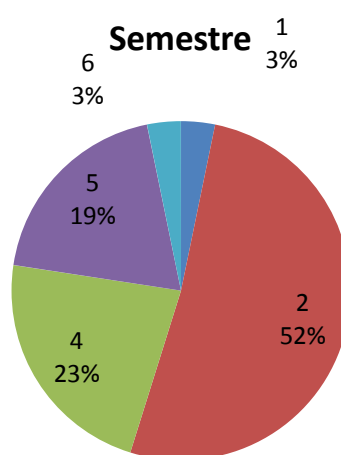


Figura 34. Porcentaje de semestres.

Comenzando con la sección B de diagnóstico, es ilustrativo observar que entre los temas de la programación que los estudiantes consideraron más difíciles, se encuentra el tema de los ciclos de repetición (Figura 35), seguido del tema de condiciones simples y múltiples. En cambio, los estudiantes se sienten más seguros con la precedencia de los operadores aritméticos y lógicos, así como también con los conceptos de variables globales y locales.

Dificultad en el tema de: Ciclos de repetición

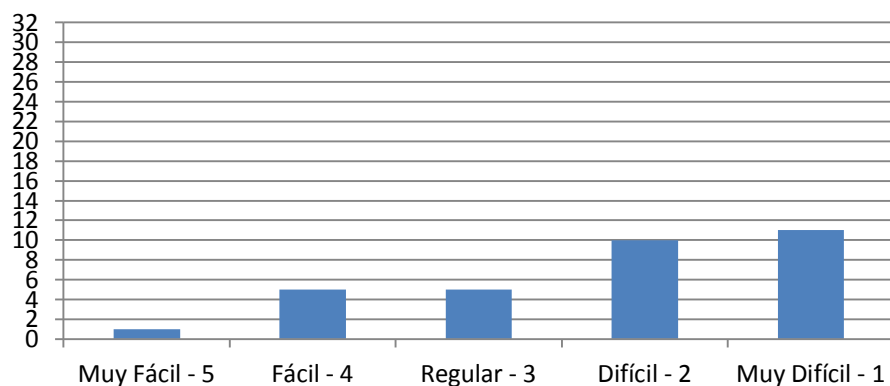


Figura 35. Total de respuestas: parte B pregunta 4.

Continuando con los resultados de la sección C, éstos sugieren que las metáforas utilizadas son fáciles de comprender por los estudiantes; en especial, la metáfora de las plataformas que representan el alcance global o local de las variables; así como también la animación de salto que realiza el actor entre las plataformas (Figura 36) y hace alusión a la llamada a una función y el retorno de los valores.

La animación de salto que realiza el personaje entre las plataformas, es una metáfora clara y sencilla de la llamada a una función y el retorno de los valores

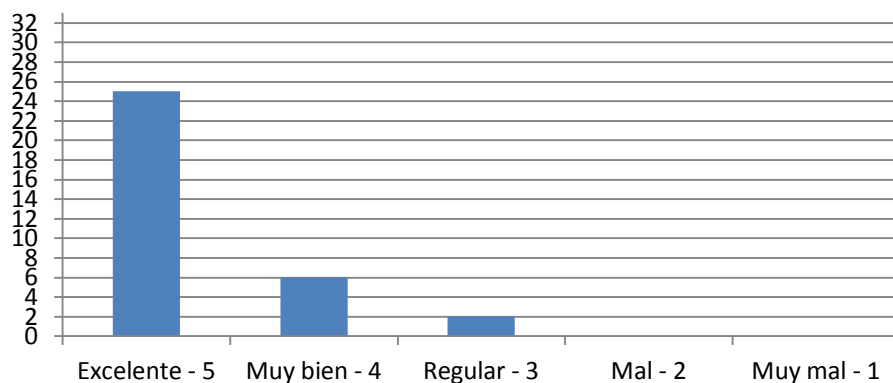


Figura 36. Total de respuestas: parte C pregunta 6.

Las encuesta también incluye opiniones positivas más personales: la estudiante Ileana comenta que los conceptos son representados de una forma intuitiva: "*porque en el programa se muestran las variables y todo muy gráfico y explícito.*", el estudiante Pedro también opina que: "*se da a entender facilmente, y los conceptos básicos son comprensibles, cumple con el objetivo principal.*", y finalmente, el estudiante Gustavo explica que: "*simplemente se ve claro, siguiendo el código de la izquierda y viendo las acciones del muñeco*".

Por otra parte, la vista de árbol no fue de gran aceptación entre los estudiantes (Figura 37), éstos señalaron como problemas la falta de control de la cámara en esta vista y el no ser posible reducir la velocidad de las transiciones. Además, la vista fue más avanzada y a los estudiantes les pareció confusa, ya que la mayoría no estaban familiarizados con la estructura de datos de árbol.

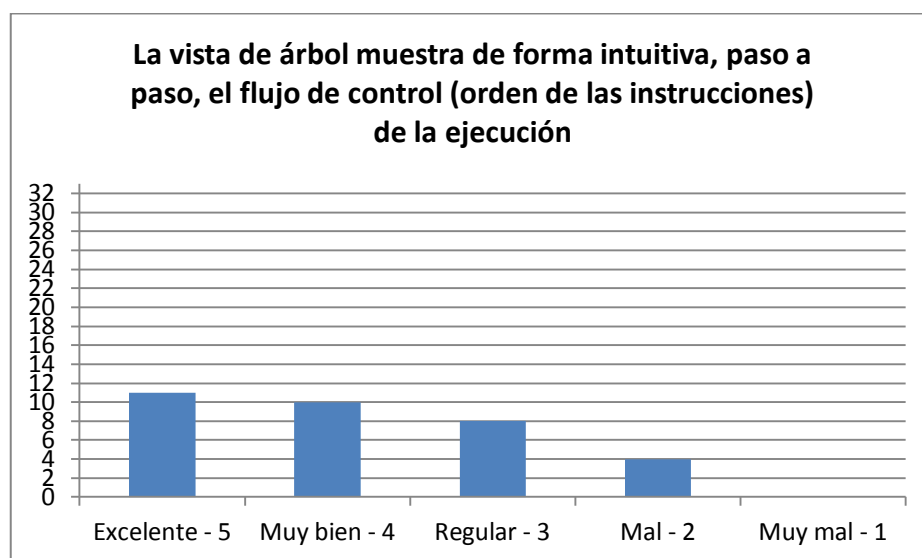


Figura 37. Total de respuestas: parte C pregunta 4.

Entre las opiniones personales referentes a la vista de árbol, la estudiante Norma sugiere que es necesario: "*ajustar la velocidad en la opción visual del árbol y ajustar el zoom en la opción visual del árbol*"; así mismo, el estudiante José recomienda que hace falta: "*una forma de explorar la totalidad del árbol*".

Durante las siguientes secciones D y E, y en el lapso de prueba con la aplicación, algunos estudiantes crearon programas inválidos para comprobar la respuesta del intérprete.

Entre los errores más comunes, el intérprete no reconoció la declaración de variables sin inicializar, cuando debería ser posible declararlas de esta forma tan utilizada. Además, la aplicación no reconoció la instrucción `while` vacía; sería ilustrativo visualizar al actor realizar trabajo de forma indefinida representando a un ciclo infinito. Así mismo, la mayoría de los estudiantes esperaban encontrar los temas de apuntadores y arreglos, al igual que los mensajes de error del compilador; sin embargo, debido a las limitaciones del proyecto actual estos temas y funcionalidades fueron pospuestos para una versión futura.

En conclusión, a través de los resultados se ilustra con cuales metáforas se relacionan mejor los alumnos, se señalaron errores y detalles importantes que hacen falta modificar, así como también se recolectaron sugerencias más detalladas para la mejora a futuro de la aplicación. Entre éstas sugerencias de nuevas características, el estudiante Antonio propone: *"que hubieran ejemplos y que incluyera los temas de arreglos y punteros"*; el estudiante Vicente sugiere que: *"podrían agregar un tutorial o una lista que indique que representan los colores"*, y finalmente, la profesora Patricia Parroquín recomienda que sería conveniente la funcionalidad de: *"mapas de memoria RAM, el muñequito podría ir a mapas de memoria, es decir tener abiertos 2 ambientes, tanto el actual (world) como el de memoria"*.

Capítulo 5. Discusiones, conclusiones y recomendaciones

Para finalizar, en este capítulo se examinan las discusiones de la problemática del proyecto una vez que éste fue concluido, analizando si se logró el objetivo inicial, así como si es posible esclarecer algunas de las interrogantes propias de la investigación; posteriormente, se presentan las conclusiones obtenidas al concluir el proyecto y, para terminar, se indican las recomendaciones para futuras investigaciones que nacen de las observaciones propias durante el desarrollo, así como de las sugerencias de estudiantes y profesores.

5.1 Discusiones

En los cursos iniciales de programación, se introduce al estudiante a una gran variedad de conceptos y términos nuevos; la mayoría de estos conceptos son de naturaleza abstracta, y su correcta comprensión puede conllevar dificultades.

La enseñanza de la programación estructurada a estudiantes con poca o ninguna experiencia involucra diversos retos educativos para los profesores. Por otra parte los estudiantes requieren recordar y asimilar una variedad de ideas nuevas, así como también manipular símbolos y reglas sin una analogía del mundo real evidente, lo cual dificulta al mismo tiempo el aprendizaje.

El objetivo del proyecto fue desarrollar una aplicación que apoye a los profesores en la enseñanza y a los estudiantes en el aprendizaje de los conceptos fundamentales de la programación estructurada a través de animaciones generadas a partir del código fuente.

Mediante la realización de las encuestas a estudiantes y profesores se obtuvieron resultados que correspondieron en su mayoría con el objetivo esperado, ilustrando la opinión positiva de los estudiantes para ciertas metáforas, así como críticas y recomendaciones para otras características.

Durante la indagación de los antecedentes se observó que aún falta mucho trabajo por hacer en la enseñanza y aprendizaje de la programación a través de aplicaciones y animaciones. Entre las animaciones de la aplicación actual, la vista del árbol de la sintaxis no obtuvo los resultados deseados, debido a su complejidad y funcionalidad limitada.

El proyecto actual se limitó a un análisis léxico y sintáctico solamente, optando por no realizar el análisis semántico. Las estructuras de control se redujeron a las sentencias *If-*

Else y *While*; al igual que los modelos y animaciones fueron limitados en su complejidad, conservando una apariencia sencilla y concreta.

Los conceptos fundamentales de la programación estructurada que fueron representados por la aplicación incluyen la asignación y declaración de variables, las estructuras de control selectivas y repetitivas, la precedencia de operadores aritméticos y lógicos, el entorno de una función, la llamada y el retorno de una función, el paso de parámetros y las funciones recursivas.

Las analogías y animaciones diseñadas para apoyar a los conceptos de la programación estructurada se representaron principalmente a través de plataformas y contenedores utilizados por un actor principal, *Runaround*. Los contenedores representan a las variables, y pueden almacenar valores literales. Las plataformas simbolizan el entorno de una función, y sobre cada una de éstas se posicionan a los contenedores. El actor *Runaround* corre por la plataforma, asigna valores a las variables, evalúa expresiones, y si es necesario busca los valores a evaluar saltando entre las plataformas. La metáfora de varias plataformas apoya al concepto de ámbito de una función, y el salto entre plataformas a la llamada y retorno de una función.

Las tecnologías web resultaron ser apropiadas para el desarrollo del proyecto, durante la evaluación de la aplicación se confirmó la facilidad de acceso desde las máquinas del laboratorio de cómputo, ofreciendo un tiempo de carga aceptable. Las animaciones web en 3D generadas con *WebGL* ofrecen un buen rendimiento, y el servicio de alojamiento de *Google AppSpot* demostró una estabilidad y rendimiento aceptables.

5.2 Conclusiones

El proyecto *Runaround* integra diferentes enfoques de las aplicaciones educativas más populares y las combina de nuevas formas, siendo una de las primeras aplicaciones de su clase en funcionar completamente en la web, sin necesidad de *plug-ins*, y con gráficas en tercera dimensión. Durante el desarrollo del proyecto se enfrentaron una diversidad de retos al combinar las técnicas de análisis léxico y sintáctico, el utilizar un entorno web, y generar gráficas y animaciones en tercera dimensión. La aplicación *Jeliot* fue la fuente principal de inspiración con sus analogías del teatro; las metáforas diseñadas se complementaron con la inclusión de un actor, *Runaround*, y las acciones indirectas realizadas por éste. Se decidió

incluir un editor de código, ya que la aplicación se orienta a un nivel intermedio, y no a los niños pequeños.

Analizando los resultados de las encuestas y las experiencias durante el transcurso del proyecto, se concluye que es viable el estudio de este tipo de aplicaciones, ya que éstas poseen el potencial para ser un apoyo en los cursos de programación.

La integración de todos estos conceptos y técnicas previamente investigados por los respectivos proyectos similares, aunado a la facilidad de acceso web, y las animaciones en tercera dimensión, hacen de este proyecto un pequeño paso más en la búsqueda de la mejora y desarrollo de este tipo de aplicaciones didácticas, cuyo objetivo principal es apoyar a los estudiantes y despertar en ellos la curiosidad y el interés por el interesante mundo de la computación.

5.3 Recomendaciones para futuras investigaciones

La aplicación actual no reconoce las sintaxis del lenguaje C en su totalidad. A través de la encuesta realizada a los estudiantes se encontró que la sentencia *For* es muy utilizada y se esperaba el poder utilizarla en la aplicación. La ausencia de un análisis semántico también es una funcionalidad esperada por los estudiantes, ya que realmente esperaban que la aplicación les indicara los errores de sintaxis. Además, los temas de apuntadores y arreglos brillaron por su ausencia y fueron los temas más solicitados.

Como trabajo a futuro, puede retomarse el crear nuevas metáforas para los apuntadores y arreglos e integrarlas a las ya existentes. El desarrollo de un análisis semántico sería de gran utilidad para indicar a los estudiantes la ubicación y el tipo de errores en su código. Es importante también expandir el léxico y la gramática de la aplicación para soportar las sentencias más utilizadas comúnmente, como el ciclo *For*, la declaración de variables sin inicializar y la sintaxis de apuntadores y arreglos

Así mismo, se recomienda realizar pruebas con más estudiantes y profesores para identificar no sólo si la aplicación es de su agrado, sino demostrar si realmente la aplicación causa un impacto en la enseñanza y el aprendizaje de la programación, y finalmente, se aconseja indagar sobre otros posibles enfoques y metáforas, como el representar los algoritmos a través de sonidos, la edición de código con plantillas o las plataformas de aprendizaje colaborativo.

Referencias

- [1] Anthony Robins, Janet Rountree, and Nathan Rountree, "Learning and Teaching Programming: A Review and Discussion," *Computer Science Education*, vol. 13, no. 2, pp. 137-172, 2003.
- [2] Essi Lahtinen, Kirsti Ala-Mutka, and Hannu-Matti Järvinen, "A study of the difficulties of novice programmers," *SIGCSE Bull.*, vol. 37, no. 3, pp. 14-18, June 2005.
- [3] Stavroula Georgantaki and Symeon Retalis, "Using Educational Tools for Teaching Object Oriented Design and Programming," *Journal of Information Technology Impact*, vol. 7, no. 2, pp. 111-130, 2007.
- [4] S. Sandoval-Reyes, P. Galicia-Galicia, and I. Gutierrez-Sanchez, "Visual Learning Environments for Computer Programming," *Electronics, Robotics and Automotive Mechanics Conference (CERMA), 2011 IEEE*, pp. 439-444, 15-18, November 2011.
- [5] I.F. de Kereki, "Scratch: Applications in Computer Science 1," *Frontiers in Education Conference, 2008. FIE 2008. 38th Annual*, pp. T3B-7-T3B-11, 22-25, October 2008.
- [6] Wang Ting-Chung, Mei Wen-Hui, Lin Shu-Ling, Chiu Sheng-Kuang, and Lin J.M.-C., "Teaching programming concepts to high school students with alice," *Frontiers in Education Conference, 2009. FIE '09. 39th IEEE*, pp. 1-6, 18-21, October 2009.
- [7] Michael Kölling, "The Greenfoot Programming Environment," *Trans. Comput. Educ.*, vol. 10, no. 4, pp. 14:1-14:21, 2010.
- [8] M. Karakus, S. Uludag, E. Guler, S.W. Turner, and A. Ugur, "Teaching computing and programming fundamentals via App Inventor for Android," *Information Technology Based Higher Education and Training (ITHET), 2012 International Conference on*, pp. 1-8, 21-23, June 2012.
- [9] Andrew Rudder, Margaret Bernard, and Shareela Mohammed, "Teaching programming using visualization," *Proceedings of the Web Based Education*, 2007.
- [10] Dimitrios Doukakis, Grammatiki Tsaganou, and Maria Grigoriadou, "Using animated interactive analogies in teaching basic programming concepts and structures," *Proceedings of the Informatics Education Europe II Conference IEEEII 2007*, pp. 257-

265, 2007.

- [11] Steven R. Hansen and N. Hari Narayanan, "On the role of animated analogies in algorithm visualizations," *Proceedings of the Fourth International Conference of The Learning Sciences*, pp. 205-211, 2000.
- [12] Samer Al-Imamy, Javanshir Alizadeh, and Mohamed A. Nour, "On the Development of a Programming Teaching Tool: The Effect of Teaching by Templates on the Learning Process," *Journal of Information Technology Education*, vol. 5, pp. 271-283, 2006.
- [13] Martin C. Carlisle, Terry A. Wilson, Jeffrey W. Humphries, and Steven M. Hadfield, "RAPTOR: a visual programming environment for teaching algorithmic problem solving," *SIGCSE Bull*, vol. 37, no. 1, pp. 176-180, February 2005.
- [14] S. Maravic Cisar, R. Pinter, D. Radosav, and P. Cisar, "Software visualization: The educational tool to enhance student learning," *MIPRO, 2010 Proceedings of the 33rd International Convention*, pp. 990-994, 24-28, May 2010.
- [15] Clare J. Hopper et al., "AnnAnn and AnnAnn.Net: Tools for Teaching Programming," *JCP*, vol. 2, no. 5, pp. 9-16, 2007.
- [16] James L. Antonakos and Kenneth C. Mansfield, *Application Programming in Structured C*, 2nd ed. United States of America: Prentice Hall, Inc., 2000.
- [17] Luis Joyanes Aguilar, Andrés Castillo Sanz, and Lucas Sánchez García, *Programación en C*. Colombia: McGraw-Hill, 2002.
- [18] Leobardo López Román, *Programación Estructurada Un Enfoque Algorítmico*, 2nd ed. México: Alfaomega, 2004.
- [19] E.W. Dijkstra, *Notes on Structured Programming.*: Technological University, Department of Mathematics, 1970.
- [20] Norma Moroni and Perla Señas, "Estrategias para la enseñanza de la programación," *JEITICS 2005 - Primeras Jornadas de Educación en Informática y TICS en Argentina*, pp. 254-258, 2005.
- [21] M.N. Ismail, N. Azilah, U. Naufal, and U.T.M.C. Kelantan, "Instructional strategy in the teaching of computer programming: A need assessment analyses," *TOJET*, vol. 9,

- no. 2, pp. 125-131, 2010.
- [22] Ricardo Pérez Calderón, "Una Herramienta y Técnica para la Enseñanza de la Programación," *CICos 2008*, pp. 229-239, 2008.
 - [23] M. and Perrenet, J. and Jochems, W.M.G. and Zwaneveld, B. Saeli, "Teaching programming in secondary school: a pedagogical content knowledge perspective," *Informatics in Education-An International Journal*, vol. 10, no. 1, p. 73, 2011.
 - [24] E. Costelloe, "Teaching programming the state of the art," CRITE Technical Report, 2004.
 - [25] W.W.F Lau and A.H.K. Yuen, "Toward a framework of programming pedagogy," *Encyclopedia of Information Science and Technology*, vol. 8, pp. 3772-3777, 2009.
 - [26] Z. Luna and A. Rosalia, "Estrategias metodológicas y su influencia en el inter-aprendizaje de la asignatura de programación en lenguaje estructurado en el segundo año de bachillerato especialización aplicaciones en informática del instituto tecnológico Ismael Pérez Pazmiño de la ," PhD Thesis 2012.
 - [27] Alfred V. Aho and Ravi Sethi, *Compilers: Principles, Techniques and Tools*, 2nd ed. United States of America: Addison-Wesley, 1998.
 - [28] Thomas Pittman and James Peters, *The Art of Compiler Design*. United States of America: Prentice-Hall, 1992.
 - [29] Kenneth C. Loudon, *Construcción de Compiladores: Principios y Práctica*. United States of America: Thomson, 2005.
 - [30] Jacinto Ruiz Catalán, *Compiladores: Teoría e Implementación*. México: Alfa-Omega, 2010.
 - [31] Bernard Teufel, Stephanie Schmidt, and Thomas Teufel, *Compiladores: Conceptos Fundamentales*. United States Of America: Addison-Wesley, 1993.
 - [32] M. E. Lesk and E. Schmidt, "Lex – A Lexical Analyzer Generator," Bell Laboratories, Murray Hill, New Jersey, Science Technical Report No. 39 1975.
 - [33] Stephen C. Johnson, "Yacc: Yet Another Compiler Compiler," Bell Laboratories, Murray Hill, New Jersey, Technical Report No. 32 1975.
 - [34] M. Najork, "Web-based algorithm animation," *Design Automation Conference*, 2001.

- Proceedings*, pp. 506-511, 2001.
- [35] Andreas Kerren and John Stasko, *Chapter 1 Algorithm Animation*, Stephan Diehl, Ed.: Springer Berlin / Heidelberg, 2002.
 - [36] J. Haajanen et al., "Animation of user algorithms on the Web," in *Visual Languages, 1997. Proceedings. 1997 IEEE Symposium on*, 1997, pp. 356-363.
 - [37] M.H. Brown and J. Hershberger, "Color and Sound in Algorithm Animation," *Computer*, vol. 15, no. 12, pp. 52-63, 1992.
 - [38] S. Ortiz, "Is 3D Finally Ready for the Web?," *Computer*, vol. 43, no. 1, pp. 14-16, January 2010.
 - [39] K. Kapetanakis and S. Panagiotakis, "Evaluation of techniques for web 3D graphics animation on portable devices," *Telecommunications and Multimedia (TEMU), 2012 International Conference on*, pp. 152-157, August 2012.
 - [40] M.A. Bochicchio, A. Longo, and L. Vaira, "Extending Web applications with 3D features," *Web Systems Evolution (WSE), 2011 13th IEEE International Symposium on*, pp. 93-96, September 2011.
 - [41] Huang Zhanpeng, Gong Guanghong, and Han Liang, "NetGL: A 3D Graphics Framework for Next Generation Web," *Multimedia Information Networking and Security (MINES), 2011 Third International Conference on*, pp. 105-108, November 2011.
 - [42] John C. Mitchell, *Concepts in programming languages.*: Cambridge University Press, 2002.
 - [43] C.A. Shaffer, M. Cooper, and S.H. Edwards, "Algorithm visualization: a report on the state of the field," *ACM SIGCSE Bulletin*, vol. 39, no. 1, pp. 150-154, 2007.
 - [44] Rosa Sancho, "Versión española de la sexta edición del manual de Frascati: Propuesta de norma práctica para encuestas de investigación y desarrollo experimental," Digital.CSIC -Consejo Superior de Investigaciones Científicas, 2003.

Apéndices

Apéndice A: Protocolo

Universidad Autónoma de Ciudad Juárez
Instituto de Ingeniería y Tecnología



Propuesta del tema para la materia de Seminario de Titulación I

Nombre: Luis Eduardo Salazar Valles	Matrícula: 86406
Programa Académico: Ingeniería en Sistemas Computacionales	
Departamento: Eléctrica y Computación	

Título: Runaround: Aplicación orientada a la enseñanza y aprendizaje de la programación estructurada a través de animaciones generadas a partir de código fuente.

Contextualización: En los cursos iniciales de programación de la UACJ se realiza una introducción a la técnica de la programación estructurada. En estos cursos el profesor enseña utilizando herramientas de apoyo como los editores gráficos de diagramas de flujo así como también el lenguaje de programación C, por ser este la base de muchos otros lenguajes. Por otra parte el estudiante se enfrenta por primera vez a diversos conceptos fundamentales necesarios para una representación y estructura adecuada de los problemas a resolver.

Definición del Problema: La enseñanza de la programación estructurada a estudiantes con poca o ninguna experiencia involucra diversos retos educativos para los profesores. Por otra parte los estudiantes requieren recordar y asimilar una variedad de ideas nuevas, así como también manipular símbolos y reglas sin una analogía del mundo real evidente, lo cual dificulta al mismo tiempo el aprendizaje.

Objetivo: Desarrollar una aplicación que apoye a los profesores en la enseñanza y a los estudiantes en el aprendizaje de los conceptos fundamentales de la programación estructurada a través de animaciones generadas a partir del código fuente

Preguntas de Investigación:

¿Qué analogías y diseño de animaciones apoyarán a la enseñanza y aprendizaje de los conceptos fundamentales de la programación estructurada?
¿Cuáles conceptos fundamentales de la programación estructurada serán representados?
¿Qué tecnologías son adecuadas para el desarrollo de la aplicación?

Justificación: La aplicación apoyará a la enseñanza y aprendizaje de los conceptos fundamentales de la programación estructurada, ya que:

- Los conceptos serán representados del código abstracto a las animaciones concretas.
- Será de fácil acceso para profesores y alumnos, debido a que podrá ejecutarse en línea a través de un navegador.
- Propiciará un ambiente para que el estudiante practique y aprenda por sí mismo.

Solución Propuesta: Se propone el desarrollo de una aplicación que genere animaciones de los conceptos fundamentales de la programación estructurada a partir del código fuente. La aplicación podrá utilizarse fácilmente en línea para ingresar código en varias sentencias.

Al procesar el código se visualizarán las representaciones de los conceptos correspondientes a través de animaciones, pasando del punto de vista abstracto y conciso del código a las animaciones concretas y detalladas, para de esta forma apoyar a profesores y estudiantes en la enseñanza y aprendizaje de la programación estructurada.

Metodología Propuesta:

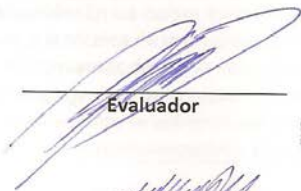
- Investigar:
 - Aplicaciones similares de apoyo a la enseñanza y aprendizaje de la programación.
 - La teoría de los compiladores y traductores así como su desarrollo.
 - Los conceptos fundamentales de la programación estructurada.
- Seleccionar los conceptos más adecuados para un nivel principiante y diseñar sus analogías y animaciones correspondientes.
- Se propone investigar la fase de análisis del proceso de compilación:
 - Análisis léxico
 - Análisis sintáctico
 - Análisis semántico
- Se propone que la aplicación dibuje en pantalla las animaciones correspondientes a partir del análisis del código fuente.
- Evaluar las distintas herramientas de programación y seleccionar las que faciliten el desarrollo de una aplicación eficiente y de fácil acceso.
- Desarrollar la aplicación:
 - Requisitos
 - Diseño
 - Codificación
 - Pruebas

Observaciones del evaluador: ☐ Aprobado ☐ Aprobado con condición ☐ Rechazado

Fecha de terminación del proyecto: Mayo 2013

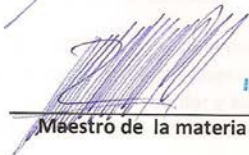
Nombre del asesor responsable: Ivonne Haydee Robledo Portillo

Luis Eduardo Salazar V.
Alumno


Evaluador




Evaluador


Maestro de la materia

UACJ
DEPARTAMENTO DE
INGENIERÍA ELÉCTRICA
Y COMPUTACIÓN


Asesor

Apéndice B: Ejemplos de la codificación

B.1 Analizador léxico (fragmento en *Python*):

```
import ply.lex as lex

tokens = (
    'AND',
    'ANDAND',
    'COMMA',
    'DIVIDE',
    'ELSE',
    'EQUAL',
    'EQUALEQUAL',
    'FALSE',
    'GE',
    'GT',
    'IDENTIFIER',
    'IF',
    'LBRACE',
    'LE',
    'LPAREN',
    'LT',
    'MINUS',
    'MOD',
    'NOT',
    'NUMBER',
    'OROR',
    'PLUS',
    'RBRACE',
    'RETURN',
    'RPAREN',
    'SEMICOLON',
    'STRING',
    'TIMES',
    'TRUE',
    'WHILE',
    'INT',
    'CHAR',
    'FLOAT',
    'VOID'
)

states = (
    ('comment', 'exclusive'),
)

def t_comment(t):
    r'\/*\*'
    t.lexer.begin('comment')

def t_comment_end(t):
    r'\*\/'
    t.lexer.lineno += t.value.count('\n')
    t.lexer.begin('INITIAL')
    pass

def t_comment_error(t):
    t.lexer.skip(1)

def t_eolcomment(t):
    r'(?:(?:\/*\/*)|(?:\#)).*'
    pass

reserved = [ 'if', 'return', 'else', 'true', 'false', 'while', 'int', 'char', 'float',
'void' ]

def t_IDENTIFIER(t):
```



```

        r'[A-Za-z][0-9A-Za-z_]*'
        if t.value in reserved:
            t.type = t.value.upper()
        return t

def t_NUMBER(t):
    r'-?[0-9]+(\.[0-9]*)?'
    return t

def t_STRING(t):
    r'"([^"\\]|\\.)*"'
    return t

t_AND = r'&'
t_ANDAND = r'&&'
t_COMMA = r','
t_DIVIDE = r '/'
t_EQUALEQUAL = r'=='
t_EQUAL = r'='
t_LPAREN = r'('
t_LBRACE = r'{'
t_RBRACE = r'}'
t_SEMICOLON = r';'
t_MINUS = r '-'
t_MOD = r '%'
t_NOT = r '!'
t_OROR = r '||'
t_PLUS = r '+'
t_RPAREN = r ')'
t_TIMES = r '*'
t_LE = r '<='
t_LT = r '<'
t_GT = r '>'
t_GE = r '>='

t_ignore = ' \t\v\r'
t_comment_ignore = ' \t\v\r'

def t_newline(t):
    r'\n'
    t.lexer.lineno += 1

def t_error(t):
    print "Lexer: Illegal character " + t.value[0]
    t.lexer.skip(1)

```

B.2 Analizador sintáctico (fragmento en Python):

```
import sys
sys.path.insert(0, '../')

import ply.yacc as yacc
from ctokens import tokens
from pymag_trees.gen import Tree

start = 'trans_unit'
precedence = (
    ('left', 'OROR'),
    ('left', 'ANDAND'),
    ('left', 'EQUALEQUAL'),
    ('left', 'LT', 'LE', 'GT', 'GE'),
    ('left', 'PLUS', 'MINUS'),
    ('left', 'TIMES', 'DIVIDE', 'MOD'),
    ('right', 'NOT'),
    ('nonassoc', 'LOWER_THAN_ELSE'),
    ('nonassoc', 'ELSE'),
)

def p_element_stmt(p):
    'element : stmt'
    p[0] = p[1]

def p_element_function(p):
    'element : type identifier LPAREN optparams RPAREN compoundstmt'
    p[0] = Tree(Node('function'),
        p[1], p[2], p[4], p[6])

def p_identifier(p):
    'identifier : IDENTIFIER'
    p[0] = Tree(Node('identifier', span(p, 1, 1)),
        Tree(Node(p[1], span(p, 1, 1))))

def p_type(p):
    '''type : INT
            | CHAR
            | FLOAT
            | VOID'''
    p[0] = Tree(Node('type', span(p, 1, 1)),
        Tree(Node(p[1], span(p, 1, 1))))

def p_stmt_if(p):
    'estmt : IF exp stmt_or_compound optsemi %prec LOWER_THAN_ELSE'
    p[0] = Tree(Node('if-then'),
        p[2], p[3])

def p_stmt_while(p):
    'estmt : WHILE exp compoundstmt optsemi'
    p[0] = Tree(Node('while'),
        p[2], p[3])

def p_stmt_if_else(p):
    'estmt : IF exp compoundstmt ELSE stmt_or_compound optsemi'
    p[0] = Tree(Node('if-then-else'),
        p[2], p[3], p[5])

def p_stmt_declaration(p):
    'estmt : type identifier EQUAL exp optsemi'
    p[0] = Tree(Node('declaration', span(p, 1, 4)),
        p[1], p[2], Tree(Node(p[3], span(p, 3, 3))), p[4])

def p_exp_number(p):
    'exp : NUMBER'
    p[0] = Tree(Node('number', span(p, 1, 1)),
        Tree(Node(p[1], span(p, 1, 1))))
```

```

def p_exp_string(p):
    'eexp : STRING'
    p[0] = Tree(Node('string', span(p, 1, 1)),
                 Tree(Node(p[1])))

def p_exp_binop(p):
    '''eexp : exp PLUS exp
           | exp MINUS exp
           | exp TIMES exp
           | exp MOD exp
           | exp DIVIDE exp
           | exp EQUALEQUAL exp
           | exp LE exp
           | exp LT exp
           | exp GE exp
           | exp GT exp
           | exp ANDAND exp
           | exp OROR exp'''
    p[0] = Tree(Node('binop', span(p, 1, 3)),
                 p[1], Tree(Node(p[2], span(p, 2, 2))), p[3])

def p_exp_call(p):
    'eexp : identifier LPAREN optargs RPAREN'
    p[0] = Tree(Node('call', span(p, 1, 4)),
                 p[1], p[3])

def p_error(p):
    pass

```

B.3 Intérprete (fragmento en *Javascript*):

```
window.IRUNAROUND = window.IRUNAROUND || {};  
window.IRUNAROUND.Backend = (function() {  
  'use strict'  
  return function() {  
    var ast;  
    var path = [];  
    var funstack = [];  
    var env = {  
      'parent': undefined,  
      'values': {  
        'printf output': '',  
        'scanf input': null  
      }  
    };  
  
    var breadthWalk = function(ast) {  
      var result = [];  
      var nodes = [];  
      nodes.push(ast);  
      while (nodes.length > 0) {  
        var cur = nodes.pop();  
        var lchild = cur.children[0];  
        var rchild = cur.children[1];  
        if (lchild && lchild.node.type) {  
          result.push(lchild);  
        }  
        if (rchild) {  
          nodes.push(rchild);  
        }  
      }  
      return result;  
    };  
  
    var envLookup = function(vname, env) {  
      if (env.values[vname] !== undefined) {  
        return env.values[vname];  
      } else if (env.parent === undefined) {  
        return undefined;  
      } else {  
        return envLookup(vname, env.parent);  
      }  
    };  
  
    var envUpdate = function(vname, value, env) {  
      if (env.values[vname] !== undefined) {  
        env.values[vname] = value;  
      } else if (env.parent !== undefined) {  
        envUpdate(vname, value, env.parent);  
      }  
    };  
  
    var evalNode = function(ast) {  
      switch (ast.node.type) {  
        // Elements  
        case 'function':  
          var fname = ast.children[1].children[0].node.type;  
          env.values[fname] = new FuncDec(fname, ast, env);  
          break;  
  
        // Statements  
        case 'if-then-else':  
          if (ast.children[0].node.value() === true) {  
            var thenstmts = breadthWalk(ast.children[1]);  
            unshiftPath(thenstmts);  
          } else {  
            var elsetstmts = breadthWalk(ast.children[2]);  
            unshiftPath(elsetstmts);  
          }  
        }  
      }  
    };  
  };  
})();
```

```

        }
    break;

    case 'while':
        if (ast.children[0].node.value() === true) {
            var condstmts = depthWalk(ast);
            unshiftPath(condstmts);
            var whilestmts = breadthWalk(ast.children[1]);
            unshiftPath(whilestmts);
        }
    break;

    case 'declaration':
        var varname = ast.children[1].children[0].node.type;
        env.values[varname] = ast.children[3].node.value();
        ast.node.value = function() {
            return varname + ' = ' + env.values[varname];
        };
    break;

    // Expressions
    case 'number':
        ast.node.value = function() {
            return parseFloat(ast.children[0].node.type);
        }
    break;

    case 'string':
        ast.node.value = function() {
            return ast.children[0].node.type;
        };
    break;

    case 'identifier':
        var idname = ast.children[0].node.type;
        ast.node.value = function() {
            return envLookup(idname, env);
        };
    break;

    case 'binop':
        var op = ast.children[1].node.type;
        var lexp = ast.children[0].node.value();
        var rexp = ast.children[2].node.value();
        switch (op) {
            case '+':
                ast.node.value = function() {
                    return lexp + rexp;
                };
                break;
            case '-':
                ast.node.value = function() {
                    return lexp - rexp;
                };
                break;
            case '*':
                ast.node.value = function() {
                    return lexp * rexp;
                };
                break;
            case '/':
                ast.node.value = function() {
                    return lexp / rexp;
                };
                break;
        }
    break;
}

};

});

})();

```

B.4 Animaciones (fragmento en Javascript):

```
window.IRUNAROUND = window.IRUNAROUND || {};  
window.IRUNAROUND.GRAPHICS = window.IRUNAROUND.GRAPHICS || {};  
window.IRUNAROUND.GRAPHICS.World = (function(THREE, TWEEN) {  
    'use strict'  
    return function() {  
        var scene, renderer;  
        var ambientLight, lightFront, lightBack;  
        var camera, controls;  
        var stepWait = 500;  
        var robotTexture;  
        var robotMorphModel;  
        var clock = new THREE.Clock();  
        var animationDelta = 0;  
        var animationElapsed = 0;  
  
        var BoxBehaviours = { Opening: 0, Open: 1, Closing: 2, Closed: 3 };  
        function Box(identifier, background, scopePlatform, minWidth, fontSize) {  
            this.open = function(callback) {  
                new TWEEN.Tween(lidRotator.rotation)  
                    .to({  
                        z: Math.PI * 3/4,  
                    }, stepWait * 2  
                ).easing(  
                    TWEEN.Easing.Elastic.Out  
                ).onStart(function() {  
                    self.behaviour = BoxBehaviours.Opening;  
                }).onComplete(function() {  
                    self.behaviour = BoxBehaviours.Open;  
                    if (callback !== undefined) {  
                        callback();  
                    }  
                }).start(animationElapsed);  
            }  
  
            this.close = function(callback) {  
                new TWEEN.Tween(lidRotator.rotation)  
                    .to({  
                        z: 0,  
                    }, stepWait  
                ).easing(  
                    TWEEN.Easing.Cubic.Out  
                ).onStart(function() {  
                    self.behaviour = BoxBehaviours.Closing;  
                }).onComplete(function() {  
                    self.behaviour = BoxBehaviours.Closed;  
                    if (callback !== undefined) {  
                        callback();  
                    }  
                }).start(animationElapsed);  
            }  
        };  
  
        var RobotBehaviours = { Stand: 0, Run: 1, StandGrab: 2, RunGrab: 3 };  
        function Robot(morph) {  
            this.grabLiteral = function(value) {  
                this.dropLiteral();  
                this.literal = new Text3d(value, 80, '#999');  
                this.literal.position.y = 150;  
                this.object3d.add(this.literal);  
                this.stand();  
            };  
  
            this.dropLiteral = function() {  
                if (this.literal !== undefined) {  
                    this.object3d.remove(this.literal);  
                    this.literal = undefined;  
                    this.stand();  
                }  
            }  
        }  
    }  
})(THREE, TWEEN);
```

```

};

this.stand = function() {
    if (this.literal === undefined) {
        this.behaviour = RobotBehaviours.Stand;
        this.morph.setFrameRange(21, 22);
    } else {
        this.behaviour = RobotBehaviours.StandGrab;
        this.morph.setFrameRange(44, 45);
    }
};

this.run = function() {
    if (this.literal === undefined) {
        this.behaviour = RobotBehaviours.Run;
        this.morph.setFrameRange(1, 20);
    } else {
        this.behaviour = RobotBehaviours.RunGrab;
        this.morph.setFrameRange(23, 42);
    }
};

this.openBox = function(callback) {
    self.curBox.open(function() {
        if (callback !== undefined) {
            callback();
        }
    });
};

this.closeBox = function(callback) {
    self.curBox.close(function() {
        if (callback !== undefined) {
            callback();
        }
    });
};

var makeTextTexture = function(text, color, fontsize, background, width, height) {
    var bitmap = document.createElement('canvas');
    var g = bitmap.getContext('2d');
    fontsize = fontsize * 1.0 || 35;
    g.font = fontsize + 'px Calibri';
    var textMetrics = g.measureText(text);

    var padding = 20 * 2;
    bitmap.width = textMetrics.width + padding < width ?
        width :
        textMetrics.width + padding;
    bitmap.height = height;
    var gx = bitmap.width / 2;
    var gy = bitmap.height / 2;

    g.fillStyle = background;
    g.fillRect(0, 0, bitmap.width, bitmap.height);

    g.fillStyle = color;
    g.font = fontsize + 'px Calibri';
    g.textAlign = 'center';
    g.textBaseline = 'middle';
    g.fillText(text, gx, gy);

    var texture = new THREE.Texture(bitmap);
    texture.needsUpdate = true;
    return texture;
}
})(THREE, TWEEN);

```

Apéndice C: Encuesta de evaluación

Universidad Autónoma de Ciudad Juárez
Departamento de Ingeniería Eléctrica y Computación
Ingeniería en Sistemas Computacionales

Estimado compañero(a)

A continuación, se te presenta una pequeña encuesta relacionada con el proyecto de titulación "Runaround: Aplicación orientada a la enseñanza y aprendizaje de la programación estructurada a través de animaciones generadas a partir de código fuente", realizado por el alumno Luis Eduardo Salazar Valles con matrícula 86406 y supervisado por la Mtra. Ivonne Haydee Robledo Portillo.

Este proyecto busca apoyar a los profesores y alumnos con la enseñanza y aprendizaje de los conceptos fundamentales de la programación estructurada utilizando metáforas y animaciones 3D. A través de esta encuesta y tu valiosa opinión, se determinará si el proyecto cumple los objetivos propuestos; y al mismo tiempo, el considerar mejoras o detectar problemas con la aplicación actual.

La información que nos proporciones será utilizada exclusivamente para fines académicos; no será divulgada para ningún otro propósito. Tu información es confidencial y muy valiosa para nosotros, te pedimos respuestas de una forma completamente honesta y objetiva; con tu opinión, comentarios o sugerencias.

¡Tú también puedes ayudar a mejorar la educación de la programación en tu universidad!

PARTE A. Datos generales						
Nombre					Matrícula	Horario
Semestre	Grupo		Edad		Género	
					<input type="checkbox"/> Masculino	<input type="checkbox"/> Femenino
Materia					Profesor	

PARTE B. Preguntas de diagnóstico						
Ordena del 1 al 5, comenzando por los temas que consideras más difíciles de comprender						
<input type="checkbox"/> Variables globales y locales	<input type="checkbox"/> Precedencia de operadores aritméticos y lógicos	<input type="checkbox"/> Condiciones simples y múltiples	<input type="checkbox"/> Ciclos de repetición			
¿Utilizas alguna aplicación didáctica como apoyo en tu clase?		<input type="checkbox"/> Sí <input type="checkbox"/> No				
¿Conoces alguna de las siguientes aplicaciones educativas? (marca las casillas que apliquen a continuación)						
<input type="checkbox"/> Alice	<input type="checkbox"/> Scratch	<input type="checkbox"/> Raptor	<input type="checkbox"/> Greenfoot	<input type="checkbox"/> DFD	<input type="checkbox"/> Jeliot	<input type="checkbox"/> Otras
¿Crees que realmente sería de utilidad utilizar este tipo de aplicaciones para aprender programación?					<input type="checkbox"/> Sí <input type="checkbox"/> No	
¿Por qué?						

PARTE C. Contenido de la aplicación	
Califica del 1 al 5 de acuerdo a tu opinión (1- muy mal , 2- mal, 3-regular, 4- muy bien, 5- excelente)	
Opciones	Calificación
Los cubos contenedores con tapas móviles, en la aplicación, ayudan a comprender la declaración y asignación de variables	1 2 3 4 5
Las plataformas que contienen a los cubos contenedores, apoyan los conceptos del alcance global y local de las variables	1 2 3 4 5
Los diferentes colores de los cubos contenedores ayudan a identificar rápidamente los tipos de datos	1 2 3 4 5
La vista de árbol muestra de forma intuitiva, paso a paso, el flujo de control (orden de las instrucciones) de la ejecución	1 2 3 4 5
El texto 3D que flota sobre las plataformas muestra el orden de las operaciones de una forma fácil de comprender	1 2 3 4 5
La animación de salto que realiza el personaje entre las plataformas de la aplicación, es una metáfora clara y sencilla de la llamada a una función y el retorno de los valores.	1 2 3 4 5
En general, ¿consideras que los conceptos fundamentales de la programación son representados de una forma intuitiva y fácil de comprender?	<input type="checkbox"/> Sí <input type="checkbox"/> No
¿Por qué?	
¿Qué otros temas de la programación que no se mostraron en la aplicación actual te gustaría que fuesen representados?	

PARTE D. Funcionalidad de la aplicación	
Califica del 1 al 5 de acuerdo a tu opinión (1- muy mal , 2- mal, 3-regular, 4- muy bien, 5- excelente)	
Opciones	Calificación
El tiempo de carga inicial de la aplicación es aceptable	1 2 3 4 5
Las animaciones 3D se visualizan correctamente y de una forma fluida	1 2 3 4 5
El manejo de la cámara 3D es muy útil y fácil de utilizar	1 2 3 4 5
Las opciones de pausar la ejecución y ajustar la velocidad son muy útiles, y funcionan correctamente	1 2 3 4 5
El editor de código fuente es fácil de utilizar y el resaltado de la sintaxis es muy útil	1 2 3 4 5
Es muy fácil interactuar con los datos de entrada y salida durante la ejecución	1 2 3 4 5
En general, ¿consideras que las opciones incluidas en la aplicación son suficientes, útiles y fáciles de usar?	<input type="checkbox"/> Sí <input type="checkbox"/> No
¿Por qué?	
¿Qué otras opciones o funcionalidades que no se encuentran disponibles actualmente en la aplicación te gustaría que fuesen incluidas?	

PARTE E. Diseño de la aplicación	
Califica del 1 al 5 de acuerdo a tu opinión (1- muy mal , 2- mal, 3-regular, 4- muy bien, 5- excelente)	
Opciones	Calificación
El diseño gráfico de la aplicación y la organización de los elementos , ayuda a localizar las opciones fácilmente	1 2 3 4 5
Los colores de la interfaz son agradables para el usuario y logran un contraste adecuado entre los elementos	1 2 3 4 5
Los diseños de los modelos 3D, formas y colores, ayudan a diferenciar rápidamente los conceptos de la programación	1 2 3 4 5
Las animaciones de los modelos 3D en la aplicación, permiten apreciar fácilmente las acciones efectuadas	1 2 3 4 5
El tipo y tamaño de letra utilizados en la aplicación, facilitan la lectura del código en el editor y del texto en las opciones	1 2 3 4 5
¿Tuviste algún problema o error al manejar la aplicación?	<input type="checkbox"/> Sí <input type="checkbox"/> No
¿Cuáles?	

Gracias por su tiempo y ayuda en este proyecto.
Sugerencias y comentarios:

Apéndice D: Resultados de las encuestas

Parte A. Datos Generales									
ID	Nombre	Matrícula	Horario	Semestre	Grupo	Edad	Género	Materia	Profesor
1	Patricia C. Parroquín Amaya								
2	Edson Rodríguez Castañón	121150	9:00-11:00 Martes y Jueves	2	D	18	Masculino	Programación I	Patricia Parroquín
3	Gustavo Alonso Quiroz Terrones	116074	9:00-11:00 Martes y Jueves	4	D	19	Masculino	Programación I	Patricia Parroquín
4	Jesus Alejandro Cardiel V.	99485	9:00-11:00 Martes y Jueves	4	D	22	Masculino	Programación I	Patricia Parroquín
5	Antonio Guadalupe López Barraza	120624	9:00-11:00 Martes y Jueves	2	D	19	Masculino	Programación I	Patricia Parroquín
6	Juan Ubaldo Ruedas Cerda	121064	9:00-11:00 Martes y Jueves	2	D	18	Masculino	Programación I	Patricia Parroquín
7	Norma Angélica Román Haro	108476	9:00-11:00 Martes y Jueves	5	D	27	Femenino	Programación I	Patricia Parroquín
8	Cedillo Mackenzie Jose Jesus	120963	9:00-11:00 Martes y Jueves	2	D	23	Masculino	Programación I	Patricia Parroquín
9	Calderon	121074	9:00-11:00 Martes y Jueves		D	32	Masculino	Programación I	Patricia Parroquín
10	Francisco Daniel Casas Zaragoza	120971	11:00-13:00 Martes y Jueves	2	E	18	Masculino	Programación I	Patricia Parroquín
11	Vicente Zesati Rocha	113030	11:00-13:00 Martes y Jueves	4	E	19	Masculino	Programación I	Patricia Parroquín
12	Sandra Luz Yañez Rivera	121052	11:00-13:00 Martes y Jueves	2	E	20	Femenino	Programación I	Patricia Parroquín
13	Cynthia Karina Valencia Bravo	122697	11:00-13:00 Martes y Jueves	2	E		Femenino	Programación I	Patricia Parroquín
14	Pedro Tovas Esquivel G	120970	11:00-13:00 Martes y Jueves	2	E	18	Masculino	Programación I	Patricia Parroquín
15	Mario Enrique Rivera Vidaña	120835	11:00-13:00 Martes y Jueves	2	E	19	Masculino	Programación I	Patricia Parroquín
16	Cintya Lizbeth Hernández S	121074	11:00-13:00 Martes y Jueves	2	E	18	Femenino	Programación I	Patricia Parroquín
17	Angel Muñiz Modesto	121013	11:00-13:00 Martes y Jueves	2	E	22	Masculino	Programación I	Patricia Parroquín
18	Pedro Adrián Valenzuela Carreón	116476	11:00-13:00 Martes y Jueves	4	E		Masculino	Programación I	Patricia Parroquín
19	Ileana Abril Miranda Bosquez	120998	11:00-13:00 Martes y Jueves	2	E	19	Femenino	Programación I	Patricia Parroquín
20	Alan Martínez Reyes	121080	11:00-13:00 Martes y Jueves	2	E	19	Masculino	Programación I	Patricia Parroquín
21	Octavio Rucobo Contreras	120988	11:00-13:00 Martes y Jueves	2	E	18	Masculino	Programación I	Patricia Parroquín
22	Marcia Argueta Ramírez	120944	11:00-13:00 Martes y Jueves	2	E	18	Femenino	Programación I	Patricia Parroquín
23	Rubi Dominguez	98656	11:00-13:00 Martes y Jueves		E	22	Femenino	Programación I	Patricia Parroquín
24	Karla Jimenez Ramirez	127457	11:00-13:00 Martes y Jueves	1	E	23	Femenino	Programación I	Patricia Parroquín

25	Allbee Daniela Gallegos G	121031	11:00-13:00 Martes y Jueves	2	E		Femenino	Programación I	Patricia Parroquín
26	Luis Alberto Bautista Juarez	114118	11:00-13:00 Martes y Jueves	4	B	20	Masculino	Programación I I	Vianey Cruz
27	José Luis Arauz Sosa	127463	11:00-13:00 Martes y Jueves	4	B	22	Masculino	Programación I I	Vianey Cruz
28	Jose Adrian Pule Tobias	108501	11:00-13:00 Martes y Jueves	5	B	20	Masculino	Programación I I	Vianey Cruz
29	Guillermo López Muela	105819	11:00-13:00 Martes y Jueves	5	B	20	Masculino	Programación I I	Vianey Cruz
30	Fredy Fernando Alvarez Sanchez	109311	11:00-13:00 Martes y Jueves	5	B	31	Masculino	Programación I I	Vianey Cruz
31	Gabino Moreno D	117932	11:00-13:00 Martes y Jueves	6	B	22	Masculino	Programación I I	Vianey Cruz
32	Emmanuel Duarte Bistrain	108567	11:00-13:00 Martes y Jueves	5	B	24	Masculino	Programación I I	Vianey Cruz
33	Ponce Gardea Jorge Alberto	113029	11:00-13:00 Martes y Jueves	4	B	19	Masculino	Programación I I	Vianey Cruz
34	Alejandro Delgado Hernández	108633	11:00-13:00 Martes y Jueves	5	B	21	Masculino	Programación I I	Vianey Cruz

Parte B. Preguntas de diagnóstico														
Ordena del 1 al 5, comenzando por los temas que consideres más difíciles de comprender						¿Conoces alguna de las siguientes aplicaciones educativas?						¿Crees que realmente sería de utilidad utilizar este tipo de aplicaciones para aprender a programar?		
ID	Variab les globales y locales	Preceden cia de operador es aritméticos y lógicos	Condi ciones simples y múltiples	Ciclos de repetición	¿Utilizas alguna aplicación didáctica como apoyo en tu clase?	A l i c e	S c c h	R a p t o	G r e f o D D	J e l t o s	O t r a s	¿Por qué?		
1	5	3	5	3	Sí									
2	5	4	3	2	Sí			Sí	Sí		Sí	"te enseña la lógica de la programación"		
3	3	2	5	4	No			Sí	Sí		Sí	"a veces puede resultar mas entretenido"		
4	3	4	5	5					Sí		Sí	"pues son herramientas que facilitan el aprendizaje"		
5	4	3	2	1	Sí				Sí		Sí	"son las herramientas basicas de la programación"		
6	2	3	5	1	No			Sí	Sí		Sí	"ayuda a entender mejor la funcionalidad de funciones y de temas en general en C."		
7	3	4	1	2	No				Sí		Sí	"facilitaría la comprensión de los conocimientos aprendidos y la forma más practica de solucionar los problemas de los clientes."		
8	5	4	3	1	Sí				Sí		Sí	"porque te ayuda a comprender la lógica del programa"		
9	1	2	3	4	No			Sí	Sí		Sí	"es mas didactico y se dificulta menos el aprendizaje"		
10	4	1	2	3	No			Sí	Sí		Sí	"porque muestra el funcionamiento del código paso a paso."		
11	2	4	3	1	No			Sí	Sí		Sí	"plantar bases"		
12	3	4	2	1	No			Sí	Sí		Sí	"el usuario interactua de tal forma que la aplicación parece un juego y eso hace que le interese un poco mas la programacion vista desde otro modo de aprendizaje"		
13	5	2	4	1	Sí			Sí	Sí		Sí	"si, porque ayuda a los alumnos a verificar si su lógica esta siendo bien implementada mediante el uso de estas herramientas."		
14	4	3	2	1	No			Sí	Sí		Sí	"son una manera más simple para introducirse a los conceptos básicos de la programación"		
15	3	4	2	1	No			Sí	Sí		Sí	"porque te explica de forma visual lo que hace el programa."		
16	1	2	3	4	No			Sí	Sí		Sí	"porque puedes comprender de una manera más gráfica y simple lo que realiza el código"		
17	1	3	4	2	Sí			Sí	Sí		Sí	"ayudan a comprender mejor los conceptos"		

18	4	1	2	3	No			Sí			Sí	"porque facilita el aprendizaje ya que, al asociar imágenes el aprendizaje se vuelve más factible, ya que según la psicología las imágenes las asociamos a lo que aprendemos. O mejor dicho para que se nos quede grabado lo que aprendemos, le asociamos imágenes."
19	2	4	3	1	Sí				Sí		Sí	"porque al empezar a programar es un buen apoyo para entenderlo más."
20	2	4	1	3	No			Sí	Sí		No	"se pierde mucho tiempo en el diagrama"
21	1	3	4	2	Sí				Sí		Sí	"para tener un orden y una estructura antes de comenzar a programar"
22	4	3	1	2	No			Sí	Sí		Sí	"porque gráficamente es más sencillo comprender algunos conceptos que en forma de código son algo complicados."
23	4	2	3	1	Sí			Sí		Sí	Sí	"si, seria una manera mas entendible"
24	4	2	1	3	Sí				Sí			
25	5	2	1	4	No			Sí	Sí		No	"entiendo mejor con un código, que el diagrama, simplemente es más fácil."
26					No			Sí	Sí	Sí	Sí	"porque es mas interactivo."
27	4	1	2	3	No						Sí	"es necesario que los alumnos entiendan el concepto de programación, creo que mucha gente no lo comprende."
28	1	4	3	2	No			Sí	Sí		Sí	"la animación es clara, una aplicación que tendria buenos fines educativos ya que al ser visual e intuitivo te permite tener una mejor percepcion de la programación"
29	4	3	1	2	No			Sí	Sí	Sí	Sí	"da una mayor explicación y orientación sobre lo que ocurre al momento de compilar un programa"
30	4	1	3	2	No			Sí			Sí	"el medio grafico es un excelente apoyo para visualizar lo que esta ocurriendo en la ejecución de un codigo."
31	1	3	2	4	Sí			Sí			Sí	"porque hay alumnos que no tienen el conocimiento o idea de lo que es programar."
32	4	3	2	1	No				Sí		Sí	
33	4	1	3	2	Sí			Sí	Sí		Sí	"porque te ayuda a entender mas a fondo el concepto"
34	4	3	1	2	No			Sí	Sí		Sí	"por que de manera gráfica te muestra lo que hace el programa, de que parte avanza a otra función."

PARTE C. Contenido de la aplicación									
Califica del 1 al 5 de acuerdo a tu opinión (1 - muy mal, 2 - mal, 3 - regular, 4 - muy bien, 5 - excelente)									
ID	Los cubos contenedores con tapas móviles, en la aplicación, ayudan a comprender la declaración y asignación de variables	Las plataformas que contienen a los cubos contenedores, apoyan los conceptos del alcance global y local de las variables	Los diferentes colores de los cubos contenedores ayudan a identificar rápidamente los tipos de datos	La vista de árbol muestra de una forma intuitiva, paso a paso, el flujo de control (orden de las instrucciones) de la ejecución	El texto 3D que flota sobre las plataformas muestra el orden de la operaciones de una forma fácil de comprender	La animación de salto que realiza el personaje entre las plataformas de la aplicación, es una metáfora clara y sencilla de la llamada a una función y el retorno de los valores	¿En general, consideras que los conceptos fundamentales de la programación son representados de una forma intuitiva y fácil de comprender?	¿Por qué?	¿Qué otros temas de la programación que no se mostraron en la aplicación actual te gustaría que fuesen representados?
1	5	5	3	3	3	5	Sí		"estructuras, uniones, punteros, arreglos"
2	5	5	5	3	4	5	Sí	"porque te va marcando paso a paso el proceso de la función"	"punteros"
3	3	5	4	4	5	4	Sí	"simplemente se ve claro, siguiendo el código de la izquierda y viendo las acciones del muñeco"	"arreglos y punteros"
4	5	4	4	2	4	4	Sí	"porque vez paso a paso como hacen las operaciones los compiladores"	
5	4	3	4	2	4	5	Sí	"sí, porque facilita la comprensión de las diferentes variables."	"arreglos, punteros."
6	4	4	3	2	3	4	Sí	"porque se relaciona mucho el dibujo del mono con el código ayuda a comprender las cosas."	"muestra de errores."
7	5	4	5	4	4	5	Sí	"queda más clara la forma en como se ejecuta el programa y hace una comprensión mas clara de su funcionamiento al simple hecho de solo imaginar vagamente su funcionalidad en momento de ejecución"	"apuntadores, ciclos for, arreglos"
8	5	5	5	5	5	5	Sí	"visualmente es muy fácil de comprender, muy amigable con el usuario."	"punteros, arreglos de

									punteros"
9	4	5	4	3	4	5	Sí	"porque se indica paso a paso lo que va sucediendo"	"punteros, estructuras"
10	5	5	5	3	5	5	Sí	"muestra visualmente que esta realizando el programa."	"fgis, punteros, estructuras"
11	4	5	2	3	4	5	Sí	"tiene un buen entorno sencillo."	"seria interesante representar un error de sintaxis"
12	5	5	4	4	5	5	Sí	"con el hecho de que tenga animación incita a los usuarios a comprender la programación de una forma muy sencilla."	"punteros, paso de parámetro por referencia."
13	5	5	5	4	5	4	Sí	"porque se muestra y visualiza fácilmente las acciones que realiza cada parte del código."	"ciclos for, punteros y funciones."
14	5	5	5	5	5	5	Sí	"porque la manera de demostrar como ocurren los procedimientos es bastante explicativa."	"punteros, arreglos."
15	5	5	5	4	5	5	Sí	"te marcan los títulos y son muy visibles a la hora de ejecutar."	"estructuras y punteros"
16	5	5	5	3	5	5	Sí	"porque es una forma más fácil de comprenderlo"	"ciclo for to, punteros y estructuras"
17	5	4	4	5	4	5	Sí	"muestra paso a paso como funciona el programa"	"punteros, parámetros por referencia, estructuras."
18	4	4	5	4	4	5	Sí	"se da a entender facilmente, y los conceptos básicos son comprensibles, cumple con el objetivo principal."	"las clases"
19	5	5	4	5	5	5	Sí	"porque en el programa se muestran las variables y todo muy gráfico y explícito."	
20	5	5	4	4	5	5	Sí	"se entiende mas fácil"	"matrices"
21	4	4	5	5	4	3	Sí	"por que este proyecto fue muy divertido, bien explícito."	"estructuras"
22	5	5	5	5	5	5	Sí	"porque las interfaces gráficas hacen que se vea de manera más sencilla y comprensible el proceso que queremos representar."	"punteros y archivos."
23	4	5	3	4	4	5	Sí	"porque se representan claramente y dejan herramientas de manejo"	"punteros"
24	4	4	4	4	4	4	Sí	"asi graficamente es mas fácil y no hay aburrimiento"	

25	4	4	5	3	5	5	Sí	"ayudaría como es que se "mueven" las variables para tener en cuenta como son"	"como es en la memoria RAM."
26	3	3	4	2	5	3	Sí	"están en la forma gráfica y no son tan aburridos como cuando aparece la pura consola."	
27	5	5	5	5	5	5	Sí	"muestra paso por paso una metáfora sencilla de como se ejecuta el programa, creo que hay alumnos que se verán beneficiados con esa manera de verlo."	"quizás estructuras pero ya sería más complicado"
28	4	4	4	5	4	4	Sí	"porque con ayuda de la animación queda más claro el retorno de variables, las diferentes operaciones, aunque parece que explica más como trabaja el compilador"	"punteros y arreglos."
29	5	4	5	3	5	5	Sí	"es una forma clara de ver que es lo que hace un programa"	"punteros, arreglos, ciclos"
30	5	5	5	5	5	5	Sí	"la manera de graficar paso a paso no permite omitir detalles del proceso que se ejecuta."	"arreglos y creación de archivos."
31	4	4	5	3	4	5	Sí	"por la muestra de código y animaciones y el paso de plataforma dan una idea de lo que es el cambio o la llamada a una función."	"los tipos de estructuras con ; funcionalidad en código y gráfico"
32	4	5	5	5	4	5	Sí	"en esta aplicación es otra forma de como visualizar de donde y para donde salen los valores, si representa un avance en la forma didáctica de esta materia"	"cadenas, concatenar, copiar, etc, archivos"
33	5	5	5	4	5	5	Sí	"de hecho con la animación en 3D ayuda al usuario comprender más la función de cada línea de código que ponga"	"arreglo de estructuras."
34	5	5	5	5	5	5	Sí	"porque maneja lo más básico de programación."	"la agregación del for."

PARTE D. Funcionalidad de la aplicación									
Califica del 1 al 5 de acuerdo a tu opinión (1 - muy mal, 2 - mal, 3 - regular, 4 - muy bien, 5 - excelente)									
ID	El tiempo de carga inicial de la aplicación es aceptable	Las animaciones 3D se visualizan correctamente y de una forma fluida	El manejo de la cámara 3D es útil y fácil de utilizar	Las opciones de pausar la ejecución y ajustar la velocidad son muy útiles, y funcionan correctamente	El editor de código fuente es fácil de utilizar y el resaltado de la sintaxis es muy útil	Es muy fácil interactuar con los datos de entrada y salida durante la ejecución	En general, ¿consideras que las opciones incluidas en la aplicación son suficientes, útiles y fáciles de usar?	¿Por qué?	¿Qué otras opciones o funcionalidades que no se encuentran disponibles actualmente en la aplicación te gustaría que fuesen incluidas?
1	5	5	3	2	3	3	No	"no son suficientes, hay temas que se pueden agregar: punteros, ciclos infinitos, estructuras, uniones, arreglos"	"mapas de memoria RAM, el muñequito podría ir a mapas de memoria, es decir tener abiertos 2 ambientes, tanto el actual (world) como el de memoria"
2	5	5	5	5	5	5	Sí	"para empezar a programar no se necesita más que los programas de eje."	"el zoom en el diagrama, que no se tenga que inicializar a cero cada variable"
3	5	5	5	5	4	5	Sí	"porque con lo que contiene hasta ahorita funciona bien."	
4	4	5	5	5	4	3	No	"pienso que le falta algun instructivo o por ejemplo en la opcion de velocidad que diga ("velocidad")"	"sonido e instrucciones asi como que diga que para declarar variables hay que inicializarlas"
5	4	5	5	5	5	5	Sí	"aparte de enseñar las variables se visualiza en donde se guarda cada variable"	"que hubieran ejemplos y que incluyera los temas de arreglos y punteros"
6	4	4	2	4	2	3	Sí	"porque con estas se vizualiza el código y su ejecución a la vez."	"errores, los ciclos, la cámara del árbol."
7	5	5	5	5	5	5	Sí	"mayor comprensión de como funcionan realmente."	"ajustar la velocidad en la opción visual del árbol, ajustar el zoom en la opción visual de árbol"
8	5	4	5	5	5	5	Sí	"son buenas y fáciles de entender"	"más variedad de ejemplos y algun espacio para prototipos de funciones y ejemplos de su aplicación."
9	4	4	4	5	5	5	Sí	"nos da una mejor idea de la secuencia y ejecución del"	"cuando se edite el código que indique que errores tengo"

								programa"	
10	5	4	5	5	3	4	Sí	"modificar el código y los menus, porque le falta depuracion en algunas partes"	"mostrar mensajes de error"
11	5	5	5	5	4	5	Sí	"tiene lo basico que suele ser complicado de entender al principio"	"podrian agregar un tutorial o una lista que indique que representan los colores"
12	5	3	4	4	4	4	Sí	"no tienen dificultad alguna, las funciones utilizadas son muy faciles de utilizar para los usuarios."	
13	5	5	3	3	2	4	Sí	"son muy fáciles de utilizar y de gran utilidad. Para comprender lo básico de la declaración de variables."	en el ciclo while revisar los ciclos infinitos, que se realicen y también errores de sintaxis como los dos puntos :"
14	5	5	5	5	5	5	Sí	"maneja las bases de la programación"	"podria mejorarse la cámara de la interfaz 3D, ya que esta se centra automaticamente al cambiar de plataforma."
15	4	4		5	5	5	Sí	"si para los ejemplos utilizados son suficientes."	
16	5	5	5	5	5	5	Sí	"tiene lo básico que debes saber de C de una manera practica."	"que el diagrama de árbol sea mas lento al correr"
17	5	5	5	4	3	5	Sí		"mejor control en la velocidad en Tree"
18	4	5	5	5	4	5	Sí	"porque, tiene las aplicaciones básicas y necesarias para visualizar un proyecto de manera didactica"	"otro modo de visualización, por ejemplo una opción que te lo fuera explicando de manera hablada."
19	5	5	5	5	5	5	Sí	"porque te lo explica muy bien en las funciones, ciclos, etc."	
20	4	5	3	5	5	5	Sí	"es sencillo"	
21	4	5	4	4	4	4	Sí	"porque puede uno cambiarle al codigo y click hace el programa"	"todo esta muy bien fuera genial poner banderas"
22	4	5	5	5	5	4	Sí	"no son tantas funciones, más cumple con las básicas y su representación es excelenter."	"punteros."
23	4	5	5	5	4	5	Sí	"son manejables"	"error de sintaxis"
24	4	4	4	4	4	4	Sí		
25	5	5	4	4	5	5	Sí	"ayuda a la adaptación de cada uno."	"fuera un poco más grande la pantalla móvil."
26	4	3	4	3	4	3	No	"hace falta agregar un poco mas de códigos."	

27	5	5	5	5	5	4	Sí	"está muy fácil de comprender y de usar, lo único podría ser un poco más intuitivo son los botones de edit y play"	"una forma de explorar la totalidad de el árbol"
28	4	5	5	5	5	4	Sí	"útiles y facil de usar si, pero no suficientes"	"esta muy bien representado por el personaje, solo cuando muestra los mensajes no se entiende bien"
29	2	4	4	5	5	5	Sí	"pues como es didáctico, muestra lo suficiente para su proposito."	
30	5	5	5	5	5	5	No	"creo que debería de tener la opcion de visualización de un cierto segmento del codigo o poder regresar sin reiniciar."	"la opcion de generar un nuevo código sin editar lo ejemplos existentes."
31	3	5	5	5	4	4	No	"faltan algunas restricciones en cuanto a codigo que envien señales de errores y posibles soluciones"	"editar al paso en que la animación pueda estar señalando la función o codigo modificado (seria muy bueno)."
32	4	5	5	5	5	5	Sí	"se muestra una interface para el usuario fácil de visualizar y sin tantos botones para ejecutarlo"	
33	5	5	5	5	5	5	Sí	"una forma mas interesante de ver de manera lógica lo que hace el software."	"una miniguia para aquellos que no vieron nada de programación lo que son los tipos de variables, funciones, etc."
34	5	5	5	5	5	5	Sí	"porque es lo mas útil que se tiene que usar. El compilador no requiere más"	

PARTE E. Diseño de la aplicación								
Califica del 1 al 5 de acuerdo a tu opinión (1 - muy mal, 2 - mal, 3 - regular, 4 - muy bien, 5 - excelente)								
ID	El diseño gráfico de la aplicación y la organización de los elementos, ayuda a localizar las opciones fácilmente	Los colores de la interfaz son agradables para el usuario y logran un contraste adecuado entre los elementos	Los diseños de los modelos 3D, formas y colores, ayudan a diferenciar rápidamente los conceptos de la programación	Las animaciones de los modelos 3D en la aplicación, permiten apreciar fácilmente las acciones efectuadas	El tipo y tamaño de letra utilizados en la aplicación, facilitan la lectura del código en el editor y del texto de las opciones	¿Tuviste algún problema o error al manejar la aplicación?	¿Cuáles?	Sugerencias y comentarios
1	5	5	5	3	5	Sí	"manejar ciclos infinitos"	
2	5	5	5	5	5	No	"aunque al meter un error a proposito no te muestra cual es"	
3	5	3	4	4	4	No		
4	5	5	4	5	5	Sí	"solo que el programa (página) no dice si hay error en el código"	
5	5	5	5	5	5	No	"ninguno"	"solo que se incluyan más ejemplos, para de ahí basarnos para hacer ejercicios y poder entender mejor la programación"
6	4	4	3	3	3	Sí	"al editar el código, es muy sensible a errores, y al ver el árbol."	"ayudar en la detección de errores, o mostrar un mensaje de error, mejorar la vista de su cámara del árbol. Hacer especificaciones del código."
7	4	5	5	5	5	No		"sería bueno el tener la opción de nosotros editar el código y que funcionará al igual que manejará ejemplos un poco más complejos de nivel un poco más alto. En general es una buena aplicación para la personas a la que se nos dificulta un poco la comprensión del código"
8	5	5	5	5	5	No	"ninguno"	"añadir más funciones al programa"
9	5	5	4	5	5	Sí	"no pude utilizar ciclos de for y variables se tienen que inicializar por fuerza"	
10	4	5	3	2	1	No		
11	5	5	3	4	4	Sí	"detectar errores en el código"	

12	4	3	3	4	4	No		"tal vez cambiarle el color a los printf y scanf en la parte 3D"
13	5	4	5	5	5	No		
14	5	5	5	5	5	No		
15	4	5	4	4	5	No		
16	5	5	5	5	5	No		
17	5	4	5	5	5	No		
18	5	5	5	4	5	No	"ninguna."	
19	5	4	5	4	5	No		
20	5	4	5	5	5	No		
21	4	4	5	4	5	No		
22	4	4	5	5	5	No		
23	4	4	4	4	4	No		
24	4	4	4	4	4	No		
25	5	5	5	5	5	No		"ninguno :)"
26	4	4	4	4	4	No		
27	4	5	5	5	5	Sí	"intenté editar el código cuando la función de editar estaba desactivada"	"me parece una aplicación muy brillante y útil."
28	4	5	5	5	5	Sí	"al inicializar muchas variables tarda más"	
29	5	5	4	5	5	No		
30	5	3	4	4	3	No		"solo podría mejorar un poco la calidad visual hasta un punto donde no afecte el desempeño, el cual es excelente."
31	4	5	4	5	5	No	"ninguno"	
32	4	5	5	5	5	No		
33	4	3	5	5	4	No		
34	5	5	5	5	5	No		