Desarrollo de un laboratorio virtual interactivo para la enseñanza de ciclos repetitivos en programación en los primeros semestres de Ingeniería de Sistemas

Harrizon Alexander Soler Galindo

Cod. 2281137

Universidad Santo Tomás Seccional Tunja

Ing. Juan Francisco Mendoza Moreno, PhD

16 de diciembre de 2024

Índice

Ficha técnica del proyecto

Título	Desarrollo de un laboratorio virtual interactivo para la enseñanza de ciclos repetitivos en programación en los primeros semestres de Ingeniería de Sistemas
Nombre estudiante	Harrizon Alexander Soler Galindo
Autor	1002397733
Director	Ing. Juan Francisco Mendoza Moreno, PhD
Palabras claves	gamificación, laboratorio virtual, godot, videojuegos, programación

Descripción

Este trabajo describe el diseño e implementación de Lemulogic, un lenguaje de programación interactivo diseñado para la enseñanza de bucles y condicionales a estudiantes de ingeniería de primeros semestres en la Universidad Santo Tomás. El laboratorio proporciona un entorno accesible vía navegador que emplea un reto en forma de minijuego para fomentar la práctica. Los hallazgos clave destacan una mejora en la comprensión de los estudiantes, con un aumento del 17 % en las calificaciones y altos niveles de satisfacción estudiantil. El estudio concluye que estas herramientas virtuales mejoran los resultados de aprendizaje y la participación, ofreciendo un enfoque escalable e innovador para la educación en programación, con posibles expansiones en funciones de programación avanzadas y multijugador.

Planteamiento del problema

La escasez de entornos de laboratorios virtuales para estudiantes de primeros semestres en ingeniería de sistemas (y afines) representa un reto significativo para su desarrollo en el aprendizaje y habilidades de conceptos fundamentales en programación (Shahid et al., 2019). Durante su introducción a esta disciplina los estudiantes enfrentan una curva de aprendizaje significativa, exacerbada por la falta de experiencias accesibles. Incluso tras dos años de formación académica, la mayoría de los desarrolladores principiantes no se considerarían competentes en el área (Maiga et al., 2019).

Otras deficiencias detectadas con la falta de laboratorios virtuales fueron:

- Algunos estudiantes tienen problemas de motivación y encuentran la programación demasiado difícil (Zhan et al., 2022).
- Las actividades pobremente diseñadas que obstaculizan el aprendizaje.
- Falta de esfuerzo intelectual, razonamiento lógico y habilidades para resolución de problemas (Shahid et al., 2019).
- Dificultades en las habilidades pedagógicas de los educadores (Pinto & Terroso, 2022).
- Deserción del curso o carrera por parte de los estudiantes, debido a la dificultad entendiendo fundamentos de programación (Santamaría & Valentina, 2023)
- Disparidades en las experiencias de aprendizaje entre los estudiantes de modalidades presencial y virtual, particularmente en casos de recursos limitados.

La implementación de laboratorios virtuales representa un desafío para las Instituciones de Educación Superior (IES), en sus procesos de enseñanza y aprendizaje. La implementación requiere un esfuerzo en términos tecnológicos y pedagógicos, además de la inversión económica, considerando la contratación de proveedores externos.

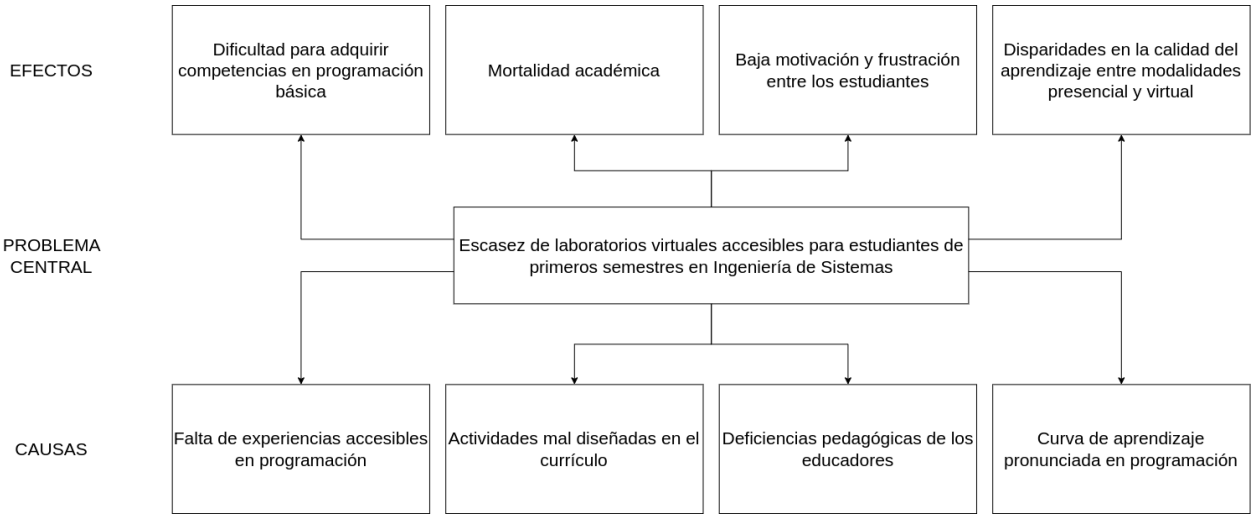
Actualmente, la Universidad Santo Tomás Seccional Tunja ha identificado las mismas falencias en su programa de Ingeniería de Sistemas en su modalidad presencial y esperan solucionarse para la inclusión de modalidad virtual.

Para los estudiantes se hacen necesarias estrategias innovadoras en el aprendizaje de programación, que conviertan conceptos complejos en conocimientos comprensibles y actividades prácticas (Zhan et al., 2022).

Árbol de problemas

Figura 1

Árbol de problemas



Justificación

El programa de Ingeniería de Sistemas de la Universidad Santo Tomás, Seccional Tunja, enfrenta desafíos importantes en la enseñanza de los fundamentos de la programación a los estudiantes de primer semestre. Estos desafíos reflejan problemas más amplios, incluyendo deserción, falta de motivación, y dificultades para desarrollar habilidades de pensamiento lógico.

El presente estudio evidencia el diseño e implementación de un laboratorio virtual de programación, para abordar los desafíos educativos en el programa de Ingeniería de Sistemas en la universidad Santo Tomás y apoyar la transición a una modalidad virtual. El laboratorio virtual representa una innovación educativa, que busca complementar y apoyar los métodos tradicionales de enseñanza con un enfoque practico e interactivo que mejora las habilidades de resolución de problemas y codificación para los estudiantes.

Se espera que la implementación de un laboratorio virtual en la Universidad Santo Tomás aporte resultados positivos, como un mayor compromiso de los estudiantes, mejora de las capacidades de resolución de problemas y una enseñanza más eficaz en los cursos de programación.

Esta iniciativa se alinea con el compromiso de la Universidad Santo Tomás por la excelencia e innovación educativas, abordando los retos presentes al tiempo que se prepara para los futuros panoramas educativos.

Objetivos

Objetivo general

Facilitar la comprensión y aplicación de ciclos repetitivos en programación, mejorando el proceso de aprendizaje de estudiantes de Ingeniería de Sistemas en los primeros semestres, mediante el desarrollo e implementación de un laboratorio virtual interactivo, accesible desde cualquier lugar y en cualquier momento.

Objetivos específicos

Cuadro 1

Objetivos específicos

Nro,	Objetivo específico
1	Diseñar una aplicación de navegador accesible y entretenida para el laboratorio virtual, permitiendo a los estudiantes de primer semestre de Ingeniería de Sistemas el acceso a los recursos educativos simulados sobre ciclos repetitivos de programación de manera autónoma.
2	Desarrollar ejercicios prácticos y casos de estudio que aborden diferentes aspectos y aplicaciones de los ciclos repetitivos en programación, con el fin de fortalecer las habilidades de resolución de problemas y la comprensión conceptual de los estudiantes, para ser implementado dentro del laboratorio virtual.
3	Evaluar el impacto del laboratorio virtual en el rendimiento académico y la satisfacción estudiantil, mediante la recopilación de datos cuantitativos y cualitativos, en colaboración con docentes y estudiantes del programa de Ingeniería de Sistemas.

Metodología

El siguiente estudio utiliza un enfoque de investigación exploratoria para diseñar e implementar el laboratorio virtual Lemulogic. Este enfoque se considera adecuado para abordar nuevos retos y oportunidades de los entornos virtuales en la enseñanza de programación. Consta de dos etapas una etapa de desarrollo para el diseño y la implementación, y una etapa de investigación que utiliza métodos cuantitativos y cualitativos para evaluar el rendimiento y experiencia del usuario.

Etapa de desarrollo

Esta fase sigue un enfoque iterativo guiado por la norma IEEE 29148 para la especificación de requerimientos de software. El diseño inicial se fundamenta en una evaluación sobre las necesidades del estudiante. Se prioriza la facilidad de acceso y experiencia de usuario, garantizando el acceso autónomo a los recursos disponibles.

El diseño del laboratorio virtual es evaluado desde un enfoque humano a través del marco de gamificación Octalysis, el cual evalúa ocho dimensiones de motivación humana (Chou, 2015).

Figura 2

Marco de gamificación Octalysis

Nota. Tomado de The Octalysis Framework, por Yu-kai Chou, 2012

Así mismo el diseño toma en cuenta las siguientes directrices de gamificación para la enseñanza de programación (Shahid et al., 2019)(Hurtado et al., 2021)(Pinto & Terroso, 2022):

- Enfatizar en las mecánicas y dinámicas de la experiencia.
- Implementar retroalimentación y retos.
- Ofrecer recompensas relevantes.

La implementación sigue una metodología ágil, con ciclos iterativos de implementación y pruebas, la cual adopta prácticas de Extreme Programming (XP), enfatizando una retroalimentación continua entre desarrollador y tutores, así como la entrega periódica de pequeños incrementos funcionales. Las siguientes prácticas elegidas para la implementación del laboratorio están basadas en recomendaciones para proyectos individuales, específicamente proyectos académicos de software (Akpata & Riha, 2004):

- Incrementos pequeños y funcionales.
- Ejecución de pruebas (unitarias, integración, funcionales, etc) por incremento
- Inspecciones regulares de refactorización.
- Seguimiento de prácticas de desarrollo validadas en la industria.
- Uso de analogías o metáforas simples en el diseño de sistemas.

Los ciclos iterativos incluyen la implementación de pruebas unitarias para incrementar la calidad, estabilidad y flexibilidad del código.

Etapa de investigación

La investigación se realizó con una muestra de 30 estudiantes de primer semestre del programa de Ingeniería de Sistemas de la Universidad Santo Tomás, Seccional Tunja. Los participantes de dividen en dos grupos, cada uno con una sesión de laboratorio virtual de dos horas. Para la recolección de datos, se emplean dos técnicas: una prueba de conocimientos y un formulario de satisfacción:

La prueba de conocimientos compuesta de 6 preguntas de opción múltiple enfocadas en bucles y condicionales, es realizada tanto antes como después de la sesión de laboratorio

virtual. Su posterior análisis cuantitativo compara los resultados pre y post prueba para medir el progreso en el aprendizaje, estos resultados se combinan para los dos grupos ya que no es relevante su diferenciación. El análisis cuantitativo de los resultados se lleva a cabo mediante tabulación de los datos para el contraste de estos.

Al final de cada sesión se lleva a cabo un formulario de satisfacción de seis preguntas (tres de calificación, tres abiertas) que cubre la facilidad de uso, calidad del contenido, utilidad percibida y satisfacción general. El análisis cualitativo incluye la revisión de las respuestas con un enfoque temático para identificar patrones, opiniones y sugerencias (de las cuales una porción es implementada en el laboratorio), destacando aspectos positivos y áreas de mejora.

Esta metodología proporciona un enfoque integral para evaluar el impacto del laboratorio virtual en el aprendizaje y la satisfacción de los estudiantes, combinando un análisis cuantitativo con una exploración cualitativa.

Resultados

El prototipo resultante es un lenguaje de programación llamado Lemulogic diseñado específicamente para facilitar la interiorización en los conceptos de bucles y condicionales en los estudiantes de primer semestre de la universidad Santo Tomás de Tunja, para lograrlo Lemulogic cuenta con:

- Acceso desde navegador web de escritorio.
- Entorno de desarrollo basado en nodos y conexiones.
- Detección de errores sintácticos.

Para motivar al usuario final en el desarrollo de algoritmos, se presenta un sistema de "Minijuegos" en forma de experiencias desarrolladas para ejecutar los algoritmos creados por el usuario.

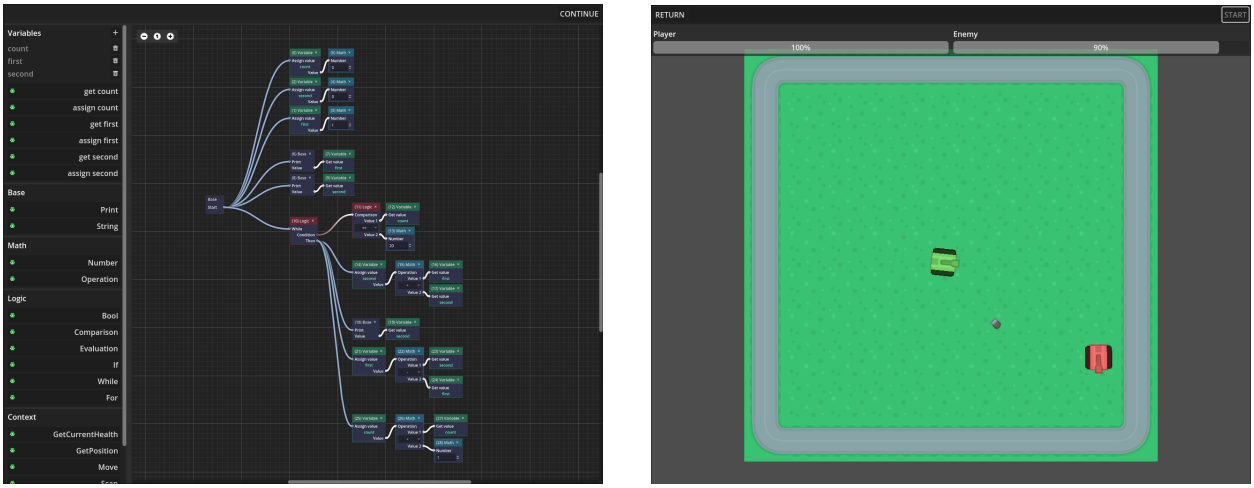
A la fecha de 16 de diciembre de 2024 es posible acceder al prototipo el cual provee funcionalidad completa del editor con minijuego en tematica de tanques, a través de la URL https://harrizonsoler.github.io/lemulogic/.

La interfaz principal de Lemulogic está basada en un canvas infinito para el desarrollo de algoritmos a través de nodos y conexiones, reflejando una estructura de grafo la cual crea una imagen mental más efectiva y ordenada para el usuario. El diseño visual de grafo está basado en las implementaciones del editor de materiales en Blender, y el grafo de eventos en Unreal Engine.

La sintaxis del lenguaje está basada en la filosofía de la lengua artística Toki Pona, la cual intenta expresar la mayor cantidad de significados con un vocabulario reducido (120 palabras). En Lemulogic se busca maximizar la funcionalidad de programación, condensando las herramientas en un conjunto limitado de nodos, sin comprometer la capacidad de crear programas sofisticados.

Figura 3

Editor de Lemulogic con algoritmo de sucesión de Fibonacci y ejecución de minijuego

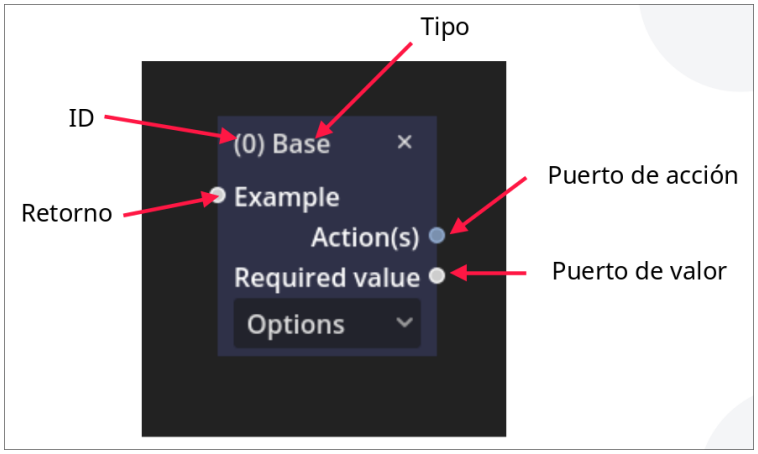


Todos los nodos del editor se componen de seis partes principales:

- **ID:** Número identificador único para facilitar la localización de errores al usuario.
- **Tipo:** El tipo o grupo del nodo se diferencia por el título en la cabecera del nodo, así como su color que comparten los nodos del mismo tipo.
- **Puerto:** Actúan como entrada o salida de información según su tipo: puertos de acción o valor que cuentan con un color especifico (azul para acción, blanco para valor).
- **Retorno:** Es un puerto con las diferencias de que es de salida y es el único que se encuentra a la izquierda del nodo.
- **Inputs:** Son contenedores de cualquier tipo que alojan información fija la cual es ingresada por el usuario, ya sean cajas de texto, listas de opciones, etc.

Figura 4

Partes de un nodo



Como cualquier lenguaje de programación, Lemulogic cuenta con reglas específicas de sintaxis y flujo, minimizando la posibilidad de errores irrecuperables en la ejecución:

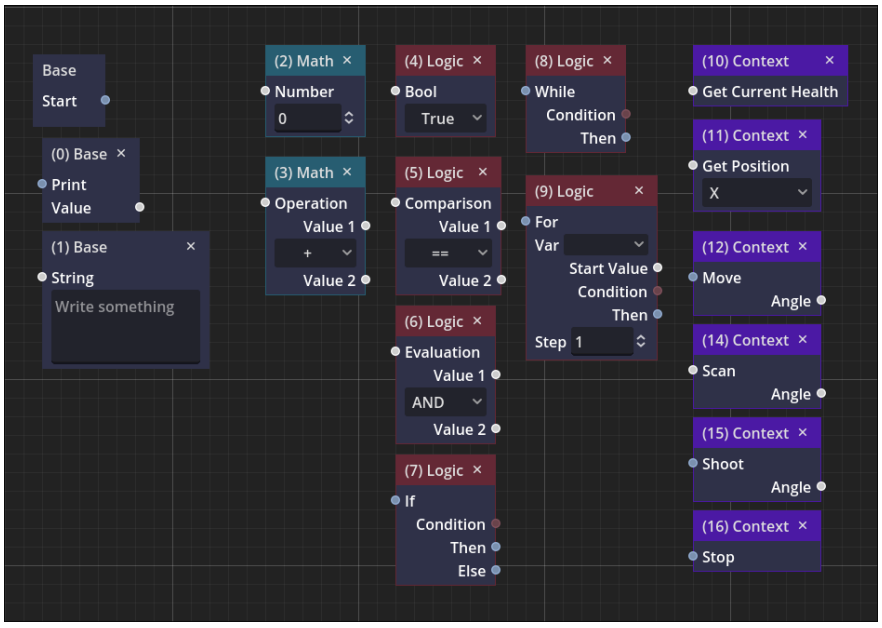
- Los nodos conectados a un puerto de acción son ejecutados en un orden de arriba hacia abajo.
- A todas las variables declaradas y usadas en el algoritmo deben serles asignado un valor al principio del algoritmo.
- Ningún puerto de un nodo puede conectarse a sí mismo u otro puerto del mismo nodo.
- No se ejecutará ninguna expresión que no esté conectada al nodo de "Start" o "Inicio".
- Las conexiones y valores en todos los nodos conectados al nodo "Start" deben ser validos (no es válido por ejemplo puertos no opcionales de nodos sin conectar o valores inválidos en los inputs de los nodos).

El editor de programación presenta al usuario un abanico de nodos (véase Figura 5) al servicio del usuario para el desarrollo de sus algoritmos, en el cual se pueden contemplar los siguientes tipos:

- **Lógica:** Estos nodos permiten la toma de decisiones en el algoritmo, así como controlar el flujo de ejecución basado en condiciones.
- **Matemáticas:** Usados para llevar a cabo operaciones matemáticas básicas y la manipulación de valores en el algoritmo.
- **Variables:** Nodos con la única función de asignar y definir variables para ser usadas a lo largo de la ejecución.
- **Bucles:** Permiten la ejecución repetida de código sobre la ejecución del algoritmo.
- **Entorno:** Este tipo de nodos leen y modifican el entorno del minijuego según las limitaciones asignadas (la asignación de nodos por el desarrollador del minijuego).

Figura 5

Canvas del editor Lemulogic con todos los nodos disponibles



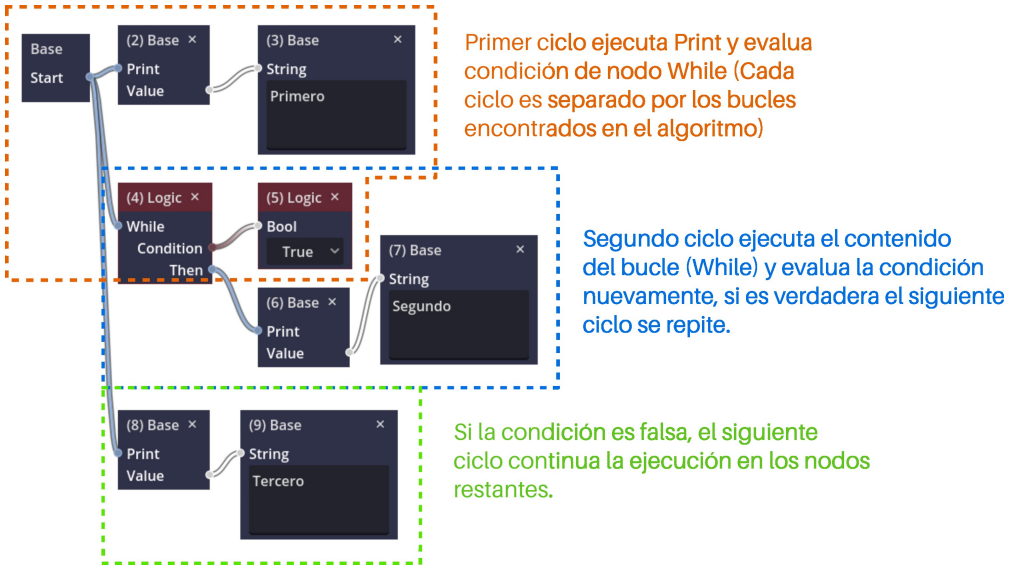
Teniendo en cuenta que el enfoque del laboratorio virtual se encuentra en los bucles y condicionales, el flujo del lenguaje de programación esta especialmente diseñado para atraer al usuario a experimentar con el comportamiento de este tipo de expresiones (If, While, For, etc).

El objetivo principal en la implementación del lenguaje es aprovechar el flujo de un ciclo infinito o extenso para crear comportamientos más complejos y detallados a la hora de ser ejecutados en el minijuego, para esto se diseñaron dos estrategias:

- Cada ciclo del interprete se dispara una sola vez por cada fotograma en el minijuego, para controlar los ciclos infinitos y no bloquear la ejecución.
- Los ciclos están separados por las condiciones de un bucle en el algoritmo. por ejemplo, si un algoritmo no tiene bucles, se ejecuta en un solo ciclo y termina. Por el contrario, si el intérprete encuentra un ciclo, evalúa su condición y si es verdadera, termina el ciclo y el próximo inicia ejecutando el ciclo, como es ejemplificado en la Figura 6.

Figura 6

Ejemplo de un flujo de ejecución con bucle



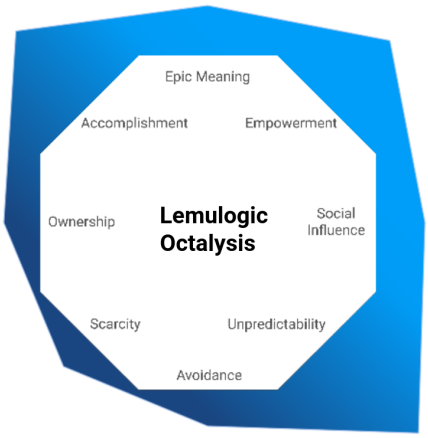
Etapa de desarrollo

La especificación de requerimientos diseñada para el laboratorio virtual se encuentra en el Anexo al final del documento, que sigue la norma IEEE 29148, en la cual se

especifican los casos de uso, sus requerimientos tanto funcionales y no funcionales, entre otros detalles. Esta especificación fue usada para la gestión del cronograma, implementación y pruebas para el laboratorio virtual. El documento se encuentra sujeto a cambios significativos en posteriores funcionalidades planeadas.

Figura 7

Visualización Octalysis de Lemulogic



La Figura 7 condensa el análisis lúdico del laboratorio virtual, basado en el marco de diseño Octalysis. El laboratorio virtual, siendo un lenguaje de programación con una motivación en vencer las metas de los minijuegos, resalta sus fortalezas en las dimensiones de motivación intrínsecas de Empoderamiento e Impredecibilidad, ya que consiste en el disfrute de programar un algoritmo de forma iterativa y experimental para cumplir el o los objetivos del minijuego. El marco también resalta las carencias en las dimensiones de Influencia Social, Significado Épico y Posesión, lo que indica metas a cumplir en futuras versiones del laboratorio, como funcionalidades multijugador y persistencia multisesión para los usuarios.

El desarrollo del laboratorio virtual sigue la metodología Extreme Programming (XP) la cual prioriza software funcional sobre documentación, promueve entregas frecuentes, minimalismo, comunicación directa y es adaptable a equipos de variados tamaños. El proceso llevó un periodo de 6 meses en iteraciones bisemanales con reuniones

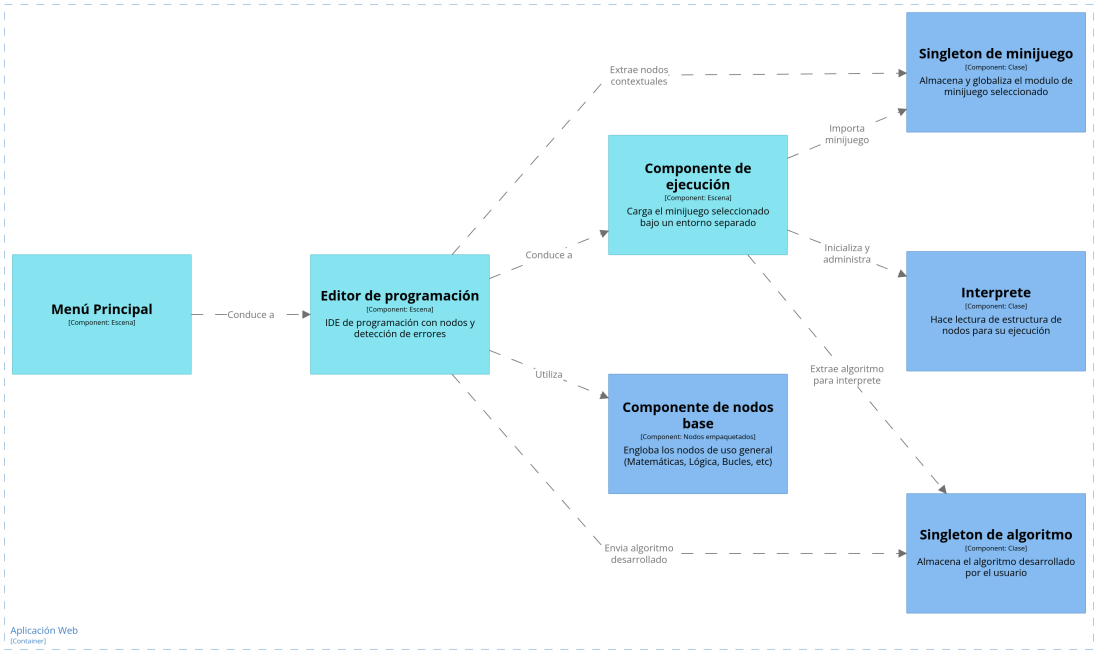
que tienen entregas planificadas por casos de uso individuales, así mismo, el seguimiento del proceso fue apoyado por pruebas unitarias.

Se utilizó GitHub como plataforma para alojar y gestionar el código del laboratorio virtual, asegurando control de versiones y acceso centralizado para futuras mejoras. El repositorio se puede encontrar en la URL https://github.com/HarrizonSoler/lemulogic Además, GitHub facilita la documentación técnica, promoviendo la transparencia, reproducibilidad y adaptabilidad del proyecto en contextos educativos con el servicio de GitHub Pages.

Arquitectura

Figura 8

Arquitectura a nivel de componentes



[Component] Lemulogic - Aplicación Web
Diagrama de componentes para la aplicación web
Thursday, November 14, 2024 at 5:02 AM Coordinated Universal Time

La Figura 8 presenta la arquitectura diseñada para el laboratorio virtual siguiendo

el modelo C4 para la visualización de arquitecturas de software.

Como entorno de ejecución se consideró el motor de videojuegos Godot Engine, principalmente por su permisible licencia MIT y su ecosistema de librerías que facilitan el desarrollo. El motor permite elegir entre varios lenguajes de programación como GDscript (para exportación a navegadores), C#, etc. Las funcionalidades y plataformas que ofrece se alinean con los requerimientos del proyecto usando el lenguaje GDScript.

La aplicación sigue el patrón de Arquitectura de Buses, ya que facilita la comunicación entre escenas del sistema, principalmente en el transporte de información sobre el minijuego seleccionado y el algoritmo desarrollado. Los componentes de la aplicación están diseñados alrededor de escenas principales que hacen uso de clases específicas para su funcionamiento,

El editor de programación consiste en un canvas que permite la conexión, creación y modificación de nodos. Los nodos de tipo contextual son extraídos del Singleton de Minijuego, el cual contiene la lista de nodos definida para el minijuego establecido.

El editor cuenta con la funcionalidad de verificación de errores disparada antes de continuar a la ejecución, el mismo escanea de forma recursiva la estructura del algoritmo en busca de dos tipos de errores:

- Variables declaradas sin definir un valor.
- Puertos no opcionales sin conectar.

El componente de Ejecución administra el intérprete según el estado de la partida, si el minijuego avisa que la partida ha terminado (por ejemplo si el jugador ha ganado o perdido) se finaliza la ejecución del intérprete, o si el jugador retorna al editor el interprete es reiniciado.

El funcionamiento del Interprete consiste en múltiples clases y estructuras con responsabilidades individuales, la estructura de este se basa en el patrón de interprete el cual es usado para ejecutar scripts de un lenguaje integrado en la aplicación; Como

beneficio este patron mantiene la lógica del minijuego y del editor separados del interprete, haciendo la base de codigo modular y mantenible.

La clase principal del interprete lleva el estado de los ciclos, como el nodo actual y el siguiente a ejecutar. La responsabilidad de comunicar y orquestar los nodos es de la clase "RoutineContext", la cual tiene la funcionalidad de ejecutar nodos de forma recursiva y facilitarles los valores retornados por otros nodos y de las variables.

El sistema para obtener un sistema de carga de minijuegos modularizado consiste en el empaquetamiento de estos a una escena desarrollada en el mismo framework el cual es cargado por el componente Singleton de minijuego el cual es visualizado a través de un entorno modularizado, aparte de la escena visual cumple con proveer una lista de nodos (conectados a su respectivo efecto o acción en la escena) para ser usados por el editor y el intérprete respectivamente.

Cabe recalcar que la ejecución del minijuego funciona de manera asíncrona a la del interprete (véase Figura 9), por lo que la finalización en la ejecución de un algoritmo no detiene la del minijuego, en consecuencia da al desarrollador total libertad en el funcionamiento de su minijuego, donde el intérprete actúa como una interfaz de eventos hacia el minijuego que se esté ejecutando.

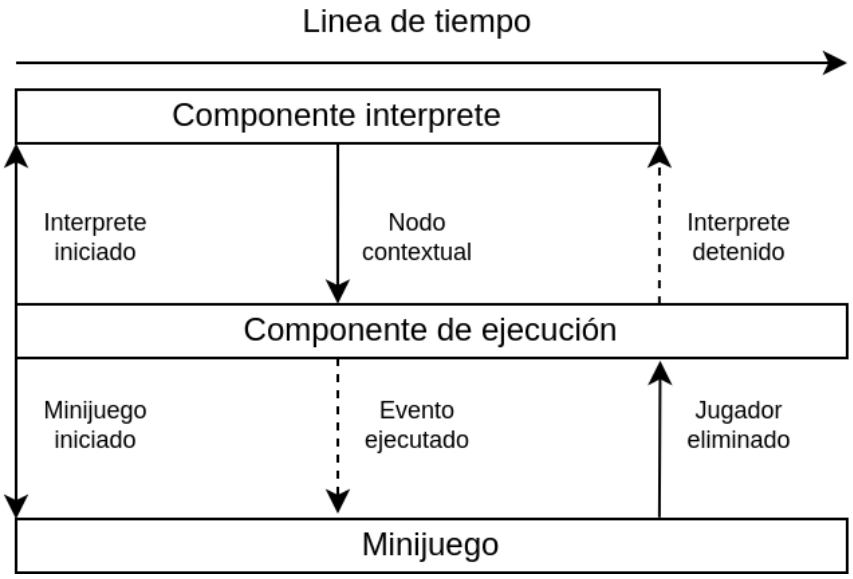
Una de las estructuras de datos más importantes de la aplicación es "Routine", la cual empaqueta todos los nodos, conexiones y variables de un algoritmo. Dicha estructura es creada por el editor de programación y leída por el intérprete por medio de un componente Singleton que globaliza y establece la estructura para toda la sesión.

A pesar de no existir la funcionalidad, el diseño de la estructura facilita una posterior implementación de multijugador, ya que siendo un conjunto de arreglos planos y HashMaps, es fácilmente serializable a formato JSON para el intercambio de esta información entre jugadores.

Así mismo la estructura contiene lógica utilitaria para la localización de nodos y conexiones por medio de listas HashMap que ofrecen un rendimiento favorable para

Figura 9

Flujo de ejecución del minijuego



acomodarse a las necesidades de velocidad en los ciclos del interprete (60fps o 16 milisegundos por ciclo).

Pruebas

El desarrollo de pruebas unitarias es implementado y documentado en el código de todos los nodos, para asegurar un funcionamiento determinístico en el flujo y comportamiento de los algoritmos desarrollados por el usuario.

Cada nodo implementa una función heredada de la clase “RoutineNode” ya sea `execute` o `get_value` según el tipo de retorno del nodo; Estas funciones reciben el contexto del interprete para extraer valores o ejecutar otros nodos así como extraen datos de los Inputs (véase partes del nodo). En estas funciones se implementan las pruebas unitarias; Estas son ejecutadas de manera local y desde el editor Godot, usando el complemento Gut para el desarrollo de pruebas unitarias en GDScript.

Las siguientes tablas exponen los resultados obtenidos en las pruebas sobre cada nodo:

Cuadro 2: Resultados en pruebas de nodos

Nombre	**Descripción**	**Resultado Esperado**	**Estatus**
`Start_MultipleConnections`	Ejecuta nodos de acción conectados al nodo Start	Ejecuta las conexiones en el orden esperado	PASA
`Print_StringInput`	Imprime valor String recibido	Imprime valor en la consola del navegador	PASA
`String_BasicValue`	Crea string con texto básico	Debe retornar el valor almacenado en nodo	PASA
`Number_Integer`	Crea valor entero	Debe retornar valor exacto	PASA
`Operation_Addition`	Suma dos números	Debe retornar valor sumado correctamente	PASA
`Bool_True`	Crea valor verdadero	Debe retornar booleano Verdadero	PASA
`Comparison_Equals`	Compara valores iguales	Debe retornar booleano Verdadero	PASA
`Evaluation_AND`	Evalúa operación AND	Debe retornar booleano Verdadero	PASA

Continued on next page

Cuadro 2: Resultados en pruebas de nodos (Continued)

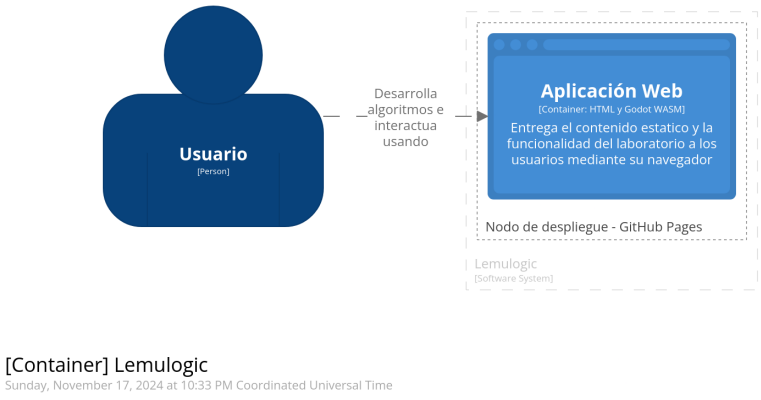
Nombre	Descripción	Resultado Esperado	Estatus
`If_TrueCondition`	Ejecuta expresiones con condición Verdadera	Debe ejecutarse contenido	PASA
`While_FixedIterations`	Ciclo repetido una cantidad especifica	Debe ejecutarse iteraciones exactas	PASA
`For_StandardLoop`	Ciclo de rango especifico	Debe completar todas las iteraciones	PASA

Despliegue

El despliegue de la aplicación web se lleva a cabo por el servicio de GitHub Pages, el cual facilita un despliegue directo desde el repositorio en cuestión, así como lanzamientos inmediatos en cambios al repositorio.

Figura 10

Arquitectura a nivel de sistema



[Container] Lemulogic
Sunday, November 17, 2024 at 10:33 PM Coordinated Universal Time

Para desplegar el proyecto en cualquier modificación al código, inicialmente el

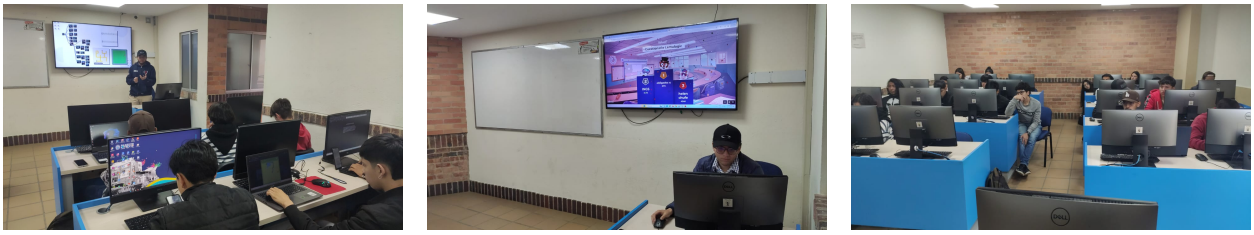
proyecto es exportado a un ejecutable web en una carpeta llamada `doc` para ser compatible con el servicio GitHub Pages. Posteriormente los cambios son aplicados al repositorio remoto y se redespliega la aplicación desde la página del repositorio con los cambios ya subidos.

Etapa de investigación

Para evaluar el impacto del laboratorio virtual en el aprendizaje de los estudiantes, se aplicó un diseño de pre-test y post-test en una única sesión con dos grupos de 25 estudiantes de primer semestre de Ingeniería de Sistemas cada uno. La prueba se centró en evaluar la comprensión de los estudiantes de los ciclos de programación repetitivos, incluidos los bucles y las sentencias condicionales.

Figura 11

Fotografías de sesiones del laboratorio virtual



El cuestionario aplicado consta de 6 preguntas elaboradas en la plataforma Kahoot la cual es una herramienta establecida de aprendizaje para la evaluación lúdica de estudiantes. Las preguntas están principalmente relacionadas a bucles y condicionales de programación:

1. ¿Cuál de los siguientes se utiliza para comprobar una condición en un algoritmo?
    a. Mientras _ Hacer
    b. Variable
    c. **Si _ Entonces**
    d. Función

2. ¿Qué sucede cuando la condición en una declaración “Si _ Entonces” es falsa?

   a. El programa se bloquea

   b. **El bloque “Sino” se ejecuta**

   c. El bloque “Entonces” se ejecuta

   d. Se ejecutan todos los bloques

3. ¿Cuándo se debe utilizar un bucle “Mientras” en lugar de “Si _ Entonces”?

   a. Para comprobar una condición una sola vez

   b. **Para repetir un bloque de código si la condición es verdadera**

   c. Para ejecutar un bloque si la condición es falsa

   d. Para ejecutar código una sola vez

4. ¿Qué símbolo se utiliza para comparar si dos variables son iguales?

   a. =

   b. <=

   c. !=

   d. >

5. ¿Qué es una variable en programación?

   a. Un valor que NO puede cambiar

   b. Una función que realiza cálculos

   c. Un numero

   d. **Un valor en memoria que puede cambiar**

6. ¿Qué palabra se utiliza para evaluar una condición o la otra?

a. AND

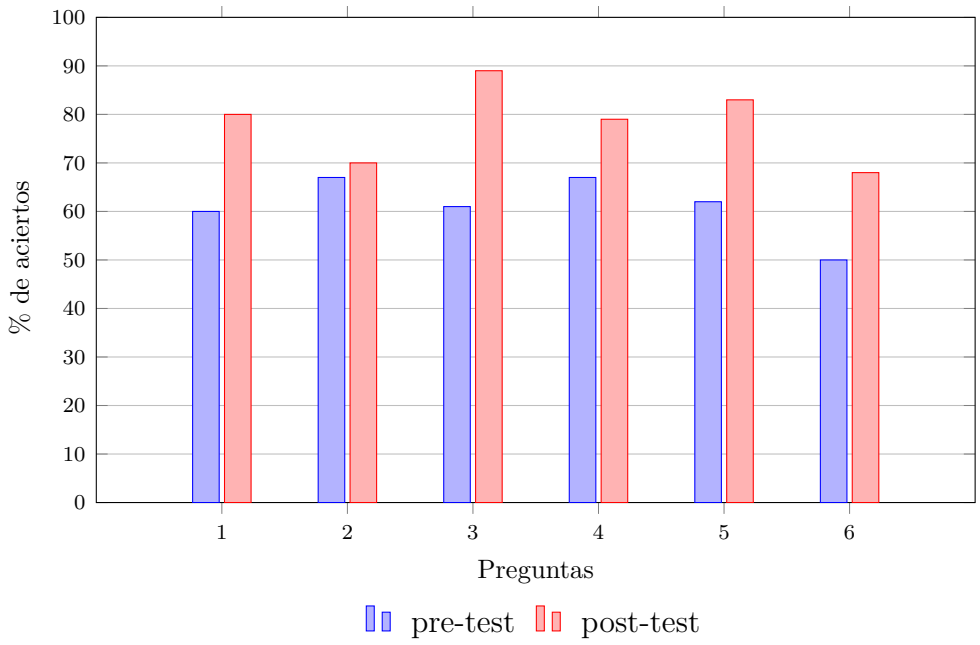
b. ELSE

c. **OR**

d. FOR

Figura 12

Grafica sobre distribución de aciertos



La Tabla 3 expone los datos del análisis cuantitativo mediante la prueba estadística *t Student* para determinar las diferencias entre los resultados antes y después de la capacitación en el producto del laboratorio virtual (Lemulogic). Los resultados de los grupos en los cuestionarios pre-test y post-test es promediado entre los dos grupos para obtener valores individuales en los aciertos de los cuestionarios.

El cuestionario pre-test fue administrado a los dos grupos al inicio de la sesión, con resultados promediando el 61 % de aciertos entre los dos grupos, Posterior a la instrucción y práctica del laboratorio virtual, se repitió el cuestionario bajo las mismas condiciones, el cual obtuvo 78 % de aciertos en promedio.

Cuadro 3

Resultados de cuestionarios

Pregunta	**Aciertos pre-test (X_1)**	**Aciertos post-test (X_2)**
1	60 %	80 %
2	67 %	70 %
3	61 %	89 %
4	67 %	79 %
5	62 %	83 %
6	50 %	68 %
$\overline{X}$	61.17 %	78.16 %
S_X	5.698	7.243
σ_X	2.548	3.239

Para obtener un contraste entre los puntajes de cuestionarios pre-test y post-test se hace uso de la prueba *t Student* mediante las fórmulas:

$$t = \frac{\overline{X}_1 - \overline{X}_2}{\sigma_{dif}} \qquad \sigma_{dif} = \sqrt{\sigma_{X1}^2 + \sigma_{X2}^2} \qquad gl = n_1 + n_2 - 2$$

$$t = \frac{61{,}17 - 78{,}16}{4{,}12} \qquad \sigma_{dif} = \sqrt{2,548 + 3{,}239} \qquad gl = 6 + 6 - 2$$

$$t = -4{,}124 \qquad \sigma_{dif} = 4{,}121 \qquad gl = 10$$

Los resultados de la prueba *t Student* muestan una diferencia significativa entre el cuestionario pre-test con respecto a los resultados post-test, $t(10) = 4{,}124, p = 0{,}001$, teniendo en cuenta que el valor crítico de tablas (con un nivel de significancia $\alpha = 0{,}95$) es

de 1,812, siendo el valor t calculado mayor al critico, lo que sugiere que el uso del laboratorio virtual tuvo un impacto positivo en la muestra de estudiantes, con el tema de bucles y condicionales.

Al final de las sesiones, se compartió a los estudiantes un formulario de satisfacción con respecto a la experiencia de usuario en la aplicación y sugerencias de posibles funcionalidades. La Tabla 4 condensa los resultados obtenidos en la recolección de datos por medio de un formulario de Google compartido.

Cuadro 4: Resultados obtenidos en encuesta de retroalimentación

Pregunta	**Respuestas**
¿Qué tan intuitivo considera el laboratorio con respecto a otras plataformas de desarrollo?	Mayoritariamente positivo, con un promedio de calificación de 4.4 sobre 5.
¿Qué tan efectiva considera la experiencia en su comprensión de bucles y condicionales?	Generalmente positivo, un 75 % calificado con 4 o más estrellas (promedio 4.2).
¿Qué tan probable es que utilices un laboratorio similar para otros conceptos de programación?	Altamente probable, indicando satisfacción con la aplicación (promedio 4.3).
¿Qué funcionalidades te gustaría agregar en el editor de nodos?	Función deshacer y tutoriales integrados que expliquen el uso del aplicativo principalmente.
¿Que otro tipo de experiencia te gustaría en la plataforma?	Entornos 3D, Rompecabezas, Música y Laberintos fueron sugerencias populares.

Continued on next page

Cuadro 4: Resultados obtenidos en encuesta de retroalimentación (Continued)

Pregunta	**Respuestas**
¿Qué otros conceptos de programación o lógica te gustarían ver integrados en futuras versiones?	Los estudiantes expresaron interés en estructuras de datos, funciones y más interfaces de control en el minijuego impartido.

El análisis cualitativo del formulario revela que los estudiantes encuentran el laboratorio virtual intuitivo y efectivo para el aprendizaje de bucles y condicionales. Sin embargo, se expresa un deseo por funcionalidades adicionales, tables como deshaces acciones, conceptos más avanzados y múltiples entornos lúdicos.

En posteriores versiones del laboratorio, un análisis más profundo podría implicar la categorización de estudiantes por edad y experiencia para obtener información más detallada; Especialmente la inclusión de tutoriales integrados y ayudas contextuales que faciliten el uso sin necesidad de intervención docente. Al abordar los comentarios proporcionados por los estudiantes, el laboratorio se puede optimizar para crear una experiencia más atractiva y eficaz.

Conclusiones

El presente estudio ha culminado exitosamente en el desarrollo e implementación de Lemulogic, un laboratorio virtual interactivo diseñado específicamente para la enseñanza de ciclos repetitivos en programación. Los resultados obtenidos demuestran el cumplimiento satisfactorio de los objetivos planteados:

En relación con el primer objetivo específico, se logró diseñar una aplicación web accesible mediante una interfaz basada en nodos, implementada con Godot Engine. La evaluación cualitativa reveló una alta satisfacción de los usuarios, con una calificación promedio de 4.4 sobre 5 en términos de intuitividad, demostrando que la plataforma cumple con los criterios de accesibilidad y usabilidad establecidos.

Respecto al segundo objetivo, se desarrollaron ejercicios prácticos integrados en un minijuego con temática de tanques, que permitió a los estudiantes aplicar conceptos de ciclos repetitivos en un entorno lúdico. El análisis mediante el marco Octalysis demostró fortalezas particulares en las dimensiones de Empoderamiento e Impredecibilidad, validando la efectividad del enfoque lúdico para el aprendizaje de programación.

En cuanto al tercer objetivo, la evaluación del impacto académico mostró una mejora aceptable en la comprensión de conceptos de programación, evidenciada por un incremento del 17 % en el rendimiento entre las pruebas pre y post (de 61.17 % a 78.16 %). La prueba t-Student ($t(10) = 4.124$, $p = 0.001$) confirma que esta mejora es estadísticamente significativa, validando la efectividad del laboratorio virtual como herramienta de aprendizaje.

Los resultados obtenidos sugieren que Lemulogic representa una solución viable para abordar los desafíos en la enseñanza de programación en la Universidad Santo Tomás, proporcionando una base sólida para futuras expansiones y mejoras del sistema. Las sugerencias de los estudiantes, como la inclusión de funcionalidades adicionales y nuevos entornos lúdicos, ofrecen direcciones claras para el desarrollo futuro del laboratorio virtual.

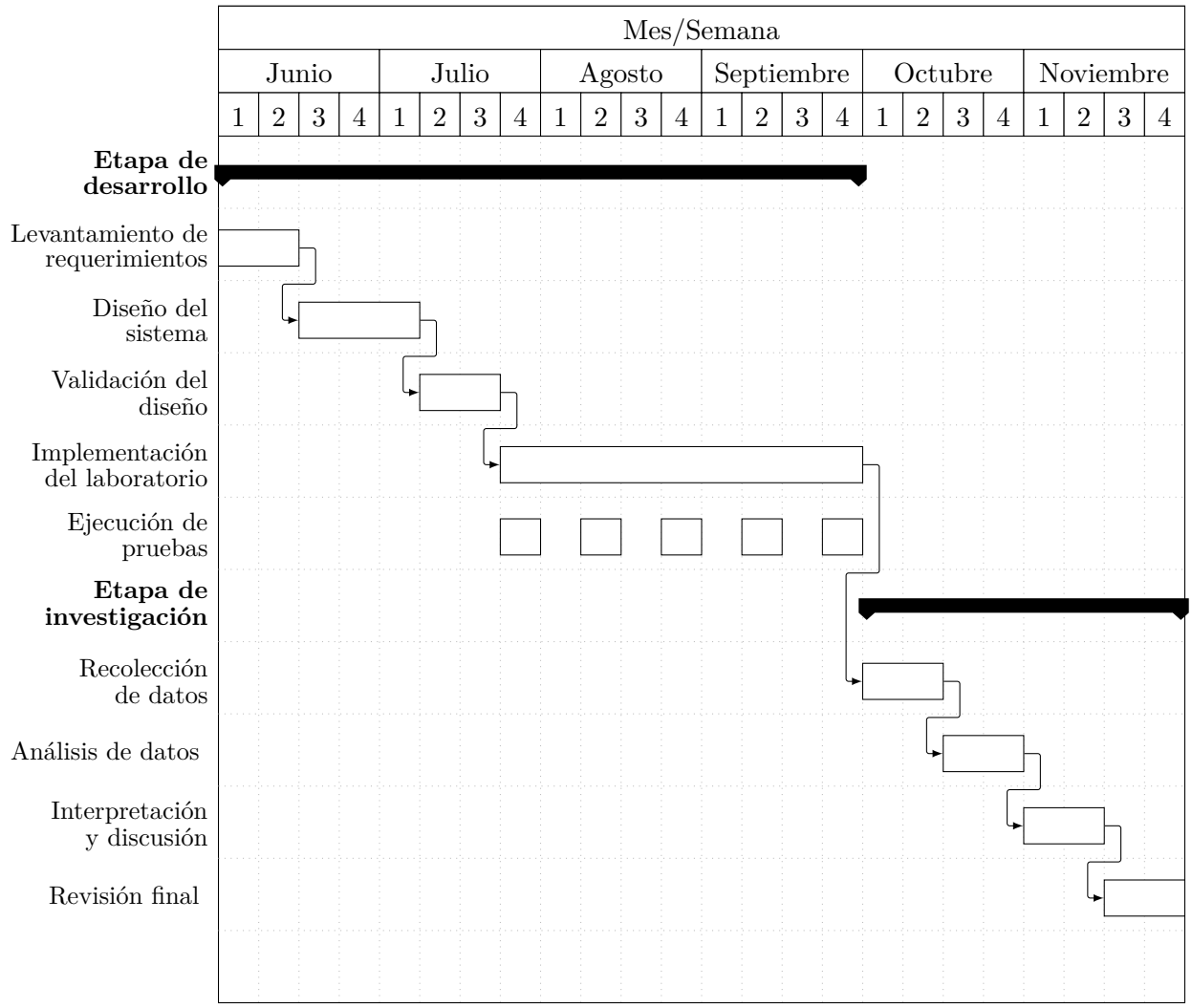
Esta implementación demuestra el potencial de la gamificación y los laboratorios

virtuales para mejorar la experiencia de aprendizaje en programación, especialmente en conceptos fundamentales como bucles y condicionales. El proyecto sienta las bases para futuras investigaciones y desarrollos en el campo de la educación en programación, contribuyendo así al objetivo más amplio de mejorar la calidad de la enseñanza en la ingeniería de sistemas.

Cronograma

Figura 13

Cronograma del proyecto



Referencias

Akpata, E., & Riha, K. (2004). Can Extreme Programming be used by a Lone Programmer. https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=4e97a3b6e4f64f7c125d7f789f8aa60d021ded2a

Chou, Y. (2015). *Actionable Gamification: Beyond Points, Badges, and Leaderboards*. Createspace Independent Publishing Platform. https://books.google.com.co/books?id=jFWQrgEACAAJ

Hurtado, G. P. G., Negrón, A. P. P., Álvarez, M. C. G., & Alvarez, M. C. G. (2021). Mobile application based on gamification to promote microlearning in Software Engineering. *Iberian Conference on Information Systems and Technologies*. https://doi.org/10.23919/cisti52073.2021.9476569

Maiga, J., Maiga, J., Emanuel, A. W. R., & Emanuel, A. W. R. (2019). Gamification for Teaching and Learning Java Programming for Beginner Students — A Review. *Journal of Computers*. https://doi.org/10.17706/jcp.14.9.590-595

Pinto, M., & Terroso, T. (2022). Learning Computer Programming: A Gamified Approach. *International Computer Programming Education Conference*. https://doi.org/10.4230/oasics.icpec.2022.11

Santamaría, M., & Valentina, J. (2023). Modelo predictivo de la mortalidad académica del programa de Ingeniería de Sistemas de la USTA Seccional Tunja basado en técnicas de Machine Learning. http://hdl.handle.net/11634/51470

Shahid, M., Wajid, A., Haq, K. U., Saleem, I., & Shujja, A. H. (2019). A Review of Gamification for Learning Programming Fundamental. *2019 International Conference on Innovative Computing (ICIC)*, *15*, 1-8. https://doi.org/10.1109/icic48496.2019.8966685

Zhan, Z., He, L., Tong, Y., Liang, X., Guo, S., & Lan, X. (2022). The effectiveness of gamification in programming education: Evidence from a meta-analysis. *Computers*

and Education: Artificial Intelligence, *3*, 100096.

https://doi.org/10.1016/j.caeai.2022.100096

ESPECIFICACIÓN DE REQUISITOS DE SOFTWARE

para

Lemulogic

Versión 1.0

Elaborado por

Harrizon Alexander Soler Galindo

Universidad Santo Tomás
Seccional Tunja

16 de diciembre de 2024

Ficha del documento

Nombre	Fecha	Motivo de cambios	Versión
Harrizon Soler	2024-10-21	Borrador inicial	1.0

Índice general

1 Introducción

El presente documento sigue el estándar IEEE 29148:2018 en su estructura y referencias normativas.

1.1. Propósito

Este documento especifica la funcionalidad inicial del laboratorio virtual Lemulogic, el cual proporciona un entorno gamificado de facil acceso para el aprendizaje de los fundamentos en programación, enfocado principalmente en bucles y condicionales. La aplicación tiene el objetivo de ser usado como herramienta pedagogica en la enseñanza de programación. Este documento es de carácter inicial y puede ser ajustado para alinearse mejor con las necesidades identificadas.

1.2. Alcance

El laboratorio virtual funcionara bajo el nombre de "Lemulogic" en un entorno web, pensado para complementar los procesos de enseñanza sobre fundamentos de programación en ambientes academicos, y asi facilitar la comprensión de dichos conceptos. La interacción del usuario gira entorno a un lenguaje de programación visual, diseñado e implementado especificamente para el aplicativo, el cual aprovecha el flujo de ejecucion a un proposito gamificado, que consiste en minijuegos intercambiables que ejecutan el codigo desarrollado por el usuario.

1.3. Referencias

Titulo	**Autor**	**Versión**	**Fecha**	**Fuente**
IEEE International Standard - Requirements engineering	IEEE	29148-2018	Nov 30, 2018	ieeexplore.ieee.org
Godot - Requisitos de sistema	Godot Community	4.3	2024	docs.godotengine.org
GitHub Docs	GitHub	Free	2024	docs.github.com

1.4. Definiciones, acrónimos y abreviaturas

Termino	Definición
Godot	Motor de videojuegos 2D y 3D multiplataforma, libre y de código abierto.
HTML	*HyperText Markup Language*. Lenguaje de marcado utilizado en la creación de páginas web.
GB	*Gigabyte*. Unidad de almacenamiento de información estandarizada.
RAM	*Random Access Memory*. Memoria de almacenaje a corto plazo para guardar información de forma temporal.

2 Descripción general

2.1. Perspectiva del producto

Lemulogic ofrece un entorno interactivo, el cual funciona en navegadores web para escritorio y proporciona minijuegos de codificación con retroalimentación inmediata sin necesidad de software adicional.

2.1.1. Carácteristicas de usuario

Se contempla un tipo unico de usuario (estudiante) que interactuará con el sistema, sin ningun tipo de registro o persistencia en el mismo. El aplicativo ofrece dos interfaces principales para el usuario:

- Interfaz de desarrollo visual
- Entorno de ejecución de minijuegos

2.1.2. Interfaces de hardware

En su estado actual, el laboratorio virtual funcionara exclusivamente en sistemas operativos de escritorio (Microsoft Windows, MacOS, Linux). No se ha considerado soporte para plataformas moviles en el momento. La documentación de Godot recomienda un minimo de 4 GB de memoria RAM entre otras restricciones para proyectos exportados a plataforma web (Vease Referencias).

2.1.3. Interfaces de software

El sistema trabajará en entorno web de escritorio, a traves de navegadores compatibles (Chrome 57+, Firefox 52+, Safari 15+, Edge 79+) los cuales tengan soporte para las tecnologias WebGL 2.0 y WebAssembly.

El laboratorio virtual será desarrollado con el motor de videojuegos Godot en el modo de exportación "Compatibilidad" para plataforma HTML.

2.1.4. Operaciones

Las operaciones de despliegue se llevaran a cabo en la plataforma Github Pages, la cual tiene un límite permisivo en ancho de banda de 100 GB por mes. Asi mismo cada despliegue tiene un periodo de inactividad de 1 minuto maximo (Vease Referencias).

2.2. Funcionalidad del producto

Cuadro 2.1: Historia de usuario *HU1*

Nombre	HU1 - Entorno de desarrollo
Actor	Estudiante
Yo (rol)	Yo como usuario de la aplicación
Deseo (funcionalidad)	Deseo tener control de un entorno interactivo de programación
Para (objetivo)	Para desarrollar mis algoritmos de manera visual
Prioridad	**Alta**
Criterios de aceptación: ▪ Como minimo las herramientas disponibles para el desarrollo del algoritmo debe cumplir el flujo de bucles y condicionales. ▪ El entorno de desarrollo debe aprovechar estrategias graficas de interaccion, ▪ El entorno debe guiar al usuario para evitar errores comunes.	

Cuadro 2.2: Historia de usuario *HU2*

Nombre	HU2 - Minijuego ilustrativo
Actor	Estudiante
Yo (rol)	Yo como usuario de la aplicación
Deseo (funcionalidad)	Deseo ejecutar mi algoritmo en un minijuego que muestre el flujo del mismo
Para (objetivo)	Para entender de manera visual y ludica el comportamiento de las expresiones en programación
Prioridad	**Alta**
Criterios de aceptación: ▪ El minijuego debe ser atractivo visualmente y "enganchar" al usuario a mejorar y/o crear los algoritmos que ejecute el mismo.	

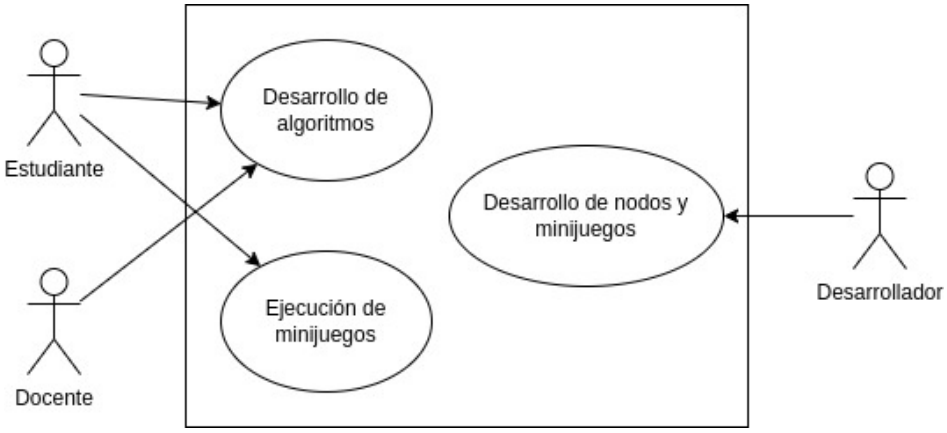
Cuadro 2.3: Caso de uso *CDU1*

Nombre	CDU1 - Desarrollo de algoritmo
Actor	Estudiante
Meta	Ejecutar un algoritmo desarrollado por actor
Precondiciones	Abrir la pagina web de la aplicación
Escenarios: ▪ **Flujo básico:** Usuario abre la aplicación web y desarrolla un algoritmo creando y conectando nodos, para posteriormente ejecutarlo exitosamente en el minijuego. ▪ **Flujo alternativo 1:** El usuario desea ejecutar el algoritmo pero el sistema detecta un error sintactico por lo que abandona la aplicación. ▪ **Flujo alternativo 2:** El sistema detecta un error sintactico, el cual es corregido por el usuario y ejecuta la experiencia exitosamente. ▪ **Flujo alternativo 3:** El usuario no queda conforme con el comportamiento de su algoritmo, asi que retorna al editor y lo modifica. Ejecuta el minijuego nuevamente con el nuevo algoritmo.	

Cuadro 2.4: Caso de uso *CDU2*

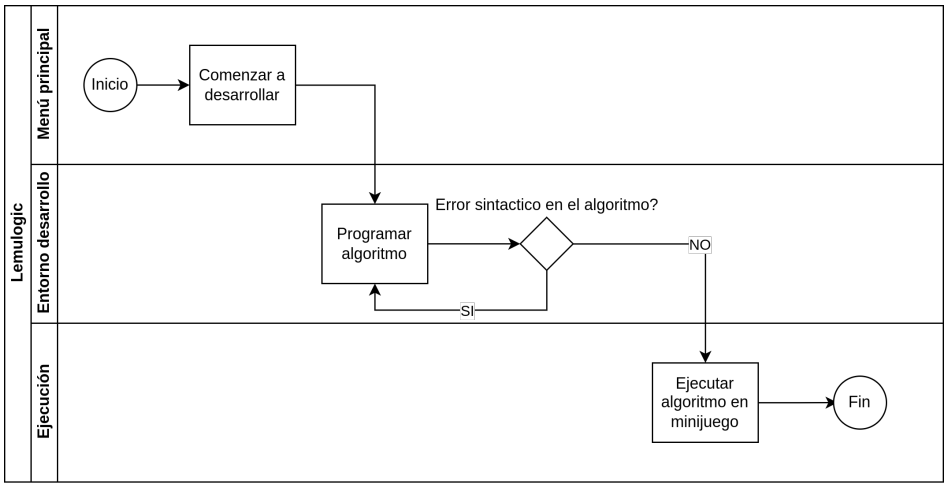
Nombre	CDU2 - Detección de errores
Actor	Estudiante
Meta	Detectar errores sintacticos previos a ejecución
Precondiciones	Abrir la pagina web de la aplicación
Escenarios:	

- **Flujo básico:** Usuario abre la aplicación web y desarrolla un algoritmo incompleto. El sistema no encuentra errores.
- **Flujo alternativo 1:** El sistema detecta un error de un nodo con conexiones incompletas.
- **Flujo alternativo 2:** El sistema detecta un error de una variable creada pero sin un valor asignado.



2.3. Carácteristicas del usuario

El publico objetivo para el aplicativo consta de estudiantes en primeros semestres de Ingenieria de Sistemas, del cual se espera un dominio basico en el manejo de un sistema operativo y navegación web.



2.4. Limitaciones

El aplicativo debe cumplir con la tarea de gamificar la enseñanza de bucles y condicionales bajo interacciones lúdicas.

2.5. Suposiciones y dependencias

El laboratorio debe contar con un desarrollador administrador que verifique y asegure (esporádicamente) el correcto funcionamiento de la pagina web y las funcionalidades del aplicativo.

3 Requisitos específicos

Cuadro 3.1: Requisito funcional *RF1*

Número	RF1
Nombre	Accesibilidad basada en navegador
Tipo	Requisito
Descripción	El laboratorio debe ser accesible desde navegadores web modernos sin la necesidad de software adicional.
Prioridad	**Alta**

Cuadro 3.2: Requisito funcional *RF2*

Número	RF2
Nombre	Editor interactivo de programación
Tipo	Requisito
Descripción	La plataforma debe incluir un editor con una interfaz basada en nodos para el diseño de algoritmos.
Prioridad	**Alta**

Cuadro 3.3: Requisito funcional *RF3*

Número	RF3
Nombre	Detección de errores y retroalimentación
Tipo	Requisito
Descripción	El editor debe resaltar errores sintácticos y lógicos.
Prioridad	**Alta**

Cuadro 3.4: Requisito funcional *RF4*

Número	RF4
Nombre	Integración con minijuegos
Tipo	Requisito
Descripción	Los algoritmos desarrollados por estudiantes deben ser ejecutables en minijuegos interactivos.
Prioridad	**Alta**

Cuadro 3.5: Requisito funcional *RF5*

Número	RF5
Nombre	Conexión de nodos
Tipo	Requisito
Descripción	El sistema debe permitir la conexión de nodos por interacción de arrastre con el ratón, con ayuda de la diferenciación de colores por tipo de conexión requerida
Prioridad	**Alta**

Cuadro 3.6: Requisito funcional *RF6*

Número	RF6
Nombre	Creación de variables
Tipo	Requisito
Descripción	El sistema debe permitir la creación de variables con nombre personalizado
Prioridad	**Alta**

Cuadro 3.7: Requisito funcional *RF7*

Número	RF7
Nombre	Modularidad
Tipo	Requisito
Descripción	Incluya un conjunto predefinido de nodos para lógica, variables, bucles e interacciones del entorno.
Prioridad	**Alta**

3.1. Requisitos de rendimiento

- El sistema debe inicializarse (pasar de la pantalla de carga inicial) en menos de 10 segundos para garantizar el uso del laboratorio por el usuario.
- El laboratorio debe minimizar el uso de recursos del sistema (CPU y memoria) para ser funcional en equipos con recursos limitados.
- El motor de ejecución de algoritmos y los minijuegos deben operar de manera independiente sin interferencias que provoquen bloqueos o retrasos. Así mismo las dos ejecuciones deben alcanzar o superar los 60 fps o 16 ms de latencia por fotograma.

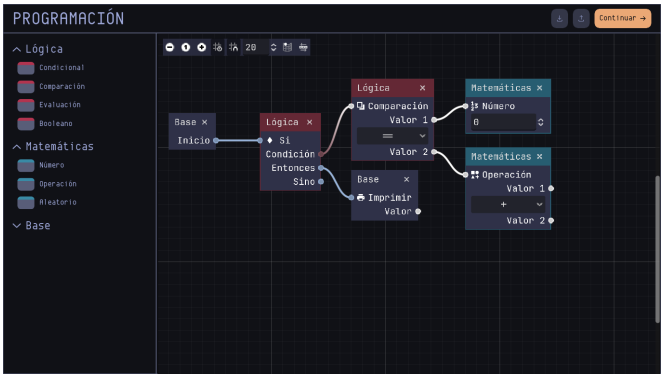
3.2. Requisitos de seguridad

Los algoritmos desarrollados por el usuario en el sistema deben operar de manera aislada de la ejecución del navegador y sistema operativo.

3.3. Restricciones de diseño

- La aplicación debe utilizar un esquema visual coherente basado en colores y tipografías que reflejen las categorías funcionales y den consistencia al diseño.
- El diseño debe adaptarse a las capacidades de interfaz grafica del motor Godot Engine.

3.4. Interfaces de usuario


PROGRAMACIÓN
Continuar
Lógica
Condicional
Comparación
Evaluación
Booleano
Matemáticas
Número
Operación
Aleatorio
Base
Base
Inicio
Lógica
Si
Condición
Entonces
Sino
Lógica
Comparación
Valor 1
Valor 2
Base
Imprimir
Valor
Matemáticas
Número
0
Matemáticas
Operación
Valor 1
+
Valor 2