

Universidad de La Habana
Facultad de Matemática y Computación



CLIF: Un DSL en Common Lisp para generar ficción interactiva

Autor:
Marian Susana Álvarez Suri

Tutor:
McS. Fernando Raúl Rodríguez Flores

Trabajo de Diploma
presentado en opción al título de
Licenciado en Ciencia de la Computación

15 de junio de 2025

github.com/marians002/CLIF

A Mima.

Agradecimientos

A mi familia por su dedicación y entrega de decenas de años.

A mi tutor por permitirme crecer con él. Por ser tan único. Por tanta experiencia y conocimientos brindados. Por la paciencia.

A mi novio por su cariño infinito, su compañía, su comprensión. A su familia, por brindarme un segundo hogar.

A Oda, por estar siempre presente, aún desde la distancia.

A todos los suspensos. En especial a Anita, por sostenerme, y a Dennis, por todas las risas.

Muchas gracias a todas las personas que me han ayudado a llegar hasta aquí.

Opinión del tutor

La ficción interactiva es un tipo de narrativa en la que el lector o usuario puede tomar decisiones que afectan el desarrollo y desenlace de la historia. Combina elementos de lectura y juego, haciendo que quien la experimenta se convierta en protagonista activo de la obra.

La ficción interactiva permite integrar diversas perspectivas culturales, lo que fomenta la sensibilidad cultural y la inclusión entre los participantes. Al permitir que los usuarios experimenten situaciones desde distintos puntos de vista, promueve la empatía y un mayor entendimiento intercultural. También tiene muchas aplicaciones en la docencia. Por otro lado en nuestro entorno y clases, en ocasiones conviene contar una historia desde varios medios: en el aula en una conferencia, en una clase práctica, o en los grupos de la asignatura de telegram. En esta tesis se propone una herramienta que permitirá justamente esto: escribir la historia una única vez y obtenerla en distintos formatos con los que los lectores/estudiantes/jugadores puedan interactuar.

Para lograr este trabajo Marian tuvo que incursionar en temas, técnicas y herramientas que no forman parte de su plan de estudio. Todo eso lo hizo de manera excelente, haciendo gala de lo buena científica de la computación que ya es.

De este tiempo de trabajo quiero señalar la seriedad, la calidad y el compromiso con que Marian realizó todas las tareas relacionadas con la tesis, sin dejar de lado otros compromisos como su docencia y actividades extradocentes.

Por todo lo anterior creo que estamos en presencia de un trabajo realizado por una excelente científica de la computación.

MSc. Fernando Raúl Rodríguez Flores
Facultad de Matemática y Computación
Universidad de la Habana
Junio, 2025

Resumen

En este trabajo se presenta CLIF: un lenguaje de dominio específico (DSL) desarrollado en Common Lisp para la creación de ficción interactiva. CLIF está orientado a contextos educativos en las áreas de matemática y computación. El DSL permite estructurar historias como grafos dirigidos de escenas, donde cada escena contiene elementos interactivos que pueden ser texto, enlaces, contenido multimedia, entre otros. CLIF permite exportar las historias a formatos de salida como Undum y LaTeX, utilizando macros y el sistema de objetos de Common Lisp (CLOS) para generar código de manera eficiente. El diseño del DSL se basa en un árbol de sintaxis abstracta (AST) que facilita la traducción a diferentes formatos, aprovechando características como el polimorfismo y la herencia múltiple de Common Lisp. CLIF es escalable y extensible a nuevos formatos de salida, ofreciendo flexibilidad a los desarrolladores y accesibilidad a nuevos usuarios.

Abstract

This work presents CLIF: a domain-specific language (DSL) developed in Common Lisp for creating interactive fiction. CLIF is aimed at educational contexts in the fields of mathematics and computer science. The DSL allows structuring stories as directed graphs of scenes, where each scene contains interactive elements that can be text, links, multimedia content, among others. CLIF enables exporting stories to output formats such as Undum and LaTeX, using macros and the Common Lisp Object System (CLOS) to efficiently generate code. The DSL design is based on an abstract syntax tree (AST) that facilitates translation to different formats, leveraging features like polymorphism and multiple inheritance in Common Lisp. CLIF is scalable and extensible to new output formats, offering flexibility to developers and accessibility to new users.

Índice general

Introducción	1
1. Elementos de narrativa y ficción interactiva	3
1.1. Teoría narrativa	3
1.2. Ficción interactiva	5
1.2.1. Ficción interactiva basada en analizadores sintácticos	5
1.2.2. Ficción interactiva de hipertexto	6
1.2.3. Ficción interactiva CYOA	7
1.3. Biblioteca Undum	9
1.3.1. Situaciones	9
1.3.2. Acciones	10
1.3.3. Cualidades	11
1.4. Aplicaciones educativas de la ficción interactiva	12
2. Elementos de diseño de lenguajes usando Common Lisp	13
2.1. Árboles de Sintaxis Abstracta (AST)	13
2.2. Lenguajes de Dominio Específico	14
2.3. Selección de Common Lisp como lenguaje del DSL CLIF	15
2.3.1. Sistema de Objetos de Common Lisp (CLOS)	16
2.3.2. Funciones Genéricas y Métodos	16
2.3.3. Notación prefija	17
2.3.4. Macros	18
2.4. MGA	19
3. Diseño conceptual del lenguaje CLIF	20
3.1. AST de CLIF	20
3.1.1. Nodos de contenido básico	21
3.1.2. Nodos de multimedia	22
3.1.3. Nodos de pizarra	24
3.1.4. Nodos de estilo y diseño	24
3.1.5. Nodos técnicos	25

3.1.6. Nodos de programación interactiva	26
3.1.7. Nodo escena	27
3.1.8. Formatos de salida	27
4. Detalles de Implementación	29
4.1. Núcleo del sistema	29
4.2. Generación de código	30
5. Ejemplo de una historia	32
Conclusiones	33
Recomendaciones	34
Referencias	35

Índice de figuras

1.1.	Obra <i>Patchwork Girl</i> , de Shelley Jackson	7
1.2.	Obra <i>afternoon, a story</i> , 4ta edición, folio abierto	7
1.3.	Serie de libros Lone Wolf	8
1.4.	Obra <i>To be or not to be</i> , de Ryan North	8
1.5.	Ejemplo de una situación en Undum	10
1.6.	Acción de abrir libro	11
1.7.	Opciones al presionar «abrir libro»	11
2.1.	Árbol de Sintaxis Abstracta.	14

Ejemplos de código

2.1.	Ejemplo de expresión para AST	15
2.2.	Ejemplo de función genérica	16
2.3.	Ejemplos de notación prefija	17
2.4.	Creación de nodo para párrafos en Undum	18
3.1.	Ejemplo de texto	21
3.2.	Ejemplo de enlace	21
3.3.	Ejemplo de título	21
3.4.	Ejemplo de lista ordenada	21
3.5.	Ejemplo de lista no ordenada con links	22
3.6.	Ejemplo de acción para agregar textos	22
3.7.	Ejemplo de imagen	22
3.8.	Ejemplo de sonido	23
3.9.	Ejemplo de cambio de volumen en audio	23
3.10.	Ejemplo de video	23
3.11.	Ejemplo de cambio de volumen en video	23
3.12.	Ejemplo de pizarra	24
3.13.	Ejemplo de borrar texto de la pizarra	24
3.14.	Ejemplo de cambiar imagen de fondo	24
3.15.	Ejemplo de código en Python	25
3.16.	Ejemplo de matemática inline	25
3.17.	Ejemplo de matemática centrada	25
3.18.	Ejemplo de declaración de variables	26
3.19.	Ejemplo de modificación de variables	26
3.20.	Ejemplo de comparación de variables	26
3.21.	Ejemplo de escena	27
3.22.	Nodo de enlace en Undum	27
4.1.	Ejemplos de nodos que heredan de <code>has-contents</code>	29
4.2.	Ejemplo del nodo <code>link</code>	30

Introducción

La ficción interactiva es un género narrativo digital que combina elementos de la literatura tradicional con mecánicas de videojuegos, donde el lector o jugador influye en el desarrollo de la historia mediante decisiones activas [1]. En un mundo cada vez más digitalizado [2], el formato virtual fomenta la participación activa del usuario, transformándolo en protagonista de la historia [3].

Compañías como Choice of Games [4], Hosted Games [5] y Delight Games [6] se enfocan en crear ficción interactiva orientada a los videojuegos. Más allá del entretenimiento, la ficción interactiva tiene aplicaciones en la educación, donde permite explorar conceptos complejos a través de la toma de decisiones y caminos personalizados [7].

La enseñanza y el aprendizaje de disciplinas afines a las matemáticas aplicadas se acelerarían con el uso de herramientas que permitan explorar las materias de manera didáctica y adaptativa [8].

En entornos académicos, los materiales educativos se presentan en formatos como libros o documentos digitales, lo que podría limitar la capacidad del lector para interactuar con el contenido y seguir una narrativa personalizada. Sin embargo, la ficción interactiva ha demostrado que es posible crear experiencias en las que el usuario toma decisiones que modifican el desarrollo de la historia [1]. Esta capacidad de integrarse en la narrativa permite que los lectores se sientan inmersos en la historia.

Para el diseño e implementación de una historia de ficción interactiva se pueden utilizar múltiples bibliotecas. Entre ellas se encuentran las herramientas basadas en texto, que permiten crear historias en un lenguaje similar al natural. Los sistemas *Inform 7* [9] y *TADS (Text Adventure Development System)* [10] son ejemplo de las herramientas basadas en texto. Otro medio para crear ficción interactiva son los motores de ficción interactiva de hipertexto [11], entre los que destacan *Twine* [12], *Ink* [13] y *Ren'Py* [14]. Además, existen bibliotecas [15], como *Evennia* [16] y *Undum* [17], que permiten integrar la ficción interactiva en aplicaciones.

Para utilizar cada uno de los programas anteriores se requieren conocimientos básicos acerca de la herramienta que se está utilizando. Si se desea cambiar de motor de ficción interactiva por cualquier motivo, habría que dedicar tiempo al estudio de la nueva herramienta que se quiere utilizar. Sin embargo, con la propuesta de esta tesis

se podrán exportar las historias creadas a varios formatos de salida como Undum, LaTeX [18] o bots de Telegram [19]. Un formato de salida es una traducción de una historia interactiva a un lenguaje o plataforma específica.

El objetivo de este trabajo es diseñar e implementar un lenguaje de dominio específico (DSL, por sus siglas en inglés) [20] para la creación de ficción interactiva orientada a historias de matemática y computación en el lenguaje Common Lisp. Además, el sistema permitirá exportar las historias interactivas a los formatos de salida definidos.

Para el cumplimiento de este objetivo general se han trazado los siguientes objetivos específicos:

- Estudiar las características y el funcionamiento de los sistemas existentes para la creación de ficción interactiva.
- Estudiar las principales características del lenguaje de programación Common Lisp, en el cual se desarrollará la solución.
- Modificar los sistemas de ficción interactiva que así lo requieran para incluir elementos matemáticos.
- Diseñar un lenguaje de dominio específico para describir historias interactivas.

La solución propuesta se basa en el desarrollo de un lenguaje de dominio específico (DSL, por sus siglas en inglés) en Common Lisp, aprovechando sus características como los macros y el sistema de objetos (CLOS). Llamaremos CLIF a este DSL.

CLIF incorpora herramientas para agregar elementos de matemática y programación a las historias, lo que facilita su uso educativo en áreas de matemática y computación.

El documento se estructura de la siguiente manera: El Capítulo 1 presenta los preliminares teóricos relacionados con narrativas y ficción interactiva. En el Capítulo 2 se profundiza en el concepto de árbol de sintaxis abstracta, así como en los lenguajes de dominio específico y en Common Lisp. A continuación, el capítulo 3 aborda temáticas relacionadas con el diseño conceptual del lenguaje CLIF, donde se describen los nodos que conforman el AST. Finalmente, el capítulo 4 analiza la implementación de código y hace referencia a cómo se realizó el proceso de generación de código en CLIF.

Capítulo 1

Elementos de narrativa y ficción interactiva

En este capítulo se presentan los preliminares del trabajo relacionados con narrativas y ficción interactiva. Se comienza describiendo elementos de la teoría narrativa en la sección 1.1. A continuación, la sección 1.2 aborda los fundamentos teóricos de la ficción interactiva, y en las subsecciones 1.2.1, 1.2.2 y 1.2.3 se describen sus principales subgéneros: juegos basados en analizadores sintácticos (*parser-based*), ficción interactiva de hipertexto y obras del estilo Elige Tu Propia Aventura (CYOA, por sus siglas en inglés). La sección 1.3 hace referencia a la biblioteca Undum como formato de salida para las historias generadas con CLIF. En la sección 1.4 se profundiza en las aplicaciones educativas de la ficción interactiva.

1.1. Teoría narrativa

La teoría narrativa ofrece un marco conceptual para comprender cómo se construyen y comunican las historias. Permite explorar las estructuras, elementos y procesos que subyacen en los textos narrativos, logrando un análisis de la forma en que el relato se organiza y se transmite al receptor.

Dentro de la teoría narrativa, se distinguen tres componentes en la construcción de historias [21, 22]. La **fábula** o «historia» constituye el nivel básico de los eventos narrados, representando la secuencia cronológica de lo que realmente ocurre. Por encima de esta, la **story** o «discurso» se refiere a la organización artística de esos eventos, es decir, cómo se presentan al receptor mediante técnicas como el punto de vista [23]. Finalmente, el **recurso narrativo** corresponde al medio o formato específico empleado para transmitir la historia, ya sea texto escrito, cine, videojuegos, entre otros. El recurso narrativo impone restricciones y posibilidades particulares para la expresión narrativa [24].

La fábula es una sucesión de eventos relacionados cuasalmente. Por ejemplo:

Una persona camina bajo la lluvia, se moja, ve una puerta abierta y entra a una habitación para refugiarse.

El discurso es cómo se presenta esa fábula. Siguiendo el ejemplo anterior, a continuación se tienen dos discursos diferentes para la misma fábula:

Era un día lluvioso. Una persona caminaba bajo la lluvia, sintiendo cómo el agua empapaba su ropa y cabello. De repente, vio una puerta abierta al final de la calle. Sin dudarlo, se apresuró hacia ella y entró en la habitación para protegerse del aguacero.

Entré en la habitación jadeando, empapado hasta los huesos. La lluvia golpeaba con fuerza afuera, y apenas hacía unos minutos caminaba por la calle, sin más abrigo que mi chaqueta mojada. Al ver la puerta abierta, supe que allí encontraría refugio.

En todos los ejemplos anteriores, el recurso narrativo empleado es el texto.

Estos tres conceptos resultan relevantes al analizar narrativas interactivas digitales porque el recurso narrativo modifica las relaciones entre fábula y *story* [25].

El objetivo de este trabajo es diseñar e implementar un lenguaje de dominio específico (DSL) para definir historias interactivas.

Una historia interactiva es una historia en la que los lectores pueden modificar lo que ocurre. Por ejemplo:

Entré en la habitación jadeando, empapado hasta los huesos. La lluvia golpeaba con fuerza afuera, y apenas hacía unos minutos caminaba por la calle, sin más abrigo que mi chaqueta mojada. Al ver la puerta abierta, supe que allí encontraría refugio.

Camino por la estancia donde encuentro un libro. Me pregunto si abrirlo será una buena idea ...

- > Abrir el libro
- > Dejar el libro sobre la mesa

Una historia interactiva se puede modelar como un grafo dirigido $G = (V, E)$, donde el conjunto de vértices V representa unidades narrativas como escenas o nodos de la historia. Las aristas $E \subseteq V \times V$ representan las transiciones o decisiones que un

usuario puede tomar para pasar de un nodo de la historia a otro.

Es de destacar el hecho de que las aristas son arcos dirigidos, pues las transiciones entre nodos modifican el orden cronológico de los acontecimientos de la historia. El grafo puede contener ramificaciones, uniones y bucles, representando, por ejemplo, múltiples finales o tramas paralelas [26, 27].

Una vez presentados los conceptos necesarios sobre teoría narrativa, en la siguiente sección se describe la ficción interactiva .

1.2. Ficción interactiva

La ficción interactiva es un género narrativo digital que combina elementos de la literatura tradicional con mecánicas de videojuegos, donde el lector o jugador influye en el desarrollo de la historia mediante decisiones activas [28]. Por ejemplo, el lector puede determinar qué personajes sobreviven o mueren, qué caminos sigue la trama y qué finales se alcanzan [1]. Entre las obras que identifican este género se incluyen *Resuelve el Misterio* [29], *Las aventuras interactivas de Sherlock Holmes: Escándalo en Bohemia* [30] y *Guardian Maia* [31].

El libro *Twisty Little Passages* [1] aborda la ficción interactiva desde una perspectiva tanto literaria como de entretenimiento. Montfort traza la evolución del género comenzando con acertijos y juegos de texto como *Adventure* [32] y *Zork* [33], hasta llegar a la era de la ficción interactiva comercial y el auge de la comunidad independiente en los años 90. En *Twisty Little Passages* se analiza la ficción interactiva como un híbrido entre narrativa, juego y programa computacional, abriendo un espacio para su reconocimiento como forma literaria [34].

Por otro lado, *Interactive Fiction as Literature* [35] es una obra que profundiza en la consideración de la ficción interactiva como una forma literaria legítima y compleja. Argumenta que la ficción interactiva no debe ser vista únicamente como un juego, sino como una narrativa que despliega nuevas formas de comprensión y experiencia estética. Esto se explica debido a que la ficción interactiva permite al lector o jugador explorar múltiples caminos que transforman la trama y los personajes.

Dentro de la ficción interactiva se pueden identificar tres subgéneros principales según su formato y medio de interacción: ficción interactiva basada en analizadores sintácticos (parser-based), ficción interactiva de hipertexto y ficción interactiva CYOA. En la siguiente sección se explican los juegos de tipo basados en analizadores sintácticos.

1.2.1. Ficción interactiva basada en analizadores sintácticos

Existen los juegos de ficción interactiva basados en analizadores sintácticos (**parser-based**, por su traducción al inglés) [36], también llamados «aventuras de texto». En

este tipo de juegos, los jugadores escriben instrucciones y el intérprete del juego las analiza para determinar qué ocurre a continuación.

Retomando la historia de la lluvia, a continuación se muestra un ejemplo de adaptación de la historia a un juego *parser-based*:

Estás caminando bajo una lluvia intensa. El agua empapa tu ropa y sientes frío. Al final de la calle, ves una puerta abierta. ¿Qué haces?

> entrar

Entras a la habitación...

Los sistemas de creación de juegos de análisis sintáctico incluyen Inform7 [9] y TADS [10]. Como ejemplo de lo que se puede producir con motores de análisis sintáctico está *Fotopia* [37], escrito por Adam Cadre y publicado en 1998. El reconocimiento de esta obra se debe a su accesibilidad y a servir como una introducción al género, ya que no requiere conocimientos avanzados de comandos por parte del jugador.

Una de las principales ventajas de este tipo de juegos es la libertad de acción que se otorga al lector o jugador, quien puede decidir de manera activa qué hacer y cómo interactuar con el entorno [36].

Sin embargo, esta libertad también conlleva ciertas limitaciones, pues si el jugador introduce comandos que no han sido programados por el autor, el sistema no los reconoce, lo que puede crear frustración por parte del jugador [1]. No obstante, los motores de parser actuales, como Inform7 y TADS [9, 10], han evolucionado para ofrecer flexibilidad en la interpretación de las acciones del usuario.

Además de la ficción interactiva basada en analizadores sintáticos, existe la ficción interactiva de hipertexto. En la siguiente sección se describen las obras de dicho género.

1.2.2. Ficción interactiva de hipertexto

La ficción interactiva de estilo **hipertexto** se desarrolla en páginas web, donde el usuario pulsa en los enlaces y accede a distintas páginas como respuesta. [11]. Como resultado de este estilo, se crearon las obras «*Patchwork Girl*» (Figura 1.1) [38] y «*afternoon, a story*» (Figura 1.2) [39].

«*Afternoon, a story*», publicada en 1990 es considerada la primera obra de narrativa hipertextual, llegando a considerarse un ejemplo clásico de literatura modernista [40]. Por otro lado, «*Patchwork Girl*», del año 1995, explora la identidad, el cuerpo y el género en una época donde eran considerados temas tabú [41].

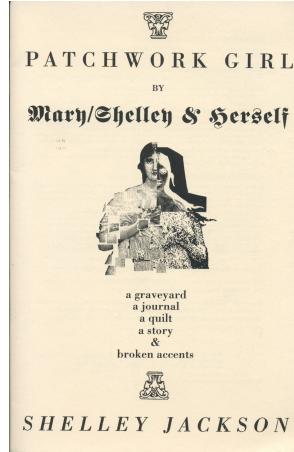


Figura 1.1: Obra *Patchwork Girl*, de Shelley Jackson

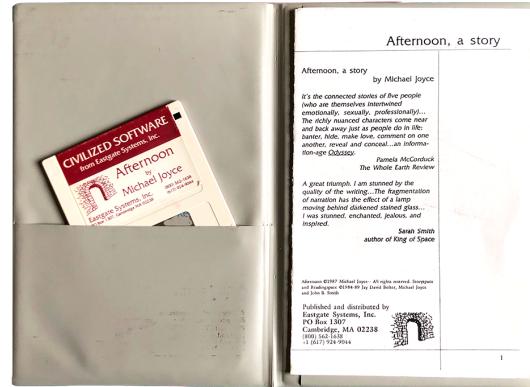


Figura 1.2: Obra *afternoon, a story*, 4ta edición, folio abierto

A partir de la ficción interactiva de hipertexto [11] surgen las obras del estilo Elige tu propia aventura, acerca de las cuales se profundiza en la siguiente sección.

1.2.3. Ficción interactiva CYOA

Las obras del estilo Choose Your Own Adventure (**CYOA**, por sus siglas en inglés) [42] o *choice-based*, se caracterizan por presentar al lector opciones periódicas que determinan el curso de la narrativa. Las obras CYOA presentan una estructura ramificada con múltiples caminos que, dependiendo de las elecciones, pueden llevar a diferentes finales [43].

El género CYOA tiene sus raíces en libros impresos de los años 70-80 [44], pero ha encontrado auge en formatos digitales, donde la navegación entre secciones y el seguimiento de estados se automatizan [45].

A principios de la década de 2010, el lanzamiento del *software* Twine [12] abrió el desarrollo de CYOA a nuevos creadores [42]. Tras el surgimiento de Twine, varios desarrolladores comerciales como Choice of Games [4] e Inkle [46] comenzaron a producir ficción interactiva con fines de lucro.

Entre los juegos comerciales en el ámbito de la ficción interactiva CYOA, podemos mencionar *80 Days*, un juego de aventuras basado en la novela de Julio Verne «*La vuelta al mundo en 80 días*» [47], que combina narrativa interactiva con exploración global [48] y *Choice of Robots*, la cual es una novela interactiva sobre la creación y evolución de robots, con finales múltiples según las decisiones del jugador [49].

Las herramientas que se utilizan para el desarrollo de ficción interactiva CYOA incluyen Twine, Undum [17], Squiffy [50] y ChoiceScript [51].

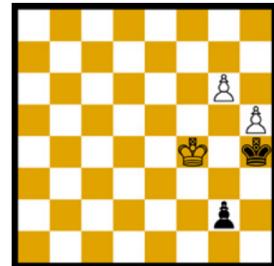
Twine permite a los autores diseñar narrativas sin necesidad de conocimientos avanzados de programación. Además brinda facilidad para integrar multimedia, exportar directamente a HTML y cuenta con recursos y extensiones. Sin embargo, una desventaja importante de Twine es que no es un motor de videojuegos completo, por lo que carece de soporte nativo para gráficos avanzados o audio integrado.

Por otro lado, Undum utiliza HTML, CSS y JavaScript, ofreciendo control sobre la presentación y la lógica del juego. Esta flexibilidad facilita la creación de historias con variables múltiples y respuestas dinámicas a las elecciones del jugador. Undum permite manejar narrativas ramificadas e integrar elementos multimedia. Por las ventajas mencionadas anteriormente, Undum se encuentra entre los formatos de salida del DSL CLIF.

Entre las obras de tipo CYOA más famosas se encuentran la novela romántica de 2018 *My Lady's Choosing* [52], la adaptación humorística de 2013 *To Be or Not To Be: A Chooseable-Path Adventure* [53] y la aventura espacial *Space and Beyond* [54], del año 1983. Además, se destaca la serie de libros *Lone Wolf* (Figura 1.3) [55], con más de 15 libros publicados.



Figura 1.3: Serie de libros Lone Wolf



Be careful: there's no way to win at chess if you're down to just your king. That one pawn you've got is worth its weight in gold, and losing it will absolutely cost you victory.

There are three moves you can make here. What do you do?

50

MOVE THE PAWN AHEAD TWO SQUARES (G4): turn to page 623

MOVE THE PAWN AHEAD ONE SQUARE (G3): turn to page 613

MOVE THE KING BACK A SQUARE (K_H3): turn to page 644

Figura 1.4: Obra *To be or not to be*, de Ryan North

Uno de los formatos de salida del DSL CLIF es Undum. En la siguiente sección se describen sus características.

1.3. Biblioteca Undum

Esta tesis permite generar historias de ficción interactiva en varios formatos de salida. Un formato de salida es una traducción de una historia interactiva a un lenguaje o plataforma específica que pueda ser visualizada o ejecutada para un usuario. Uno de estos formatos es Undum [17].

Undum es un sistema de creación de historias de estilo CYOA que se pueden reproducir en navegadores web [56]. Esta biblioteca, creada por I.D. Millington en 2010, permite crear narrativas dinámicas donde la historia se cuenta en fragmentos y se seleccionan diversas opciones para avanzar. Undum funciona completamente con código Javascript y HTML5, sin requerir servidores externos [56]. Las historias se muestran mediante contenido de tipo texto, audio o imágenes en una página web.

Los juegos de Undum se basan en tres conceptos: situaciones, acciones y cualidades. En las siguientes secciones se describen cada uno de ellos, comenzando con las situaciones.

1.3.1. Situaciones

Una situación es un fragmento de código responsable de agregar contenido a la página web y responder a la interacción del usuario [17]. Toda interacción del usuario se realiza haciendo clic en enlaces dentro del contenido.

Por ejemplo, en la figura 1.5 se muestra el texto correspondiente a una situación en Undum con el comienzo de la historia de la lluvia usada al principio del capítulo.

Usualmente, un enlace cambiará la situación actual a otra situación que puede escribir en la pantalla y comenzar a responder a nuevas interacciones. En el ejemplo anterior, si se presiona el texto en rojo, cambiaremos de la situación actual a una nueva situación donde el personaje se habrá acercado a la puerta abierta.

Cuando una situación cambia, todos los enlaces previamente disponibles se eliminan para evitar que el jugador retroceda para probar opciones alternativas después de comprometerse con una.

En todo momento hay exactamente una situación activa. Estas situaciones, junto con los enlaces entre ellas, forman la estructura del juego.

En la definición de historia narrativa en el contexto de los grafos, las situaciones serían los nodos del grafo, mientras que los enlaces entre situaciones constituyen las aristas.

Desde el punto de vista de la historia, una situación es una unidad narrativa que define un estado particular del relato. El elemento análogo a una situación en libros de cuentos tradicionales sería una escena.

El segundo elemento de Undum a tener en cuenta para crear historias interactivas son las acciones, las cuales se describen en la siguiente sección.

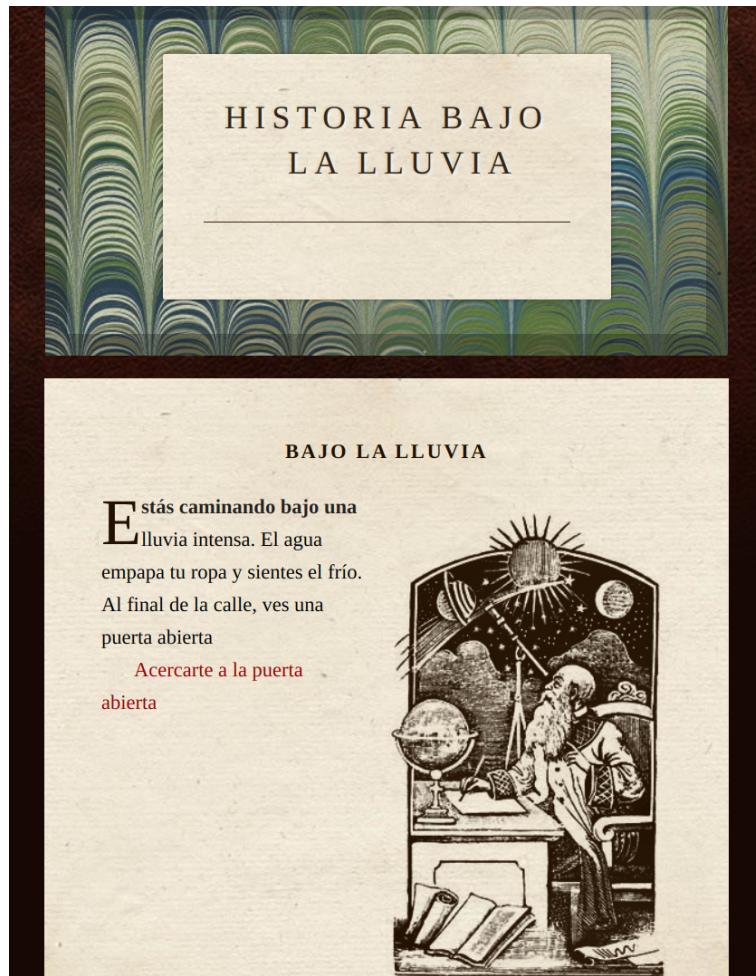


Figura 1.5: Ejemplo de una situación en Undum

1.3.2. Acciones

Una situación puede ofrecer al jugador una serie de acciones para realizar. Estas acciones solo ocurren dentro de dicha situación y normalmente no provocan un cambio de situación.

Las acciones en Undum representan operaciones interactivas de mayor complejidad que los simples enlaces entre situaciones. Estas permiten tres tipos fundamentales de modificaciones narrativas: pueden alterar el contenido presente en una situación mediante la incorporación de nuevo material textual, pueden transformar las cualidades del personaje o del mundo narrativo, y tienen la capacidad de generar enlaces condicionales.

Un ejemplo sería una acción de «examinar» que no solo revela información oculta

dentro de la misma situación, sino que potencialmente desbloquea nuevas rutas narrativas, todo ello sin requerir necesariamente una transición inmediata a otra situación. A continuación se muestra un ejemplo siguiendo la historia de la lluvia:



Figura 1.6: Acción de abrir libro

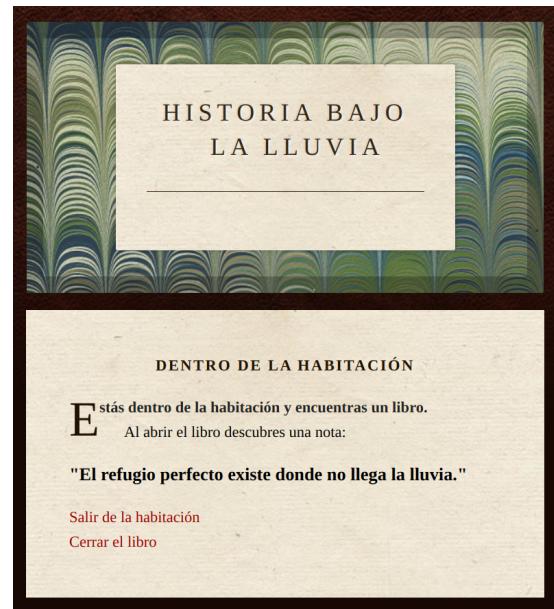


Figura 1.7: Opciones al presionar «abrir libro»

Cuando tenemos un personaje, resulta útil poder definir estados en los que se encuentra. Para eso podemos usar las cualidades en Undum, que se describen en la siguiente sección.

1.3.3. Cualidades

Las cualidades representan el estado actual del personaje o del mundo mediante valores numéricos [17].

Las cualidades se pueden mostrar de distintas formas: mediante una barra de progreso, un símbolo, una palabra, un entero, etc [17]. Por lo tanto, desde la perspectiva del usuario, las cualidades pueden representar cualquier tipo de valor.

Por ejemplo, en la historia de la lluvia, una cualidad podría ser: “probabilidad de que siga lloviendo” o “velocidad de lectura del personaje”.

Las situaciones, acciones y cualidades de Undum permiten crear historias interactivas con flexibilidad.

Además del entretenimiento, la ficción interactiva se ha usado con enfoques educativos. En la siguiente sección se abordan las aplicaciones educativas de la ficción

interactiva.

1.4. Aplicaciones educativas de la ficción interactiva

La ficción interactiva promueve la imaginación y el pensamiento innovador al brindar a los lectores libertad de elección [3]. Puede utilizarse para aprender un idioma extranjero, como lo muestra la plataforma LingoStories [57] mediante la selección de aventuras según el nivel de idiomas del usuario; para estudiar matemáticas, a través del libro interactivo *The Dregg Disaster: An Algebra I Gamebook* [58], que obliga a resolver problemas algebraicos para avanzar en la narrativa; e incluso para comenzar a programar desde la app móvil *Code Adventures* [59] con los rompecabezas que propone.

El libro *Teaching and Learning with Interactive Fiction* [60] explora el valor pedagógico de la ficción interactiva como herramienta educativa que transforma al estudiante en un participante activo y responsable de su propio aprendizaje. La obra analiza cómo la capacidad de elección, la manipulación del entorno narrativo y la participación activa promueven los principios educativos de creación, identidad y distribución del conocimiento.

Los estudios [3, 61] muestran que complementar la educación formal con ficción interactiva aumenta la motivación en áreas de STEM (Science - Technology - Engineering - Mathematics, por sus siglas en inglés), particularmente en grupos subrepresentados, como las minorías étnicas y las personas con discapacidades. Por ejemplo, se ha logrado un aumento de retención en cursos introductorios de programación [62].

El objetivo de esta tesis es crear ficción interactiva con enfoques educativos en las áreas de la matemática y la computación. Para lograrlo, se quieren tener las historias en distintos formatos para que una parte se desarrolle, por ejemplo, en Undum, y otra en un bot de Telegram. Es por esto que sería conveniente tener la misma historia representada en varios formatos. Una herramienta muy conveniente para esto es Common Lisp, como se muestra en el próximo capítulo.

Capítulo 2

Elementos de diseño de lenguajes usando Common Lisp

En este capítulo se describen los fundamentos teóricos relacionados con el diseño de lenguajes. Se comienza abordando la definición de árbol de sintaxis abstracta (AST, por sus siglas en inglés) en la sección 2.1. La sección 2.2 introduce los conceptos de Lenguajes de Dominio Específico (DSLs), mientras que la sección 2.3 justifica la selección de Common Lisp como lenguaje para implementar el DSL CLIF, destacando características como su sistema de objetos (CLOS), funciones genéricas y macros. Finalmente, en la sección 2.4 se analiza la herramienta de creación de generadores multilenguaje MGA.

2.1. Árboles de Sintaxis Abstracta (AST)

Los Árboles de Sintaxis Abstracta (AST) representan jerárquicamente, mediante un árbol, la estructura sintáctica abstracta o simplificada de un programa escrito en un lenguaje de programación. Estos árboles son fundamentales en el proceso de creación de lenguajes y en la fase de generación de código [63].

En los AST, cada nodo representa una instrucción del lenguaje. Estos nodos contienen la información necesaria para la generación de código de la instrucción que representa [64]. Por ejemplo, si el lenguaje posee la instrucción declaración de variable, durante la construcción del AST es necesario definir un nodo referente a esta instrucción mediante una clase.

El árbol es abstracto en el sentido de que no representa cada detalle que aparece en la sintaxis verdadera, es decir, no contiene los paréntesis y otros símbolos que inevitablemente aparecen en el código. Los ASTs poseen gran importancia en la construcción de los lenguajes debido a que pueden intervenir en varias fases del análisis:

como producto del análisis sintáctico, como elemento intermedio en sucesivos análisis semánticos y como entrada para la generación de código [64].

En la figura 2.1 se muestra un ejemplo de un árbol de sintaxis abstracta que representa a la instrucción de asignación $d = a + b * c$. El AST que se muestra tiene como raíz el nodo que representa la operación (=); tanto la parte izquierda (d) como la parte derecha ($a + b * c$) son representadas como subárboles hijos del nodo raíz. De una forma similar ocurre con el nodo (+) y el nodo (*).

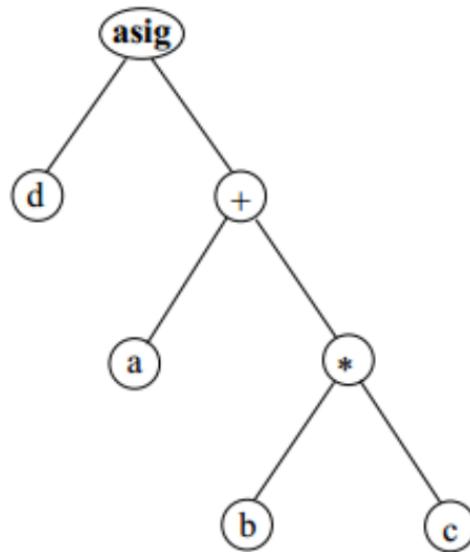


Figura 2.1: Árbol de Sintaxis Abstracta.

En la siguiente sección se describen los lenguajes de dominio específico.

2.2. Lenguajes de Dominio Específico

Un Lenguaje de Dominio Específico (DSL, por sus siglas en inglés *Domain-Specific Language*) es un lenguaje de programación optimizado para una clase específica de problemas [20]. Un DSL utiliza los conceptos y reglas de un campo o dominio particular, lo que los hace más expresivos para su área de aplicación. Ejemplos de DSLs incluyen SQL (*Structured Query Language*) [65], diseñado para gestionar y consultar bases de datos relacionales [66]; HTML (*HyperText Markup Language*) [67], enfocado en la estructuración de contenido web [68]; LaTeX [18], especializado en composición tipográfica de documentos académicos [69]; y MATLAB [70], ampliamente utilizado en computación numérica y modelado científico [71].

El uso de DSLs otorga claridad en los códigos ya que su sintaxis especializada elimina la necesidad de incluir detalles técnicos de implementación [20]. Los DSLs permiten una mayor expresividad y concisión en tareas específicas, lo que facilita el desarrollo, mantenimiento y validación de soluciones.

El objetivo de este trabajo es diseñar un DSL para la creación de ficción interactiva de forma que se obtengan las historias en distintos formatos de salida como Undum y bots de Telegram. Para lograr esto, se usó el lenguaje Common Lisp. En la siguiente sección se explican las características de Common Lisp que motivaron a su selección como lenguaje para la implementación del DSL CLIF.

2.3. Selección de Common Lisp como lenguaje del DSL CLIF

Lisp es un lenguaje de programación creado originalmente en 1958 por John McCarthy y sus colaboradores en el Instituto Tecnológico de Massachusetts [72]. Su nombre deriva de LISt Processing (Procesamiento de listas). Es un lenguaje de programación multiparadigma, que permite escribir programas de manera imperativa, funcional, o declarativa [73, 74]. Debido a la variedad de dialectos Lisp existentes en 1981 (Maclisp, Interlisp, zetalisp) se creó el estandar Common Lisp en 1986 [72].

Entre las características más significativas de Common Lisp se encuentran la sintaxis con notación prefija en la que todas las expresiones se encuentran encerradas entre paréntesis. Se seleccionó Common Lisp para realizar la implementación del DSL CLIF debido a las ventajas que ofrece como son la presencia de herencia múltiple, el uso de funciones genéricas que permiten tener polimorfismo en múltiples argumentos y el uso de los macros. [73, 74].

Al implementar la solución en Common Lisp y usar su sistema de macros, fue posible obtener el AST de los lenguajes de salida de CLIF sin necesidad de implementar el análisis lexicográfico ni sintáctico. Esto ocurre porque el código ya está representado internamente como un árbol de sintaxis abstracta que las macros pueden procesar y reescribir directamente. Por tanto, fue posible reducir el proceso de compilación de un lenguaje al diseño de las clases que forman los nodos del AST y la realización de la fase de generación de código como se muestra en el capítulo 4. Además, al usar la sintaxis prefija de Common Lisp y seleccionar nombres apropiados para los constructores de estas clases, se pudo obtener el AST directamente. Por ejemplo, una expresión como:

```
1 (scene inicio (text "Texto de ejemplo"))
```

Ejemplo de código 2.1: Ejemplo de expresión para AST

representa un AST donde `scene` es el nodo raíz y los elementos restantes son sus hijos.

En las siguientes secciones se abordan las principales características de Lisp que fueron necesarias para la implementación de CLIF, comenzando con su sistema de objetos.

2.3.1. Sistema de Objetos de Common Lisp (CLOS)

El Sistema de Objetos de Common Lisp (CLOS) es un elemento clave del lenguaje, pues define conceptos tales como objetos, clases y métodos. CLOS se considera multiparadigma [75] porque integra principios de programación orientada a objetos, como herencia y polimorfismo, junto con características funcionales.

Entre las características funcionales de CLOS se incluyen las funciones de primera clase. Las funciones de primera clase son entidades que pueden ser tratadas como cualquier otro valor en un lenguaje de programación, lo que significa que pueden asignarse a variables, pasarse como argumentos a otras funciones o devolverse como resultados.

Otra de las características funcionales de CLOS es la evaluación basada en expresiones, que se caracteriza por que las operaciones se definen como expresiones que producen valores. Las estructuras de control como condicionales o bucles devuelven valores, facilitando un estilo de programación declarativo.

El sistema de objetos de Common Lisp difiere de otros lenguajes como Python, C# o Java en algunas características; por ejemplo, los métodos no pertenecen a ninguna clase y permite el polimorfismo en múltiples argumentos [75].

CLOS simplifica la creación, extensión y mantenimiento del DSL CLIF, pues permite definir clases abstractas y métodos genéricos. En la siguiente sección se profundiza en las funciones genéricas y métodos en Common Lisp.

2.3.2. Funciones Genéricas y Métodos

Una función genérica define una operación abstracta, especificando su nombre y una lista de parámetros, pero sin implementación. Estas funciones son genéricas debido a que pueden recibir cualquier objeto como argumento; sin embargo, no ejecutan ningún código en sí mismas. [73]. Las funciones genéricas en Lisp, al igual que las funciones ordinarias, reciben como entrada una lista de argumentos y realizan una serie de operaciones [73].

Por ejemplo, a continuación se define una función genérica `describe` que invoca diferentes métodos según el tipo del argumento.

```

1  (defgeneric describe (obj)
2    (:documentation "Describe el objeto recibido."))
3
4  (defmethod describe ((obj number))

```

```

5   (format t "Es un numero: ~a~%" obj))

6
7   (defmethod describe ((obj string))
8     (format t "Es una cadena de texto: \"~a\"~%" obj))

9
10  (describe 42)           ; Salida: Es un numero: 42
11  (describe "Hola")      ; Salida: Es una cadena de texto: "Hola"

```

Ejemplo de código 2.2: Ejemplo de función genérica

Si solo se define una función genérica, independientemente de los argumentos con los que se llame, se genera un error. Los métodos proporcionan la implementación real de las funciones genéricas.

Cada método proporciona una implementación de la función genérica para clases particulares de argumentos. Estos métodos indican qué tipos de argumentos pueden manejar especializando los parámetros. En el código anterior se puede observar que el primer método especializado permite recibir un argumento de tipo `number`, mientras que el segundo método recibe un argumento de tipo `string`.

El uso de las funciones genéricas posee ventajas debido a que admiten que los métodos especializados tengan múltiples parámetros, proporcionan un marco que hace que la herencia múltiple sea manejable, y permiten usar construcciones declarativas, admitiendo usos comunes sin mucho código.

En CLIF, las funciones genéricas de CLOS se utilizan para generar código. Para ello se emplea la función `generate-code`, que se especializa simultáneamente en un nodo del AST y un formato de salida.

En la siguiente sección se describe en qué consiste la notación prefija de Common Lisp.

2.3.3. Notación prefija

La sintaxis de notación prefija es una característica de Lisp que diferencia su estructura de la mayoría de lenguajes de programación. En este esquema, los operadores preceden a sus operandos, lo que permite una representación uniforme de código y datos.

En Lisp, toda expresión se escribe como una lista delimitada por paréntesis, donde el primer elemento es la función u operador, y los siguientes son sus argumentos. Por ejemplo:

```

1   (+ 2 3)          ; Equivalente a 2 + 3 en notacion infija
2   (* (+ 1 2) 4)    ; Equivalente a (1 + 2) * 4

```

Ejemplo de código 2.3: Ejemplos de notación prefija

CLIF aprovecha esta notación para definir su AST y las transformaciones entre formatos, empleando nombres adecuados para los constructores. Cada nodo del árbol se construye mediante expresiones prefijas que especifican el tipo de elemento y sus atributos:

```

1  (defnode html-p (has-contents has-css-classes)
2    ()
3    :documentation "A class to represent a <p> tag in html
4      .
      "
      :string-obj ("<>p ~a: ~a>" css-classes contents))

```

Ejemplo de código 2.4: Creación de nodo para párrafos en Undum

Cuando los nombres de las funciones no son suficientes, se emplean macros. En la siguiente sección se describen las macros de Common Lisp.

2.3.4. Macros

Un macro es una función que genera código Lisp: un programa que genera programas [73].

Las macros tienen un funcionamiento diferente al de las funciones. Una función produce un resultado, pero una macro produce una expresión que al ser evaluada produce un resultado [73]. Esta conversión de expresión a resultado se realiza antes de que se evalúe el código fuente, y el paso que construye la nueva expresión que será evaluada se llama macro-expansión.

Las macros ofrecen ventajas al permitir reutilizar patrones de código a lo largo del programa sin necesidad de repetirlos explícitamente, reduciendo así la cantidad de código escrito. Además, al expandirse en tiempo de compilación, el código generado por las macros no incurre en sobrecarga en tiempo de ejecución, ofreciendo eficiencia sin sacrificar expresividad.

Durante el diseño de CLIF, las macros se utilizaron en el proceso de compilación, evitando así tener que realizar el análisis lexicográfico y sintáctico. Por tanto, se pudo pasar directamente a la generación de código una vez construido el AST.

Durante el diseño de CLIF, las macros se utilizaron en el proceso de compilación, evitando así tener que realizar un análisis lexicográfico y sintáctico explícito. Esto permitió pasar directamente a la generación de código una vez construido el Árbol de Sintaxis Abstracta (AST).

Para lograr esto, fue necesario abordar tres aspectos fundamentales: definir los nodos del AST, diseñar constructores adecuados y generar código a partir del AST,

implementando la traducción de cada nodo a los distintos formatos de salida soportados.

En la Facultad de Matemática y Computación de la Universidad de La Habana se han desarrollado varios trabajos previos con características similares [76, 77], donde se identificaron patrones recurrentes en este tipo de procesos. En particular, en el año 2017, se realizó un trabajo que permitió automatizar gran parte de estas tareas. En la siguiente sección describiremos la herramienta que surgió de este desarrollo: MGA.

2.4. MGA

La herramienta MGA (Multi-languages Generators Automation, por sus siglas en inglés) [77] es una biblioteca desarrollada en Common Lisp para automatizar la creación de generadores multilenguaje. Un generador multilenguaje es un programa que permite crear aplicaciones capaces de traducir programas escritos en un DSL a múltiples lenguajes de salida.

El diseño de un lenguaje se simplifica a dos aspectos fundamentales: la definición de los nodos del AST y la creación de constructores adecuados para esos nodos. Los constructores pueden ser funciones en Lisp o macros, dependiendo de las necesidades específicas del nodo.

Una vez construido el AST, durante el proceso de generación de código ocurren patrones repetitivos en el método `generate-code`. Aquí es donde MGA demuestra su utilidad: automatiza tanto la definición de las clases de los nodos como la creación de sus constructores.

Por ejemplo, en condiciones normales, el desarrollador tendría que definir manualmente la clase y su constructor, ya sea como una función o una macro. Con MGA, ambas tareas se realizan simultáneamente mediante el macro `defnode`, donde solo es necesario especificar la clase y el nombre del constructor, agilizando significativamente el proceso de desarrollo. Además, el macro `gcode` simplifica la generación de código al encapsular patrones comunes, reduciendo así la cantidad de código repetitivo que debe escribirse.

En el siguiente capítulo se describe el diseño conceptual del lenguaje CLIF.

Capítulo 3

Diseño conceptual del lenguaje CLIF

CLIF (*Common Lisp Interactive Fiction*, por sus siglas en inglés) es un sistema diseñado para crear historias interactivas del estilo CYOA. Las historias escritas en CLIF se pueden exportar a varios motores de ficción interactiva. CLIF está orientado a generar historias interactivas en el contexto de las asignaturas de matemática y computación.

La estructura de una historia en CLIF se compone de escenas, que representan diferentes momentos dentro de la narrativa. Cada escena contiene texto descriptivo, elementos interactivos y posibles transiciones hacia otras escenas. Esto permite al autor definir un flujo dinámico donde las decisiones del usuario alteran el curso de la historia.

Una de las características de CLIF es su capacidad para generar contenido dinámico mediante la pizarra virtual. Además, existe una herramienta para agregar código. Esto resulta especialmente útil para presentar ejemplos, fórmulas o instrucciones que varían según el contexto.

Como se usó Common Lisp para el desarrollo de CLIF, para obtener los elementos del lenguaje es necesario definir constructores de las clases que representan a los nodos del AST. Los constructores de estas clases se definen de manera que simplifiquen la creación de instancias y aseguren la consistencia del AST.

En la siguiente sección se explican los nodos que conforman el AST de CLIF.

3.1. AST de CLIF

A continuación, se describen los tipos de nodos que componen el AST (Abstract Syntax Tree) de CLIF, los cuales permiten estructurar las narrativas interactivas.

3.1.1. Nodos de contenido básico

Los nodos de contenido básico son útiles para crear la estructura principal de la narrativa. Los nodos de contenido básico son los nodos de texto, enlace, títulos de sección, listas y acciones. A continuación se brindan detalles de cada uno de ellos:

- Nodo de texto: Representa bloques de texto. Es el elemento básico para construir narrativas. Recibe una lista de elementos de texto. Por ejemplo:

```
1 (text "Ante ti hay una... " "puerta.")
```

Ejemplo de código 3.1: Ejemplo de texto

- Nodo de enlace: Se utiliza para cambiar entre situaciones o llevar a diferentes secciones de la historia. Recibe el destino al que irá el usuario y una lista de textos. Por ejemplo:

```
1 (link :to situacion-2
2   :text "Abrir la puerta.")
```

Ejemplo de código 3.2: Ejemplo de enlace

- Nodo de títulos de sección: Permite dar un título a las secciones. Es útil para dividir la historia en partes o resaltar información importante. recibe una lista de elementos de texto. Por ejemplo:

```
1 (title "Bajo la lluvia.")
```

Ejemplo de código 3.3: Ejemplo de título

- Nodo de lista: Permite crear listas que pueden usarse para enumerar opciones, pasos o elementos. Los nodos de lista reciben una lista de elementos de texto. Para crear los elementos de las listas, se utilizan los nodos de elemento de listas Estos nodos pueden recibir cualquier objeto de CLIF. En CLIF hay dos tipos de listas:

- Listas ordenadas: son aquellas que tienen algún tipo de ordenación definida, ya sea numérica, alfabética, etc. Por ejemplo:

```
1 (ord_list
2   (item "Elemento 1")
3   (item "Elemento 2"))
```

Ejemplo de código 3.4: Ejemplo de lista ordenada

- Listas no ordenadas: no tienen un orden definido. Por ejemplo:

```

1  (unord_list
2    (item (link entrar "Abres la puerta"))
3    (item (link quedarse "Decides esperar afuera"))
)

```

Ejemplo de código 3.5: Ejemplo de lista no ordenada con links

- Nodo de acción para agregar texto: Permite llevar a cabo una acción de agregar texto al activarlo. Recibe una función que define cómo mostar el texto y una lista de textos. Por ejemplo:

```

1  (action
2   :func show-text
3   ("Clic para mostrar un texto"
4   "Texto a mostrar al hacer clic"))

```

Ejemplo de código 3.6: Ejemplo de acción para agregar textos

En la siguiente sección definimos otro tipo de nodos: los nodos de multimedia.

3.1.2. Nodos de multimedia

Los nodos de multimedia acompañan la narrativa con imágenes, sonido y videos. En CLIF están concebidos los siguientes tipos de nodos multimedia:

- Nodo de imágenes: Permite incrustar imágenes en la historia con los atributos tamaño, alineación y texto alternativo. Por ejemplo:

```

1  (image :alignment centered
2   :alt-text "Texto alternativo"
3   :src "/tmp/images/image.png")

```

Ejemplo de código 3.7: Ejemplo de imagen

- Nodos de sonido: Son útiles para ambientar escenas o proporcionar información auditiva. Reciben un texto alternativo opcional, la ubicación del archivo y, en el caso del nodo de cambio de volumen, el nuevo volumen. Los nodos de sonido se dividen en tres tipos:

- Comenzar sonido: Inicia la reproducción de un sonido. Por ejemplo:

```

1  (start_sound :alt-text "Sonido de lluvia de
2      fondo"
3      :src "/tmp/sound/rain.mp3")

```

Ejemplo de código 3.8: Ejemplo de sonido

- Detener sonido: Detiene la reproducción de un sonido.
- Cambiar volumen: Ajusta el volumen de un sonido en reproducción. Por ejemplo:

```

1  (change_vol :alt-text "Sonido de lluvia de fondo"
2      :src "/tmp/sound/rain.mp3"
3      :vol 85)

```

Ejemplo de código 3.9: Ejemplo de cambio de volumen en audio

- Nodos de video: Se emplean para agregar videos a las historias y permiten manejar el video. Reciben un texto alternativo opcional, la ubicación del archivo y, en el caso del nodo de cambio de volumen, el nuevo volumen. Se incluyen los siguientes tipos de nodos de video:

- Comenzar video: Inicia la reproducción de un video. Por ejemplo:

```

1  (start_video :alt-text "Video de lluvia"
2      :src "/tmp/video/rain.mp4")

```

Ejemplo de código 3.10: Ejemplo de video

- Parar video: Detiene la reproducción de un video.
- Cambiar volumen: Ajusta el volumen de un video en reproducción. Por ejemplo:

```

1  (start_video :alt-text "Video de lluvia"
2      :src "/tmp/video/rain.mp4"
3      :vol 70)

```

Ejemplo de código 3.11: Ejemplo de cambio de volumen en video

Una forma de mostrar contenidos relevantes en los centros educativos es mediante el uso de la pizarra. CLIF incluye soporte para una pizarra digital, cuyos nodos se describen en la siguiente sección.

3.1.3. Nodos de pizarra

La pizarra es un elemento interactivo que simula una superficie donde se puede escribir y borrar contenido dinámicamente. Los nodos asociados incluyen:

- Mostrar_pizarra: Hace visible la pizarra. Recibe el `id` de la pizarra a mostrar. Opcionalmente, recibe un contenido inicial título, texto y/o autor. Por ejemplo:

```

1  (show_whiteboard :id "wb"
2           :title "Titulo de la pizarra"
3           :text "Texto inicial")
```

Ejemplo de código 3.12: Ejemplo de pizarra

- Ocultar_pizarra: Elimina la pizarra de la vista. Recibe el `id` de la pizarra a ocultar.
- Borrar_pizarra: Limpia todo el contenido de la pizarra. Recibe el `id` de la pizarra a borrar.
- Eliminar_elementos: Elimina elementos específicos de la pizarra. Recibe el `id` de la pizarra a borrar y el contenido que se desea eliminar. Por ejemplo:

```

1  (erase_whiteboard :id "wb"
2           :del "Titulo de la pizarra")
```

Ejemplo de código 3.13: Ejemplo de borrar texto de la pizarra

En la siguiente sección se definen los nodos de estilo y diseño.

3.1.4. Nodos de estilo y diseño

Los nodos de estilo y diseño controlan la presentación visual. Permiten adaptar la estética al tono de la historia y mejorar la legibilidad. Los nodos de estilo y diseño son:

- Cambiar_imagen_de_fondo: Modifica la imagen de fondo de la escena. Útil para transiciones entre ambientes. Recibe la ubicación de la imagen de fondo y el id de la escena. Por ejemplo:

```

1  (change_bg_image :id "intro"
2           :src "/tmp/images/bg.png")
```

Ejemplo de código 3.14: Ejemplo de cambiar imagen de fondo

- Cambiar color de fondo: Modifica el color de fondo de la escena. Recibe el nuevo color y el id de la escena.
- Cambiar tipos de letra: Ajusta la fuente del texto. Puede usarse, por ejemplo, para resaltar citas. Recibe la nueva fuente y el texto a modificar.
- Cambiar colores de letra: Personaliza el color del texto para enfatizar o diferenciar contenido. Recibe el color de letra y el texto a modificar.

El DSL CLIF está enfocada en redactar historias interactivas en el área de matemáticas y computación. Es por ello que se necesitará introducir contenido matemático y de programación. Para ello se crearon los nodos técnicos que se abordan en la siguiente sección.

3.1.5. Nodos técnicos

Los nodos técnicos permiten insertar elementos de matemática o de código. Para esto existen los nodos:

- Insertar código: Permite incluir bloques de código con sintaxis resaltada. Es ideal para usarlo en tutoriales o ejemplos técnicos. Recibe el lenguaje en el que se introduce código y el texto del código. Por ejemplo:

```
1 (code_node :lang python
2   :code (print("hello world")))
```

Ejemplo de código 3.15: Ejemplo de código en Python

- Insertar matemática: Posibilita agregar contenido con fórmulas matemáticas al texto usando el formato LaTeX. Reciben una lista de textos matemáticos. Estos nodos pueden ser de dos tipos:

- Inline: Fórmulas matemáticas integradas en el texto. Por ejemplo:

```
1 (math_inline
2   (a^n = b^n + c^n, n>2))
```

Ejemplo de código 3.16: Ejemplo de matemática inline

- Centrada: Fórmulas destacadas en su propio bloque. Por ejemplo:

```
1 (math_center
2   (a^2 = b^2 + c^2)
3   (a^3 = b^3 + c^3))
```

```
4   (a^4 = b^4 + c^4))
```

Ejemplo de código 3.17: Ejemplo de matemática centrada

En la siguiente sección se describen los nodos de programación interactiva.

3.1.6. Nodos de programación interactiva

Los nodos de programación interactiva permiten realizar acciones con variables. Para esto tenemos tres tipos de nodos:

- Declarar_variables: Crea y almacena valores para su uso posterior. Recibe una lista de variables con un valor inicial. Por ejemplo:

```
1 (declare_vars
2   (nombre "Juan")
3   (edad 25)
4   (puntaje 100.5)
5   (activo true))
```

Ejemplo de código 3.18: Ejemplo de declaración de variables

- Modificar_variables: Cambia el valor de variables existentes. Recibe una lista de variables con el nuevo valor de cada una. Por ejemplo:

```
1 (modify_vars
2   (edad (+ edad 1))
3   (puntaje (* puntaje 1.1))
4   (activo (not activo)))
```

Ejemplo de código 3.19: Ejemplo de modificación de variables

- Comparar_variables: Permite tomar decisiones basadas en el estado de las variables, habilitando ramificaciones en la historia. Por ejemplo:

```
1 (if (>= puntaje 15)
2   (link "Abrir libro")
3   else
4   (link "Dejar libro"))
```

Ejemplo de código 3.20: Ejemplo de comparación de variables

En la siguiente sección se describe el nodo escena.

3.1.7. Nodo escena

Los nodos de escena representan una unidad narrativa completa. Contienen todos los elementos anteriores organizados en una secuencia lógica. Las escenas pueden enlazarse entre sí mediante enlaces para crear flujos interactivos. Por ejemplo:

```

1  (scene "Introduccion"
2    (text "Bienvenido a nuestra historia interactiva")
3    (ord-list
4      (item (link adventure1 "Comenzar" (bold "aventura")))
5      (item (link tutorial "Ver tutorial"))))

```

Ejemplo de código 3.21: Ejemplo de escena

Para obtener la escena en un formato de salida dado, es necesario definir, para cada uno de estos nodos cómo se genera el código correspondiente en el formato de salida deseado. En la siguiente sección se definen cuáles son dichos formatos de salida.

3.1.8. Formatos de salida

Para cada formato de salida es necesario tener una manera de representarlos. En CLIF, cada formato de salida se representa mediante una clase en Lisp. Por ejemplo, el formato Undum está representado por la clase `undum-language`, mientras que LATEX está representado por la clase `latex-language`.

Para generar el código correspondiente a un nodo del AST en un formato específico, se especializa la función genérica `generate-code` en dos argumentos: el nodo del AST y la clase que representa el formato de salida.

Por ejemplo, para obtener cómo se describe el nodo de enlace en el formato de salida representado por la clase `undum-language`, se define el método:

```

1  (defmethod generate-code ((node link) (lang undum-
   language))
2  ;; código para link en html)

```

Ejemplo de código 3.22: Nodo de enlace en Undum

En caso de que un formato de salida no soporte un tipo de nodo determinado, por ejemplo, el nodo `play-sound` en PDF, el método que genere el código para ese formato de salida es responsable de manejar esta situación. Para facilitar este proceso, todos los nodos multimedia de CLIF incluyen un atributo `alt-text` de texto alternativo que el autor puede utilizar para describir el contenido en formatos que no lo soporten directamente.

Actualmente, CLIF soporta los siguientes formatos de salida:

- **Undum:** Los nodos se traducen a código HTML5, CSS3 y JavaScript para crear historias interactivas en navegadores web.
- **LATEX:** Los nodos se transforman en comandos de LATEX para generar documentos PDF.

Para extender CLIF con nuevos formatos de salida, basta con definir una nueva clase que represente el formato e implementar los métodos correspondientes de `generate-code` para cada tipo de nodo del AST que se desee soportar. En el siguiente capítulo se describe el proceso de implementación de código.

Capítulo 4

Detalles de Implementación

La implementación de CLIF se estructura en dos componentes principales: el núcleo del sistema y la generación de código. En la siguiente sección se describe el núcleo del sistema.

4.1. Núcleo del sistema

Las clases del AST se definen usando el macro `defnode` de la biblioteca MGA. Este macro se encarga de crear la clase y un constructor adecuado para cada nodo.

La mayoría de nodos heredan de la clase abstracta `has-contents`, que provee la estructura fundamental para elementos con contenido. Por ejemplo, los nodos de texto, listas ordenadas e itálicas se definen de la siguiente forma:

```
1 (defnode text (has-contents)
2 ())
3 :documentation "Nodo para texto plano")
4
5 (defnode ord_list (has-contents)
6 ())
7 :documentation "Nodo para listas ordenadas")
8
9
10 (defnode italic (has-contents)
11 ())
12 :documentation "Nodo para escribir texto en italica.")
```

Ejemplo de código 4.1: Ejemplos de nodos que heredan de `has-contents`

El nodo de enlace entre situaciones es otro ejemplo de nodo que hereda de `has-contents` y que, además, tiene atributos específicos:

```

1  (defnode link (has-contents)
2    ((to :type symb)
3     (text :type string))
4     :documentation "Nodo de enlace")

```

Ejemplo de código 4.2: Ejemplo del nodo `link`

En la siguiente sección se profundiza en el proceso de generación de código.

4.2. Generación de código

El proceso de generación de código en CLIF transforma el AST en archivos listos para su ejecución. Para el formato Undum, por ejemplo, esto implica crear dos tipos de archivos: un documento HTML base y un archivo JavaScript que contiene toda la lógica de las situaciones.

El sistema de generación de código en CLIF se fundamenta en una implementación del patrón *visitor* [78] mediante el polimorfismo de múltiples argumentos provisto por CLOS [73]. Este diseño permite separar la estructura del árbol de sintaxis abstracta de las operaciones realizadas sobre él. Cada nodo del AST define métodos especializados que toman como argumentos tanto el tipo de nodo como el lenguaje de destino, permitiendo que CLOS seleccione dinámicamente la implementación adecuada en tiempo de ejecución.

La clave del mecanismo de generación de código reside en el mecanismo de sustitución de plantillas. Por ejemplo, un nodo *ítalicas*, que recibe una lista de elementos, en LaTeX se escribiría como:

```
\textit{elementos-del-nodo}
```

De manera similar, una lista ordenada en LaTeX se transforma en:

```
\begin{enumerate}
  elementos-del-nodo
\end{enumerate}
```

Cada uno de los elementos del nodo debería saber cómo generar su código. Este patrón de sustitución de plantillas está contemplado en MGA con el macro `gcode` que permite hacer esta generación de la siguiente forma:

```

1  (gcode italic latex-language
2   "\\textit{~a}" elements)

```

Donde `elements` es el nombre del *slot* del nodo `italic`. Para el caso de las listas ordenadas:

```
1      (gcode ord_list latex-language
2      " \\"begin{enumerate}~%~"
3      ~{~a~^~%~}~%~"
4      \"end{enumerate}" elements)
```

Para incorporar un nuevo formato de salida, como soporte para Twine, se sigue un proceso sistemático. Primero se define una clase que represente el nuevo formato, como `twine-language`. Luego se implementan métodos especializados para cada nodo del AST que traduzcan su contenido al formato esperado por la plataforma destino. Por ejemplo, un nodo `escena` podría traducirse a un *pasaje* en Twine [79]. La escalabilidad del diseño queda demostrada al poder añadir estos comportamientos sin modificar el código existente de los nodos ni de otros formatos de salida.

La arquitectura resultante ofrece ventajas en mantenibilidad y extensibilidad, pues los formatos de salida pueden evolucionar en paralelo, mientras que el núcleo del AST permanece estable. Esto facilita escenarios como la generación a múltiples formatos a partir de una misma historia, o la incorporación futura de otros motores de ficción interactiva.

Capítulo 5

Ejemplo de una historia

Conclusiones

En este trabajo se presentó CLIF: un lenguaje de dominio específico para generar ficción interactiva en múltiples formatos y que fue desarrollado en Common Lisp.

Las historias de ficción interactiva generadas con CLIF pueden obtenerse en varios formatos como Undum, L^AT_EXo bots de Telegram. CLIF incluye elementos para facilitar la creación de historias interactivas enfocadas en las áreas de matemática y computación. Para ello integra una pizarra virtual y herramientas de inserción de código. El uso de las historias desarrolladas con CLIF debería fomentar las actividades educativas de matemática y computación debido a su accesibilidad e interactividad, lo cual resultaría atractivo y novedoso para los usuarios.

Como CLIF está desarrollado en Common Lisp, es posible obtener directamente el AST para cada uno de los formatos de salida a partir de los constructores de los nodos del árbol. Esto simplifica el proceso de compilación, eliminando la necesidad de implementar un analizador lexicográfico ni sintáctico.

Una de las ventajas de CLIF es la extensibilidad que se traduce en la posibilidad de añadir nuevas abstracciones al lenguaje sin implementar un nuevo analizador sintáctico. Esto se puede lograr agregando nuevos nodos al AST utilizando el macro `defnode` para definir nuevas clases de nodos. Además, como CLIF está diseñado para ser multiplataforma, permite generar código para diferentes lenguajes de salida a través de la función `generate-code`. Cada uno de los nuevos formatos de salida se define como una clase e implementa métodos especializados para cada nodo del AST.

Recomendaciones

Como trabajo futuro se recomienda añadir soporte para formatos como Twine y ampliar las capacidades de generación para Undum con nuevas etiquetas HTML. Además, se podrían agregar funcionalidades que permitan crear historias simultáneas en Telegram donde varios usuarios influyen a la vez en el transcurso de una narrativa central y común para todos.

También se recomienda convertir conferencias y contenidos de clases que actualmente se encuentran en formatos estáticos a una versión interactiva y estudiar cómo incide en el proceso de enseñanza aprendizaje.

Referencias

- [1] Nick Montfort. *Twisty Little Passages: An Approach to Interactive Fiction*. MIT Press, 2003 (vid. págs. 1, 5, 6).
- [2] *Viviendo en un mundo cada vez más digital*. <https://www.signeblock.com/viviendo-en-un-mundo-cada-vez-mas-digital-n-8505-es>. SigneBlock, 2021 (vid. pág. 1).
- [3] Redactores de Teacher Academy Blog. «Interactive Fiction in Education: A Powerful Tool for Engagement». En: *Teacher Academy Blog* (2021). ISSN: 2603-5525 (vid. págs. 1, 12).
- [4] *Choice of Games LLC*. <https://www.choiceofgames.com>. Choice of Games LLC, 2025 (vid. págs. 1, 7).
- [5] *Hosted Games*. <https://hostedgames.org>. Hosted Games, 2025 (vid. pág. 1).
- [6] *Delight Games LLC*. <https://www.delightgamesllc.com>. Delight Games LLC, 2025 (vid. pág. 1).
- [7] Larry Ferlazzo. *The best places to read and write choose your own adventure "stories*. <https://larryferlazzo.edublogs.org/2009/05/03/the-best-places-to-read-and-write-choose-your-own-adventure-stories/>. Mayo de 2009 (vid. pág. 1).
- [8] Marc Schäfer. «Manipulatives as Mediums for Visualisation Processes in the Teaching of Mathematics». En: *Mathematics Teaching and Professional Learning in sub-Saharan Africa*. Cham: Springer International Publishing, 2021, págs. 5-21. URL: https://doi.org/10.1007/978-3-030-82723-6_1 (vid. pág. 1).
- [9] Graham Nelson y the Inform community. *Inform 7: Design System for Interactive Fiction*. <https://ganelson.github.io/inform-website/>. 2022. URL: <https://ganelson.github.io/inform-website/> (visitado 15-11-2023) (vid. págs. 1, 6).
- [10] M. J. Roberts. *Text Adventure Development System*. <https://www.tads.org/>. Nov. de 2013 (vid. págs. 1, 6).

- [11] Dan Cox y Taylor Howard. *Unofficial Ink Cookbook: Chapter 1*. <https://videlais.github.io/Unofficial-Ink-Cookbook/Chapter1/>. Unofficial Ink Cookbook, 2020 (vid. págs. 1, 6, 7).
- [12] Chris Klimas y contributors. *Twine: An Open-source Tool for Telling Interactive, Nonlinear Stories*. <https://twinery.org>. Ver. 2.4.1. 2024 (vid. págs. 1, 7).
- [13] Joseph Humphrey y Jon Ingold. *Ink: Narrative Scripting Language*. <https://www.inklestudios.com/ink/>. Scripting language for interactive narrative in games. Inkle Ltd, 2023. (Visitado 20-11-2023) (vid. pág. 1).
- [14] Tom Rothamel. *Ren'Py Visual Novel Engine*. <https://www.renpy.org>. Ren'Py, 2025 (vid. pág. 1).
- [15] *Programming Library Guide: What Is a Programming Library?* <https://careerfoundry.com/en/blog/web-development/programming-library-guide/>. Explains the concept of programming libraries, their uses, and differences from frameworks. CareerFoundry, 2024 (vid. pág. 1).
- [16] Volund Griatch. *Evennia MUD Server Framework*. <https://github.com/evennia/evennia>. 2025 (vid. pág. 1).
- [17] Ian Millington. *Undum Documentation*. <https://idmillington.github.io/undum/>. 2017 (vid. págs. 1, 7, 9, 11).
- [18] *LaTeX – A Document Preparation System*. <https://www.latex-project.org>. LaTeX Project, 2025 (vid. págs. 2, 14).
- [19] J. Smith y H. Lee. «Developing Interactive Fiction as Telegram Bots». En: *Interactive Storytelling Conference* (2020), págs. 45-58 (vid. pág. 2).
- [20] *Domain-Specific Languages (DSLs)*. <https://www.jetbrains.com/mps/concepts/domain-specific-languages>. JetBrains, 2025 (vid. págs. 2, 14, 15).
- [21] Seymour Chatman. *Story and Discourse: Narrative Structure in Fiction and Film*. Ithaca, NY: Cornell University Press, 1978. ISBN: 978-0801491862 (vid. pág. 3).
- [22] Mieke Bal. *Narratology: Introduction to the Theory of Narrative*. Toronto: University of Toronto Press, 1985. ISBN: 978-0802063679 (vid. pág. 3).
- [23] Gérard Genette. *Narrative Discourse: An Essay in Method*. Trad. por Jane E. Lewin. Ithaca, NY: Cornell University Press, 1980. ISBN: 978-0801410993 (vid. pág. 3).
- [24] Marie-Laure Ryan. *Avatars of Story*. Minneapolis: University of Minnesota Press, 2006. ISBN: 978-0816643412 (vid. pág. 3).

- [25] Janet H. Murray. *Hamlet on the Holodeck: The Future of Narrative in Cyberspace*. New York: Free Press, 1997. ISBN: 978-0684827237 (vid. pág. 4).
- [26] Jeffrey Heer et al. «Graphical Histories for Visualization: Supporting Analysis, Communication, and Evaluation». En: *IEEE Transactions on Visualization and Computer Graphics* 14.6 (2008), págs. 1189-1196. DOI: 10.1109/TVCG.2008.153. URL: <http://vis.stanford.edu/files/2008-GraphicalHistories-InfoVis.pdf> (vid. pág. 5).
- [27] Shixia Liu et al. «A Survey on Interaction Histories in Visual Analytics». En: *IEEE Transactions on Visualization and Computer Graphics* 26.1 (2020), págs. 55-75. DOI: 10.1109/TVCG.2019.2934289. URL: <https://par.nsf.gov/servlets/purl/10180883> (vid. pág. 5).
- [28] *Interactive Fiction*. <https://www.computerhope.com/jargon/i/interactive-fiction.htm>. Computer Hope, 2025 (vid. pág. 5).
- [29] Lauren Magaziner. *¡Resuelve el Misterio!* Ciudad, País: Editorial Ficción Interactiva, 2023 (vid. pág. 5).
- [30] Arthur Conan Doyle (adaptación interactiva). *Las aventuras interactivas de Sherlock Holmes: Escándalo en Bohemia*. Ciudad, País: Editorial Digital Interactiva, 2021 (vid. pág. 5).
- [31] Metia Interactive. *Guardian Maia*. Ficción interactiva en línea. 2022 (vid. pág. 5).
- [32] Warren Robinett. *Adventure*. 1980 (vid. pág. 5).
- [33] Tim Anderson et al. *Zork*. 1977 (vid. pág. 5).
- [34] *The Interactive Fiction Community Forum*. <https://intfiction.org>. intfiction.org, 2025 (vid. pág. 5).
- [35] Tyler Burge. «Interactive Fiction as Literature». En: *Revista BYTE* 12.5 (1979), págs. 135-142 (vid. pág. 5).
- [36] Clara Fernández-Vara. *Parser-Based Interactive Fiction: A Primer*. <https://clarafv.com/2019/05/13/parser-based-interactive-fiction-a-primer/>. 2019 (vid. págs. 5, 6).
- [37] Adam Cadre. *Fotopia*. 1998. URL: <https://ifdb.org/viewgame?id=ju778uv5xaswnlpl> (vid. pág. 6).
- [38] Shelley Jackson. *Patchwork Girl; or, a Modern Monster by Mary/Shelley and Herself*. Storyspace diskette. 1995 (vid. pág. 6).
- [39] Michael Joyce. *afternoon, a story*. Storyspace diskette. 1990 (vid. pág. 6).
- [40] Espen J. Aarseth. *Cybertext: Perspectives on Ergodic Literature*. Baltimore, MD: Johns Hopkins University Press, 1997. ISBN: 9780801855795. URL: <https://dl.acm.org/doi/abs/10.5555/265941> (vid. pág. 6).

- [41] Norbert Bachleitner. «Formas de literatura digital». En: *Tema y Variaciones de Literatura* 45 (2019), págs. 63-75. URL: https://publikationen.ub.uni-frankfurt.de/opus4/frontdoor/deliver/index/docId/47248/file/Bachleitner_Formas_literatura_digital.pdf (vid. pág. 6).
- [42] *CYOA Definition: Choose Your Own Adventure Examples.* <https://writing-games.com/text-game-terms/cyoa-definition-choose-your-own-adventure-examples/>. Writing Games, 2025 (vid. pág. 7).
- [43] Michael J. Tresca. *The Evolution of Fantasy Role-Playing Games*. McFarland, 2011, pág. 100 (vid. pág. 7).
- [44] Edward Packard. *The Cave of Time*. Bantam Books, 1979 (vid. pág. 7).
- [45] Alex Mitchell y Kevin McGee. «Creating interactive fiction with Inform 7». En: *Proceedings of the 3rd International Conference on Interactive Digital Storytelling*. 2010, págs. 176-181 (vid. pág. 7).
- [46] Inkle Studios. *Ink and Inkle: Narrative Games*. 2013. URL: <https://www.inklestudios.com/ink/> (vid. pág. 7).
- [47] Julio Verne. *La vuelta al mundo en ochenta días*. francés. París: Pierre-Jules Hetzel, 1873 (vid. pág. 7).
- [48] Inkle Studios. *80 Days*. 2014 (vid. pág. 7).
- [49] Choice of Games. *Choice of Robots*. 2014 (vid. pág. 7).
- [50] *Squiffy*. <https://textadventures.co.uk/squiffy>. TextAdventures.co.uk, 2024 (vid. pág. 7).
- [51] Choice of Games. *ChoiceScript Language Reference*. 2010. URL: <https://www.choiceofgames.com/make-your-own-games/choicescript-intro/> (vid. pág. 7).
- [52] Kitty Curran y Larissa Zageris. *My Lady's Choosing: An Interactive Romance Novel*. Philadelphia, PA: Quirk Books, 2018. ISBN: 9781683690139 (vid. pág. 8).
- [53] Ryan North. *To Be or Not To Be: A Chooseable-Path Adventure*. New York, NY: Broadway Books, 2013. ISBN: 9780307946942 (vid. pág. 8).
- [54] R. A. Montgomery. *Space and Beyond (Choose Your Own Adventure, #4)*. Chooseco, 1983 (vid. pág. 8).
- [55] Joe Dever. *Lone Wolf*. New York, NY: Berkley Books, 1984. ISBN: 9780425076185 (vid. pág. 8).
- [56] Tom Idling. *Undum: Interactive Fiction Tool*. 2010 (vid. pág. 9).
- [57] *LingoStories*. <https://lingostories.org>. 2025 (vid. pág. 12).

- [58] Chris Matthews. *The Dregg Disaster: An Algebra 1 Workbook (Choose Your Own Adventure)*. Chooseco, 2022. ISBN: 9781937133931 (vid. pág. 12).
- [59] *Code Adventures: Coding Puzzles For Kids*. <https://play.google.com/store/apps/details?id=com.cyborc.codeadventures>. 2025 (vid. pág. 12).
- [60] Paul Broyles. *Teaching and Learning with Interactive Fiction*. <https://bdesilets.com/if/TLIF.pdf>. 2015 (vid. pág. 12).
- [61] Deborah Kozdras, Denise Haunstetter y James King. «Interactive Fiction: ‘New Literacy’ Learning Opportunities for Children». En: *E-learning* 3 (dic. de 2006). DOI: 10.2304/elea.2006.3.4.519 (vid. pág. 12).
- [62] Joanna Marquez et al. «Gender Differences in Response to Educational IF». En: *Journal of Computing in Higher Education* 35 (2023), págs. 127-149 (vid. pág. 12).
- [63] GeeksforGeeks. *Parse Tree and Syntax Tree*. <https://www.geeksforgeeks.org/parse-tree-and-syntax-tree/>. Consultado: abril 2025 (vid. pág. 13).
- [64] Alfred V. Aho et al. *Compilers: Principles, Techniques, and Tools*. Pearson Education, 2006 (vid. págs. 13, 14).
- [65] *Welcome to SQL.org*. <https://www.sql.org>. SQL.org, 2025 (vid. pág. 14).
- [66] C. J. Date. *An Introduction to Database Systems*. 8th. Addison-Wesley, 2003 (vid. pág. 14).
- [67] *HTML: HyperText Markup Language*. <https://developer.mozilla.org/en-US/docs/Web/HTML>. Mozilla Developer Network (MDN), 2025 (vid. pág. 14).
- [68] M. Berschady. *HTML5: The Definitive Guide*. O'Reilly Media, 2020 (vid. pág. 14).
- [69] Leslie Lamport. *LaTeX: A Document Preparation System*. 2nd. Addison-Wesley, 1994 (vid. pág. 14).
- [70] *MATLAB - MathWorks*. <https://www.mathworks.com/products/matlab.html>. MathWorks, 2025 (vid. pág. 14).
- [71] Desmond J. Higham y Nicholas J. Higham. *MATLAB Guide*. 2nd. SIAM, 2005 (vid. pág. 14).
- [72] Sofía Liaqat. *Lisp Programming Language Guide: History, Origin, and More*. <https://sofialiaqat215.medium.com/lisp-programming-language-guide-history-origin-and-more-8381509b4b5d>. 2024 (vid. pág. 15).
- [73] Paul Graham. *On Lisp: Advanced Techniques for Common Lisp*. Englewood Cliffs, NJ: Prentice Hall, 1994. ISBN: 0-13-030552-9 (vid. págs. 15, 16, 18, 30).
- [74] Peter Seibel. *Practical Common Lisp*. Berkeley, CA: Apress, 2005. ISBN: 978-1590592397 (vid. pág. 15).

- [75] *Common-Lisp.net*. <https://common-lisp.net>. Common Lisp Foundation, 2025 (vid. pág. 16).
- [76] Claudia Porto Copetillo. «LMML: Lenguaje para la modelación matemática en Lisp». Trabajo de Diploma. La Habana, Cuba: Universidad de La Habana, 2019 (vid. pág. 19).
- [77] Yasmany Arcia Corcho. «Automatizar la creación de Generadores Multilenguajes». Trabajo de Diploma. La Habana, Cuba: Facultad de Matemática y Computación, Universidad de La Habana, 2017 (vid. pág. 19).
- [78] Alexander Shvets. *Sumérgete en los patrones de diseño: Dominando los patrones esenciales para un diseño de software efectivo*. Bookey, 2018. ISBN: 978-1988563146. URL: <https://refactoring.guru/es/design-patterns/book> (vid. pág. 30).
- [79] Chris Klimas. *Twine: An Open-Source Tool for Telling Interactive, Nonlinear Stories*. Self-published, 2009 (vid. pág. 31).