FARAUTE, Vol. 17, No. 2: 40-46; 2022

Online ISSN 2542-3061
Print ISSN 1698-7418

Implementación de una extensión para el depurador del entorno de enseñanza-aprendizaje del lenguaje de programación Hedy

Herrera, M.[1] y Pelay, J. [1]

[1] *Universidad de Carabobo. Facultad Experimental de Ciencias y Tecnología. Dpto. de Computación. Bárbula, Edo. Carabobo, Venezuela.*

*__Autor de correspondencia__: mherrera@uc.edu.ve, pelayjesus@pm.me

Recibido: 23/09/2025, Revisado: 15/10/2025, Aceptado: 15/11/2025

Resumen

El lenguaje de programación Hedy es una herramienta educativa diseñada para enseñar programación textual a niños de 10 a 14 años. Con traducciones a más de 60 idiomas, es utilizada globalmente en entornos escolares y cuenta con una comunidad activa de educadores y traductores. Sin embargo, a pesar de su adopción, su depurador original tenía una limitación significativa, restringiendo su funcionalidad a tan solo una parte de la trayectoria de aprendizaje de los estudiantes. Este trabajo trazó como objetivo, extender el depurador del lenguaje a la totalidad del trayecto de aprendizaje-enseñanza de Hedy, el cual está conformado por 18 niveles. La propuesta consistió en superar esta limitación, mediante el uso de *source mapping* para vincular el código fuente de Hedy con el de *Python* generado por el transpilador, así como de la librería *Skulpt*, encargada a su vez de ejecutar el código de *Python* en el navegador. Los resultados demuestran la implementación exitosa del depurador en todos los niveles. Además, se logró una mejora en el rendimiento, reduciendo a una sola la llamada *HTTP* por cada programa ejecutado. La manipulación detallada de la librería *Skulpt* no solo optimizó el proceso, sino que también estableció un precedente para su uso en futuras investigaciones y desarrollos. Este avance no solo mejora la experiencia de aprendizaje en Hedy, sino que también abre nuevas oportunidades para la investigación en herramientas de depuración para el aprendizaje de la programación

Palabras Claves: Lenguaje de Programación; Depurador; Programación Gradual; Educación en Programación; Source Mapping

Implementation of an extension for the debugger of the teaching-learning environment of the Hedy programming language

Abstract

The Hedy programming language is an educational tool designed to teach textual programming to children aged 10 to 14. With translations into more than 60 languages, it is used globally in school environments and has an active community of educators and translators. However, despite its adoption, its original debugger had a significant limitation, restricting its functionality to only a part of the students' learning journey. This work aimed to extend the debugger of the language to the entirety of Hedy's teaching-learning path, which consists of 18 levels. The proposal consisted of overcoming this limitation by using source mapping to link Hedy's source code with the Python code generated by the transpiler, as well as the Skulpt library, which is responsible for executing the Python code in the browser. The results demonstrate the successful implementation of the debugger at all levels. Additionally, an improvement in performance was achieved by reducing the HTTP call to just one for each executed program. The detailed manipulation of the Skulpt library not only optimized the process but also set a precedent for its use in future research and developments. This advance not only enhances the learning experience in Hedy but also opens new opportunities for research in debugging tools for programming education.

Keywords: Programming Languages; Debugger, Gradual Programming, Programming Education; Source Mapping

1. Introducción

En la era de la información, la enseñanza de la programación se ha consolidado como una competencia fundamental en la educación básica y media a nivel mundial. El desarrollo del pensamiento algorítmico, la capacidad para la resolución de problemas y el diseño de sistemas son habilidades transferibles que benefician a los estudiantes en múltiples disciplinas. Sin embargo, el primer contacto con la programación textual a menudo representa una barrera significativa para los principiantes, especialmente niños y adolescentes. La complejidad de la sintaxis, con sus reglas estrictas sobre paréntesis, comas y palabras clave, puede generar una alta carga cognitiva y frustración en los estudiantes, llevando a elevadas tasas de abandono en los cursos introductorios [1].

Para mitigar estas dificultades, [3] creó la idea de un lenguaje de programación gradual, donde los conceptos son introducidos de manera progresiva de forma tal que, a medida que se enseñan, la sintaxis del lenguaje cambia y se vuelve más compleja. Esta autora fue quien construyó Hedy, un lenguaje compuesto de 18 niveles, cuyo último nivel es un subconjunto sintáctico de *Python*.

Según [4] Hedy es más fácil de aprender que *Python*, "bajando la carga cognitiva asociada con la sintaxis". De acuerdo con las respuestas de los participantes del estudio, "lo que más les gustó de Hedy [...] fue su facilidad de uso", por lo que Hedy se constituye como una alternativa real, para la enseñanza de programación textual.

Aunado a lo anterior, Hedy además de ser un lenguaje de programación, está compuesto por un entorno de enseñanza-aprendizaje, en el cual los profesores pueden crear clases, asignaciones y evaluar a sus estudiantes. Más específicamente, entre los diversos componentes del entorno se encuentra un depurador, el cual permite: conocer el valor de las variables del programa, saltar líneas y ejecutar el programa paso a paso [5].

Gracias a su facilidad de uso, y estar traducido a 68 idiomas, Hedy ha tenido una acogida favorable a nivel mundial, con 45 mil usuarios registrados y más de 700 mil programas guardados en su base de datos. Hedy también es utilizado en más de 1000 escuelas alrededor del mundo, de acuerdo con datos provistos por su comité organizativo.

En este contexto, un depurador es de especial importancia en un entorno pedagógico, pues ofrece la posibilidad de que en tiempo de ejecución, el usuario pueda encontrar posibles problemas en el código o incluso entender el flujo del programa de una mejor manera. Es así como, aprender a depurar el código es una tarea esencial para un programador, sin embargo, pudiera calificarse como una tarea compleja, cuya experticia es difícil de alcanzar e incluso desarrollar. Para acompañar este hecho, algunas opiniones expresadas por estudiantes, hacen referencia a que ."encontrar *bugs* es más difícil que arreglarlos"[2].

Es importante señalar que al momento de desarrollarse la investigación,el depurador presentaba algunas limitaciones y, por ende, oportunidades de mejora. Entre las de mayor importancia, se pudieran mencionar: solamente funcionaba en los primeros siete niveles de los dieciocho que conformaban a Hedy. Adicionalmente, mientras el depurador emulaba la ejecución paso a paso, es decir, al ejecutar toda la línea durante la ejecución, no se lograba apreciar el flujo correcto del programa, especialmente cuando había presencia de varios comandos en una sola línea.

2. Antecedentes

Depurador de Hedy

El depurador utilizado hasta antes de la extensión, fue desarrollado por Redeelar en su trabajo de fin de carrera: "Designing and implementing a debugger for the Hedy programming language" (2022) para la Universidad de Leiden (ubicada en los Países Bajos). En dicho trabajo, Redeelar desarrolló los aspectos necesarios para el funcionamiento básico de un depurador, a saber: mostrar las variables que se encuentran actualmente almacenadas además de su valor y manipular el flujo de control del programa. Adicionalmente, el estudio, detallaba un análisis del impacto tanto cuantitativo como cualitativo del depurador, mediante entrevistas realizadas a seis participantes y analizando los tiempos de carga del sitio *web*, luego de implementar el depurador.

Acorde con Radeelar, en las figuras 1 y 2 se aprecia el funcionamiento del depurador, el cual sigue el proceso que se describe a continuación:

Figura 1. Captura de pantalla mostrando la ejecución de la primera línea de un programa. Tomado de: [5], *Designing and implementing a debugger for the Hedy programming language*

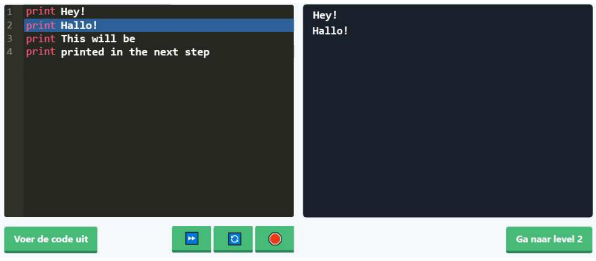

Figura 2. Captura de pantalla mostrando la ejecución de la segunda línea de un programa. Tomado de: [5], *Designing and implementing a debugger for the Hedy programming language*

> La función es llamada cada vez que `debugRun()` es invocada. La función [...] lee el programa de Hedy y la línea de depuración actual. Luego divide el programa por líneas. Finalmente, define el código a ser ejecutado según las líneas de código, desde la primera hasta la actual.

Por consiguiente, el proceso no hace uso de símbolos del programa ni de interrupciones, simplemente toma ventaja de que los primeros niveles de Hedy no contienen indentación para poder ejecutarlos (y compilarlos) línea a línea. Por ello, Radeelar concluía que una de las mejoras posibles para el depurador sería extender su uso para trabajar con bucles y bloques anidados.

Arquitectura del sistema de enseñanza-aprendizaje de Hedy

La arquitectura del sistema, tal como señala la Figura 3, sigue una estructura cliente-servidor, en la cual los programas son escritos en el navegador y luego enviados al servidor, en donde son transpilados de

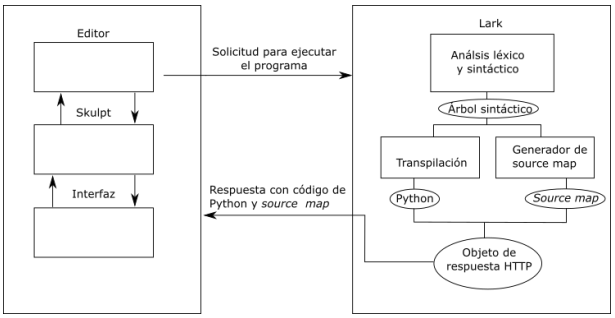

Figura 3. Diagrama de arquitectura del entorno de programación de Hedy.

Hedy a *Python*, para luego ser ejecutados localmente en el navegador.

Asimismo, muestra un diagrama con la descripción general de los aspectos de la arquitectura que están involucrados en el proceso de escribir y ejecutar programas en Hedy, siguiendo el modelo cliente-servidor.

Más específicamente, en el sistema de programación de Hedy, el servidor cumple la función de traducir los programas de Hedy a *Python*, haciendo uso de las siguientes librerías:

- Flask: un *microframework* que maneja los aspectos web del sistema, contándose entre ellos el manejo de solicitudes *HTTP*, renderizado de plantillas *HTML* y manejo de rutas.
- Lark: esta librería toma gramáticas escritas con una sintaxis propia adaptada de la sintaxis de EBNF (*Extended Backus-Naur Form*), y genera analizadores sintácticos, uno para cada nivel en cada idioma.

Por su parte, el cliente se encarga de ofrecer un entorno de programación a los usuarios, de forma tal, que sean capaces de escribir los programas y ejecutarlos. Para ayudar con ésto, se usaron las siguientes librerías:

- *CodeMirror*: es un editor de código diseñado para funcionar en navegadores, ofreciendo posibilidades de personalización para distintos aspectos de la interfaz, a fin de adaptarse a las necesidades de los usuarios. Una de sus características más importantes es la posibilidad de agregar coloreado de sintaxis, haciendo uso de gramáticas libre de contexto y un generador de *parsers* incluidos en su sistema.

- *Skulpt*: es una librería que compila *Python* en *JavaScript*. El código generado por *Skulpt* deberá ser introducido en su propio entorno de ejecución.

3. Implementación

Cómo se describió en la sección anterior, Hedy hace uso de la librería *Skulpt* para poder ejecutar los programas transpilados en *Python*, en el navegador. Para ser capaces de identificar cuáles líneas de *Python* corresponden a cuáles líneas de código Hedy, el compilador genera un *source map*. Posteriormente, el *source map* será usado para colorear la línea apropiada, durante la ejecución de Hedy. Finalmente, la estructura de datos no es más que la correspondencia entre las posiciones del programa de Hedy y las posiciones del programa de *Python*. En el listado 1 se puede apreciar un ejemplo de un *source map* para un programa de nivel 1.

```
1/1-1/16 --- imprimir Hola!
1/1-1/16 --- print("Hola!")

2/1-2/30 --- preguntar Cual es tu nombre?:
2/1-2/38 --- respuesta = input("Cual es tu
    nombre?")

3/1-3/9  --- eco hola
3/1-3/22 --- print("hola " + respuesta)
```

Listado 1. *Source map* ejemplo. Fuente: elaboración propia.

En este escenario la correspondencia es trivial, es decir, toda la primera línea del programa de Hedy se corresponde con cada línea del programa de *Python*. En otros casos, como los referidos a comandos adelante o girar una línea de Hedy, pueden generarse varias líneas de código *Python*.

El hecho de que una línea de Hedy genere varias líneas de *Python*, trae como consecuencia que se deba marcar en cuál de las líneas de *Python* pausar el programa. Para ello, se agrega un comentario al lado de la línea relevante para marcarla, y allí entonces debe ser colocado el punto de pausa. Los criterios usados para decidir en qué líneas deben ser ubicados los puntos de pausa, fueron los siguientes:

- En los comandos que representan el llamado a alguna función de *Python* y que ocupan una línea entera. Algunos de los comandos son: `imprimir`, `preguntar`, `eco`, `dormir`, entre otros.
- En las instrucciones de asignación también se coloca un punto de parada. Para los comandos de control de flujo, se coloca el punto de parada en la primera línea del comando.
- En los comandos de tortuga, se coloca el punto de parada en la línea en donde se ejecuta el llamado a la función correspondiente al módulo tortuga. Es decir, para el comando adelante, el punto de parada es la línea con el llamado a la función `t.forward`.

Una vez ha finalizado el paso de transpilación, el cliente debe procesar el programa recibido, configurando el entorno de ejecución y estableciendo los puntos de parada del programa. Cabe señalar, que la librería usada para ejecutar los programas de *Python* en el navegador, es la llamada *Skulpt*. Esta librería, para poder ejecutar el programa en modo depuración, provee una abstracción llamada *Suspensions*, las cuales son "continuaciones, generadas bajo demanda [...] y permiten la suspensión y subsecuente reanudación de ejecución de código de *Python*"[7].

De esta manera, las suspensiones pueden ser utilizadas para simular operaciones de entrada y salida, o para suspender la ejecución del programa en alguna sentencia. Para ello, *Skulpt* define dos tipos de suspensión:

- `Sk.promise`: se encarga de reiniciar la ejecución cuando la promesa de *JavaScript* se resuelve o retorna un error.
- `Sk.debug`: la ejecución debe ser suspendida manualmente y resumida.

Aunque *Skulpt* incorpora en su código las suspensiones, no son suficientes para proveer un entorno de depuración sin errores; debido a esto se usaron las modificaciones propuestas por [6]. En el Listado 2 se muestra la estructura de las suspensiones modificadas:

```
{
    asyncContext: <numero o undefined>,
    child: <Suspension, o null si estamos
        en el nivel mas interno>,
    $blk: <numero del bloque del cual fue
        creado la suspension>,
```

```
    $loc: <variables globales>,
    $gbl: <variables globales>,
    $tmps: <variables locales>
}
```

Listado 2. *Source map* ejemplo. Fuente: elaboración propia.

Durante la depuración del programa se debió distinguir entre código síncrono y asíncrono, ya que los mecanismos para manejarlos eran ligeramente distintos. Para las suspensiones provenientes de instrucciones síncronas, se debió mantener una pila con las suspensiones y así disponer de una historia de programa válida. La función handleSuspension fue la encargada de esta tarea, manejando las suspensiones y manipulando la pila. Para ello, la función manejó varios casos distintos:

- El dato recibido no contiene todos los campos de una suspensión: esto significa que, o terminó la ejecución del programa, o se está esperando por la resolución del llamado a una función definida por el usuario. Una vez se completa el llamado a esta función, se cambia el puntero a la suspensión actual para apuntar al padre.
- La suspensión es síncrona: en este caso se busca el sitio apropiado dentro de la pila de suspensiones, comparando la posición de la suspensión actual con la que se recibe por parámetro.
- La suspensión es de tipo promesa: en este caso se desenrolla la promesa, ejecutando todas las funciones que se puedan requerir hasta que se termine.

Una vez el programa se detiene en una línea de *Python*, también debe ser capaz de mostrarle al usuario las variables del programa y su valor actual, así como la línea de Hedy a la cual corresponde esa línea de *Python*. Para ello se utilizó la función `incrementDebugLine`, la cual recibe la suspensión activa del depurador y extrae de ésta el valor de sus variables globales (Hedy no tiene noción de ámbito de variables, así que todas sus variables son globales).

En este contexto, para colorear la línea apropiada se itera sobre el *source map*, y se obtienen la línea de inicio y final de cada sentencia. Si la sentencia dada por el *source map* se encuentra dentro del rango de la suspensión actual, se decide cuál línea colorear en base a los siguientes criterios:

- Si es un comando que no es de control de flujo, o es de control de flujo simple, se colorea la primera línea del código de Hedy.
- Si el comando es de control de flujo complejo, se espera hasta llegar al nivel más alto y también se colorea la primera línea del código de Hedy.

4. Resultados

Gracias al cambio de enfoque en la implementación, descrito en la sección anterior, la nueva versión logró el funcionamiento correcto en todos los niveles del depurador del lenguaje. Otro resultado de importancia, es la mejora de rendimiento obtenida gracias a esta nueva implementación.

En este sentido, para poder comparar ambas implementaciones, se creó un *script* que simuló la depuración de un programa de 18 líneas. Este *script* fue escrito haciendo uso del marco de pruebas de integración de *Cypress* que utiliza el sistema de aprendizaje-enseñanza de Hedy para ejecutar pruebas unitarias de su sitio *web*.

El *script* simuló las acciones de un usuario depurando un programa, tales como, escribir el programa en el editor y hacer clic en los botones de iniciar la depuración y ejecutar el siguiente paso. Para los propósitos de esta prueba, solamente se tomó el tiempo de la sección que ejecuta el programa paso a paso.

Este *script* fue ejecutado 20 veces, utilizando el entorno *Electron* del cual dispone *Cypress*, en una computadora *MacBook Air M3*. Para simular un entorno de ejecución real se aplicó *throttling* a las solicitudes *HTTP* al servidor. Este *benchmark* arrojó como resultado, que la antigua implementación del depurador empleó un tiempo promedio de 12.26 segundos mientras que la nueva tardó 3.03 segundos, representando una mejora del 24.7 %.

5. Conclusiones

Hedy es un sistema que ha alcanzado un interés creciente dentro de las comunidades de programadores,

investigadores y profesores de programación, alcanzando cientos de miles de visitas al mes, por usuarios del todo el planeta. En vista de esto, se hizo evidente la necesidad de implementar un depurador que estuviera a la altura de los requerimientos de esta creciente comunidad de usuarios.

Desde el punto de vista del proceso de desarrollo en la ingeniería del software, las herramientas para depurar un programa, son consideradas como indispensables, para brindar apoyo durante su construcción. Más específicamente, son utilizadas por los profesionales de la programación para encontrar errores; y por estudiantes en formación para comprender, de una manera más clara, el flujo de ejecución de los programas que desarrollan. En este sentido, el depurador desarrollado para el sistema de enseñanza-aprendizaje de Hedy, tomó como punto de partida estos principios del uso de los depuradores y los trasladó a su entorno educativo.

De esta manera, los objetivos de la presente investigación, propusieron la implementación de un depurador que se integró con la librería *Skulpt*, de forma tal que permitió pausar la ejecución de los programas de *Python* generados por el transpilador de Hedy. Trabajar con esta librería planteó un desafío, cuya resolución estuvo apoyada en el marco de desarrollo SCRUM, permitiendo integrar un flujo de trabajo a partir de un conjunto de *sprints*, cuyos resultados conformaron los entregables de cada una de las fases del desarrollo.

Otro hallazgo importante es que al ser un depurador simbólico, debía mostrar que el paso de ejecución de Hedy se correspondiera con la línea de *Python* que estuviera suspendida en ejecución. ParaCon este fin se utilizó un source map, con el fin de asignar las líneas de Hedy a las líneas de *Python*. Así, cuando el programa se pausara en alguna línea de *Python*, esta estructura de datos serviría como un puente para colorear la línea apropiada de Hedy como activa. Finalmente, los usuarios verían la ejecución de su programa en Hedy línea por línea, y no un programa en *Python* generado por el transpilador.

6. Recomendaciones

En aras de continuar con la mejora continua del depurador, se propone:

- Debido a que la librería *Skulpt* ha probado ser un desafío para razonar y trabajar sobre ella, se recomienda al equipo de Hedy que tome bajo su cargo un *fork* propio de dicha librería con el fin de mantenerlo, para así no depender del proceso de desarrollo del equipo de *Skulpt*.
- El código del depurador fue desarrollado siguiendo los estándares de desarrollo de la librería *Skulpt*, la cual usa *JavaScript* para desarrollar sus módulos. Por otro lado, el equipo de Hedy utiliza *TypeScript*, un lenguaje de programación que agrega tipado estático a *JavaScript*, haciendo el código más robusto. Una evaluación de la factibilidad de convertir este código de *JavaScript* a *TypeScript*, agregando soporte para tipos, pudiera ser de gran valor.

7. Trabajos Futuros

En este aspecto, sería relevante estudiar a profundidad el impacto del depurador en el proceso de enseñanza-aprendizaje de la programación y su utilidad en el aula de clase.

Por otro lado, el depurador desarrollado durante el proyecto pudiera ser extendido añadiendo otras funcionalidades, a saber: puntos de parada en el programa de Hedy escogidos por el usuario, habilidad para manipular el valor de las variables durante la ejecución así como la visualización de la ejecución del programa y del paso anterior, entre otros.

8. Agradecimientos

Se extienden agradecimientos a la Organización Hedy por abrir sus puertas y ofrecer apoyo durante la realización de esta investigación, y en particular a la profesora Felienne Hermans.

A la Universidad de Carabobo, especialmente al Departamento de Computación y su cuerpo docente.

Referencias

[1] Blackwell, C., Lauricella, A., and Wartella, E. (2014). Factors influencing digital technology use in early childhood education. *Computers & Education*, 77:82–90.

[2] Fitzgerald, S., McCauley, R., Hanks, B., Murphy, L., Simon, B., and Zander, C. (2009). Debugging from the student perspective. *IEEE Transactions on Education*, 53(3):390–396.

[3] Hermans, F. (2020). Hedy: a gradual language for programming education. In *Proceedings of the 2020 ACM Conference on International Computing Education Research, ICER '20*, pages 259–270. Association for Computing Machinery.

[4] Marleen Gilsing, J. P. y. F. H. (2022). Design, implementation and evaluation of the hedy programming language. *Journal of Computer Languages*, 73:1–17.

[5] Redelaar, F. (2022). Designing and implementing a debugger for the hedy programming language.

[6] Scheinder, J. (2020). Design and implementation of a graphics window and debugger for webtigerjython. Master's thesis, ETH Zurich.

[7] Skulpt (n.d.). Skulpt's docs. Recuperado el 01 de Agosto del 2025, de `https://skulpt.org/docs/index.html`.