Escuela Superior
de Ingeniería y Tecnología
Universidad de La Laguna

Trabajo de Fin de Grado

Lenguaje de Dominio Específico para la evaluación de actividades de Pensamiento Computacional

Domain Specific Language for the evaluation of Computational Thinking activities

Sergio Reyes de León

La Laguna, 13 de julio de 2023

D. **Coromoto León Hernández**, con N.I.F. 78.605.216-W profesora Catedrático de Universidad del área de Lenguajes y Sistemas Informáticos, adscrita al Departamento de Ingeniería Informática y de Sistemas de la Universidad de La Laguna, como tutora

D. **Rafael Herrero Álvarez**, con N.I.F. 54.063.043-W Personal Investigador en Formación del área de Lenguajes y Sistemas Informáticos, adscrito al Departamento de de Ingeniería Informática y de Sistemas de la Universidad de La Laguna, como cotutor

I N F O R M A (N)

Que la presente memoria titulada:

"*Lenguaje de Dominio Específico para la evaluación de actividades de Pensamiento Computacional*"

ha sido realizada bajo su dirección por D. **Sergio Reyes de León**, con N.I.F. 78.716.025-C.

Y para que así conste, en cumplimiento de la legislación vigente y a los efectos oportunos firman la presente en La Laguna a 13 de julio de 2023

Agradecimientos

A mis tutores, Coromoto y Rafael, por su infinita paciencia y a todos los que me han apoyado.

Licencia

© Esta obra está bajo una licencia de Creative Commons Reconocimiento 4.0 Internacional.

Resumen

El presente Trabajo de Fin de Grado aborda el desarrollo de un lenguaje de dominio específico (DSL) para la evaluación de actividades de pensamiento computacional en la plataforma code.org. El enfoque principal se centra en implementar este DSL utilizando el lenguaje de programación C#.

El objetivo de este Trabajo de Fin de Grado es diseñar y desarrollar un lenguaje de dominio específico que permita evaluar de manera efectiva las competencias y habilidades relacionadas con el pensamiento computacional en la plataforma code.org.

Además, se ha desarrollado un extensión para el navegador Chrome que extraerá los datos de las actividades de la plataforma code.org para que posteriormente puedan ser evaluados mediante el DSL.

Se espera que el desarrollo de este DSL facilite la evaluación y mejora de las habilidades de pensamiento computacional, proporcionando a los educadores una herramienta efectiva y flexible para el seguimiento del progreso de los estudiantes en esta área fundamental

Palabras clave: DSL, lenguaje de dominio específico, pensamiento computacional

Abstract

This Final Degree Project deals with the development of a domain-specific language (DSL) for the evaluation of computational thinking activities in the code.org platform. The main focus is on implementing this DSL using the C# programming language.

The objective of this Final Degree Project is to design and develop a domain-specific language to effectively assess competencies and skills related to computational thinking on the code.org platform.

In addition, an extension has been developed for the Chrome browser that will extract data from the activities of the code.org platform so that they can be subsequently evaluated using the DSL.

The development of this DSL is expected to facilitate the assessment and improvement of computational thinking skills, providing educators with an effective and flexible tool for tracking student progress in this fundamental area.

Keywords: DSL, domain-specific language, computational thinking

Índice general

Índice de Figuras

Índice de Tablas

Capítulo 1

Introducción

Un lenguaje de dominio específico es una notación diseñada para un determinado fin. Frente a los lenguajes de propósito general poseen mayor expresividad y facilidad de uso. Se pueden clasificar según la construcción del lenguaje, formato del lenguaje o desde el punto de vista del dominio del problema.

Según la construcción del lenguaje:

- Internos: desarrollados como una API dentro de un lenguaje de propósito general
- Externos: Poseen su propia sintaxis y necesitan un parser para poder ser procesados.

Según el formato del lenguaje:

- Textuales: Están formados por un conjunto de sentencias.Se generan a partir de una gramática y posteriormente se utiliza un parser para interpretarlo.
- Gráficos: Estos lenguajes no utilizan texto, en su lugar hacen uso de conectores y figuras simples.

Desde el punto de vista del dominio del problema:

- Horizontales: son aquellos en los que el cliente que utilizará el lenguaje no pertenece a ningún dominio específico.
- Verticales: el cliente que utilizará el lenguaje pertenece al mismo dominio que el lenguaje en sí.

1.1. Motivación

En la actualidad, el pensamiento computacional se ha convertido en una habilidad fundamental en el ámbito de la informática y la tecnología. Esta capacidad cognitiva implica la resolución de problemas, el razonamiento lógico, la abstracción y la descomposición de tareas, habilidades que son cada vez más requeridas en diversas disciplinas y sectores profesionales.

El uso de un Lenguaje de Dominio Específico para la evaluación del pensamiento computacional puede contribuir a la estandarización y comparabilidad de los resultados obtenidos. Esto permitirá a los investigadores y educadores analizar y comparar de manera más precisa el progreso de los estudiantes y evaluar la eficacia de diferentes enfoques pedagógicos en la enseñanza del pensamiento computacional.

1.2. Objetivo

El objetivo de este TFG es crear un DSL interno, contextual y vertical, a partir del lenguaje de propósito general C# para la evaluación de actividades relacionadas con la programación y el Pensamiento Computacional.

1.3. Metodología

Teniendo en cuenta los objetivos y antecedentes descritos anteriormente, se proponen las siguientes tareas para la ejecución del proyecto:

Tarea 1: Revisión Bibliográfica (antecedentes)
Recopilar información sobre lenguajes de dominio específico (DSL) para la gestión de actividades de Pensamiento Computacional, además de sobre los modelos de gestión existentes que tratan su forma de funcionamiento, así como estudios sobre el grado de satisfacción con los mismos.

Tarea 2: Diseño de un prototipo de aplicación
Diseñar el lenguaje de dominio específico para la evaluación de actividades de Pensamiento Computacional. Para ello se crearán los respectivos diagramas UML.

En esta primera aproximación se hará uso de las actividades desarrolladas por CODE.org.

Tarea 3: Codificación del prototipo diseñado
Implementar el lenguaje de dominio específico. Se utilizará el lenguaje C#.

Tarea 4: Verificación y pruebas
Realizar las pruebas pertinentes para comprobar el correcto funcionamiento de cada uno de los elementos de la aplicación.

Tarea 5: Documentación
Redacción de la memoria del Trabajo Fin de Grado, realización del vídeo de presentación del mismo, así como el resumen para su presentación oral.

1.4. Organización de la memoria

La memoria de este Trabajo de Fin de Grado está estructurada en diferentes secciones que abarcan los aspectos fundamentales de la investigación y desarrollo realizado.

En primer lugar, se presenta una introducción que contextualiza el tema y justifica la importancia del proyecto. A continuación, se incluye una revisión de la bibliografía, donde se examinan los trabajos previos relacionados con la evaluación del pensamiento computacional y el uso de lenguajes de dominio específico. Luego, se detalla la metodología empleada, describiendo el proceso de diseño e implementación del DSL y su integración en la plataforma code.org. En la sección de conclusiones, se resumen los resultados, se discuten las implicaciones y se plantean líneas futuras de investigación.

Finalmente, se incluye una bibliografía que recopila las fuentes consultadas durante el desarrollo del proyecto.

Capítulo 2

Revisión bibliográfica

La revisión bibliográfica realizada en este proyecto aporta una base sólida de conocimiento previo relacionado con la evaluación del pensamiento computacional y el uso de lenguajes de dominio específico en este ámbito. A través de esta revisión, se ha obtenido una comprensión profunda de los enfoques existentes, las herramientas utilizadas y las investigaciones relevantes en el campo de estudio. Esto ha permitido identificar las necesidades y los desafíos en la evaluación del pensamiento computacional, así como identificar las mejores prácticas y los aspectos clave a considerar en el diseño y desarrollo del DSL propuesto.

Se buscó información sobre las aplicaciones informáticas en el mercado desarrolladas para comprender y medir el proceso de gestión del aprendizaje de pensamiento computacional. Para llevar a cabo este proceso se utilizaron las siguientes herramientas para realizar búsquedas de documentación: PuntoQ, Dialnet y Google Académico.

Además, se siguió un procedimiento de búsqueda sistemática denominado PRISMA para las revisiones sistemáticas de la literatura en el contexto de la Ingeniería de Software.

De los resultados de las búsquedas se han seleccionado estas referencias:

[4] En la actualidad existen numerosos problemas, que si bien pueden ser solucionados con herramientas de propósito general, es más apropiado abordarlos con aplicaciones específicas para ese dominio. En este contexto se encuentran los Lenguajes Específicos del Dominio. Un Lenguaje Específico del Dominio es un conjunto reducido de construcciones y operaciones que brindan una mayor expresividad y optimización para un dominio particular.

La construcción de Lenguajes Específicos del Dominio (DSL) no es una tarea trivial ya que implica tres fases muy importantes como lo son: Análi-

sis, Diseño e Implementación. Si bien estas fases coinciden con un proceso de desarrollo de software tradicional tienen otras particularidades relacionadas con los DSL, es que introducen nuevos desafíos para la investigación y desarrollo de los mismos.

[7] Este artículo describe la creación de un modelado específico de dominio para la construcción de módulos en sistemas de gestión del aprendizaje (LMS). Se parte de un metamodelo para la construcción de un lenguaje específico de dominio (DSL) que con ingeniería dirigida por modelos (MDE) y tras aplicar las debidas transformaciones se obtiene de un modelo independiente de la plataforma, el despliegue de este modelo se realizó en varias plataformas LMS.

[8] El Pensamiento Computacional es una metodología basada en la implementación de los conceptos básicos de las ciencias de la computación para resolver problemas cotidianos, diseñar sistemas domésticos y realizar tareas rutinarias. Esta nueva forma de abordar los problemas nos permite resolver con eficacia y éxito problemas que de otra forma no son tratables por una persona.

Una primera y errónea idea que se puede tener del Pensamiento Computacional es creer que es una materia exclusiva para personas del ámbito de la ingeniería informática y computación. Existe un interés y esfuerzo creciente en incorporar el Pensamiento Computacional a través de proyectos, juegos, entornos de programación, etc. en el currículum de escuelas y universidades.

[1] Los DSL no son más que un mecanismo de abstracción. El interés actual radica en la creciente comprensión de que algunas abstracciones se resisten a una fácil representación en lenguajes modernos como C#.

Durante los últimos 30 años los desarrolladores han utilizado objetos como el principal mecanismo de abstracción, los cuales funcionan muy bien porque resulta que gran parte del mundo es jerárquico.

LINQ es un buen ejemplo de un DSL interno porque su sintaxis es una sintaxis legal de C#, pero una sintaxis extendida (específica del dominio). Un DSL externo describe un lenguaje creado con un lexer y un analizador, donde crea su propia sintaxis. SQL es un buen ejemplo de un DSL externo: para el cual fue necesario escribir una gramática y una forma de interpretarla en algún otro código ejecutable.

Capítulo 3

Herramientas de Desarrollo

A continuación se detallan las herramientas que se han utilizado para la elaboración de este trabajo y el motivo de su elección.

- **Draw.io:** Se ha utilizado la herramienta online gratuita Draw.io para diseñar el diagrama de clases del DSL. Draw.io ofrece una interfaz intuitiva y diversas opciones de personalización, lo que facilita la creación de diagramas claros y comprensibles para ilustrar conceptos y estructuras complejas. https://draw.io
- **C#:** El lenguaje elegido para desarrollar el DSL ha sido C# en su versión 11. Es un lenguaje compilado, tipado y además multiplataforma, robusto y orientado a objetos que permite desarrollar aplicaciones eficientes y escalables. Además, cuenta con una gran comunidad de desarrolladores y una amplia gama de bibliotecas y herramientas disponibles para facilitar el desarrollo. https://learn.microsoft.com/es-es/dotnet/csharp
- **Rider:** Es un entorno de desarrollo integrado (IDE) multiplataforma para C# y .NET. Aunque es un programa de pago se ha utilizado una licencia para estudiantes. Se eligió por su rica funcionalidad y facilidad de uso. Ofrece características avanzadas de depuración, resaltado de sintaxis, completado de código y refactorización, lo que agiliza el desarrollo del DSL y facilita la detección y corrección de errores. https://www.jetbrains.com/es-es/rider
- **xUnit.net:** Se optó por xUnit.net como el marco de pruebas unitarias para el proyecto. xUnit.net es una biblioteca de pruebas unitarias ampliamente utilizada en el ecosistema .NET. El motivo de su uso es que proporciona una sintaxis clara y concisa para escribir pruebas y ofrece una amplia cobertura de funcionalidades para facilitar la ejecución y el análisis de las pruebas unitarias. https://xunit.net

- **ReportGenerator:** Se utilizó como una herramienta para generar informes detallados sobre la cobertura de código en el proyecto. Permite visualizar de manera clara y concisa la cobertura de las pruebas unitarias, lo que facilita la identificación de áreas de código que no están adecuadamente cubiertas por las pruebas y orienta la toma de decisiones para mejorar la calidad del DSL. https://reportgenerator.io

- **Javascript:** Es un lenguaje de programación interpretado y débilmente tipado. Se ha hecho uso de este lenguaje, para el desarrollo de la extensión de Chrome que extrae la información de code.org, consultando la documentación de google para la implementación de dicha extensión, por lo que era la opción de lenguaje más viable para este fin. https://developer.chrome.com/docs/extensions/mv3/devguide

- **Visual Studio Code:** Es un editor ligero y altamente personalizable, su amplia variedad de extensiones y su integración con herramientas de desarrollo hacen que sea una elección popular para el desarrollo en varios lenguajes de programación, y por ello se ha utilizado para escribir el código para la extensión de Chrome con javascript. https://code.visualstudio.com

- **GitHub:** Es una plataforma de control de versiones y colaboración para el proyecto. Su elección se debe a que permite un seguimiento eficiente de los cambios en el código, brinda un repositorio centralizado para el almacenamiento y la gestión del código fuente del proyecto. https://github.com

Capítulo 4

EvaluationDSL

El lenguaje de dominio específico desarrollado ha sido bautizado como EvaluationDSL. El diseño de este DSL ha sido un aspecto fundamental para sentar las bases de su desarrollo.

Tras analizar cómo se estructuran los cursos en la plataforma code.org, partiendo del paradigma de programación orientada a objetos, se han creado las siguientes clases:

- User: Representa al usuario que realiza las actividades.
- Course: Clase que representa a los cursos. Un usuario puede realizar varios cursos.
- Lesson: Hace referencia al contenido de los cursos. Un curso está compuesto de varias lecciones.
- Activity: Se utiliza para representar las actividades que posee una lección.
- Evaluation: Representa la evaluación para una lección realizada por un alumno. Esta clase realiza el cálculo de la nota media de todas las actividades de dicha lección.

También se han definido los siguientes tipos enumerados:

- Gender: indica el género de los usuarios.
- Type: define el tipo de lección.
- Difficulty: especifica la dificultad de la lección.
- Status: define en qué estado se encuentra la actividad.

Repositorio de código del DSL: https://github.com/serele/evaluation-dsl

4.1. Diagrama de clases

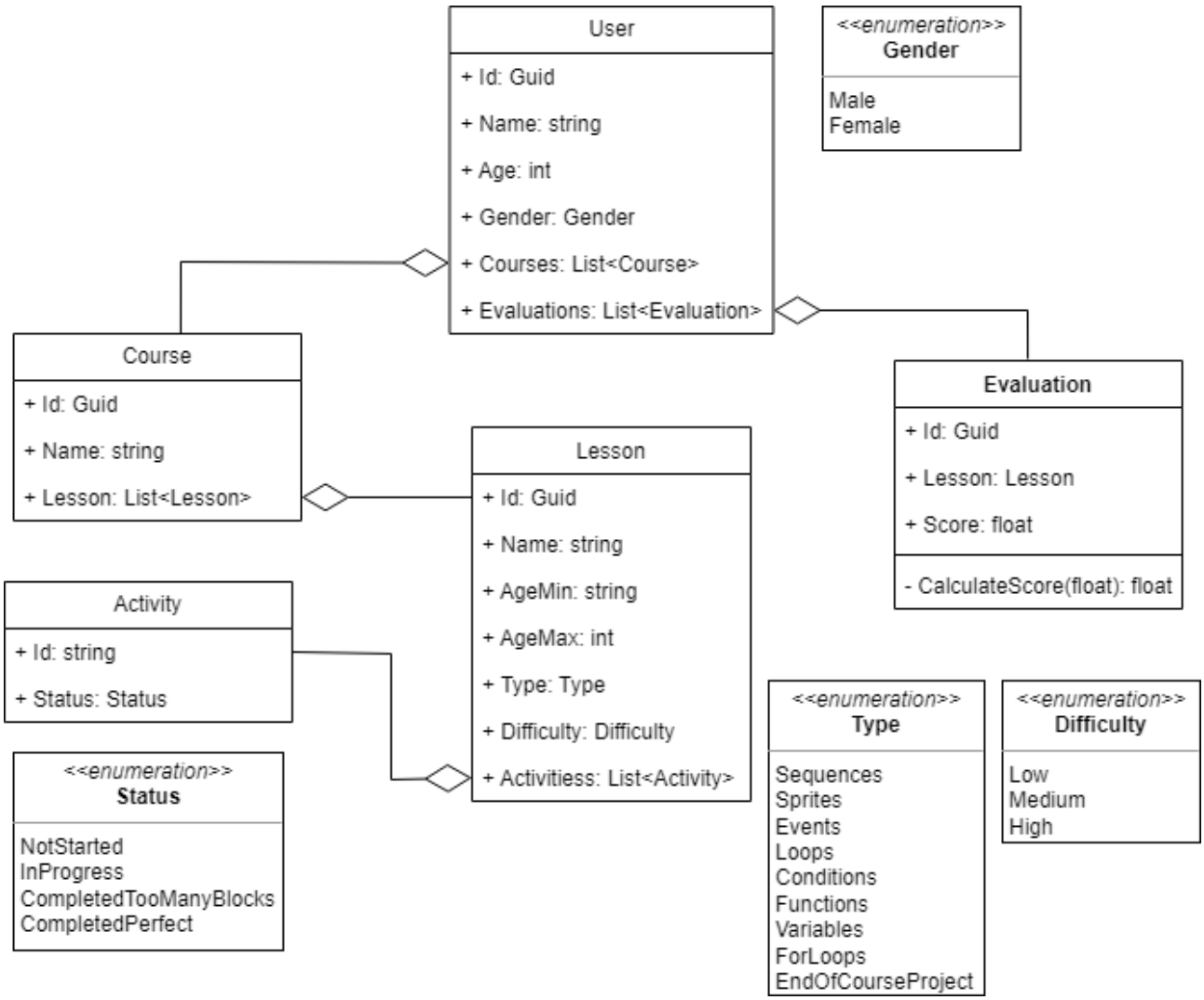


Figura 4.1: Diagrama de clases

4.2. Código del DSL

En .NET el árbol de carpetas se define de la siguiente manera: al directorio principal se le denomina solución y de ella cuelgan los proyectos.

La carpeta Entities del proyecto EvaluationDSL contiene las clases mencionadas.

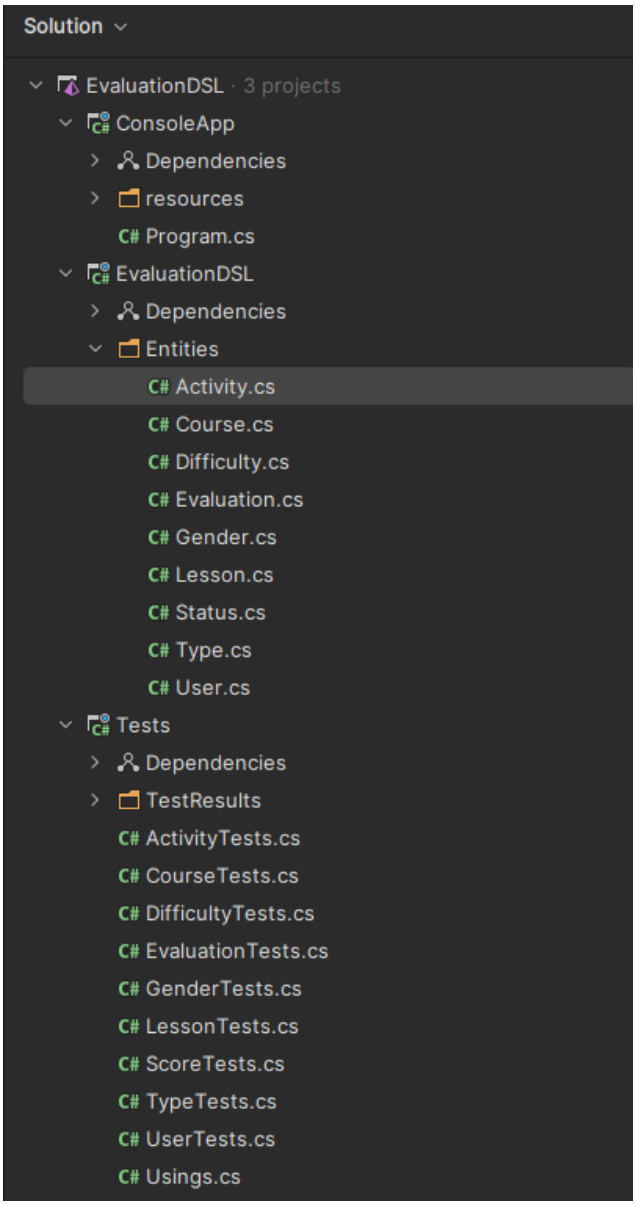
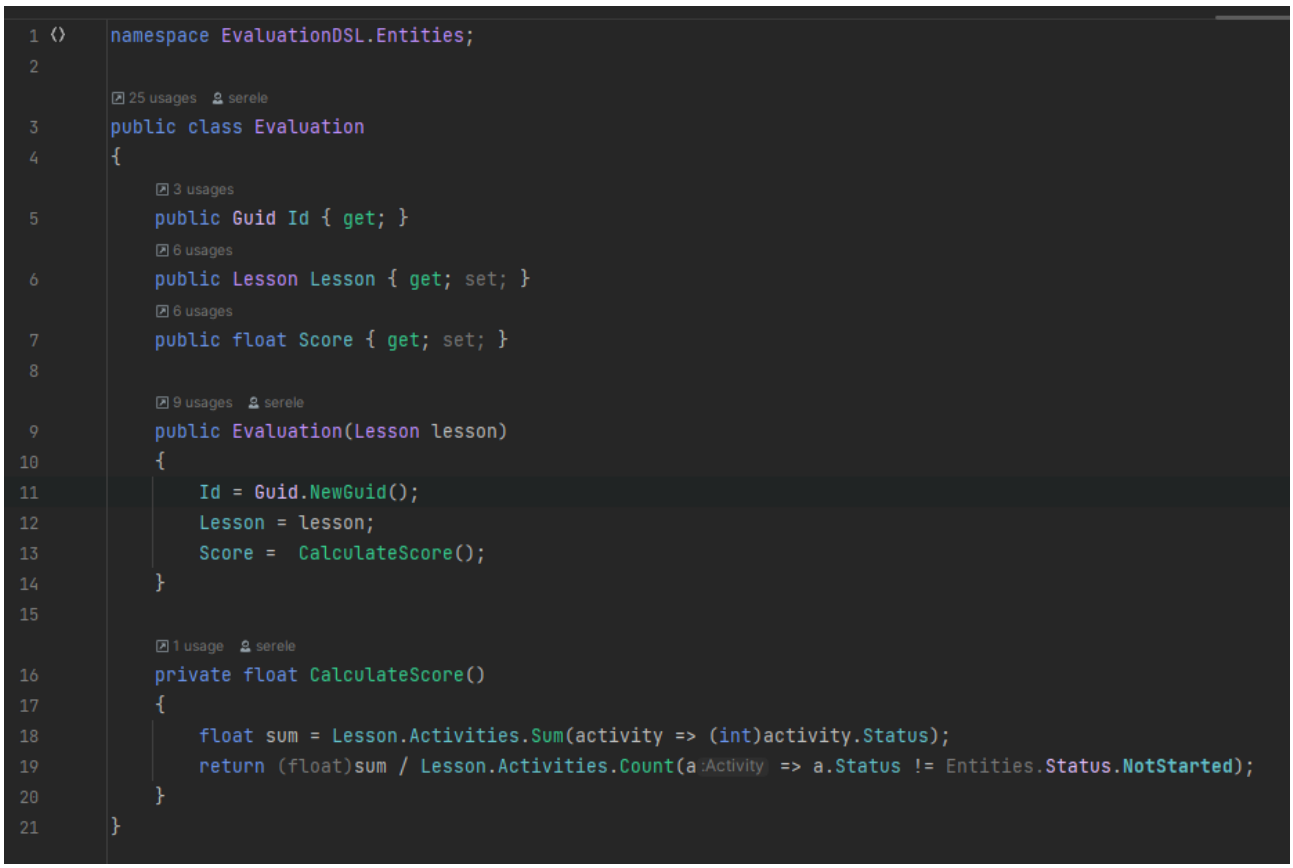


Figura 4.2: Estructura de la solución

La clase Evaluation posee las propiedades Id para dotar de identificador único a cada evaluación; Lesson, la lección sobre la que se aplica la evaluación; y Score, valor que se calcula con la función CalculateScore la cual devuelve el cociente entre sumatorio de la nota de cada actividad y el número de actividades que tiene la lección de dicha evaluación.



```
namespace EvaluationDSL.Entities;

25 usages  serele
public class Evaluation
{
    3 usages
    public Guid Id { get; }
    6 usages
    public Lesson Lesson { get; set; }
    6 usages
    public float Score { get; set; }

    9 usages  serele
    public Evaluation(Lesson lesson)
    {
        Id = Guid.NewGuid();
        Lesson = lesson;
        Score =  CalculateScore();
    }

    1 usage  serele
    private float CalculateScore()
    {
        float sum = Lesson.Activities.Sum(activity => (int)activity.Status);
        return (float)sum / Lesson.Activities.Count(a:Activity => a.Status != Entities.Status.NotStarted);
    }
}
```

Figura 4.3: Clase Evaluation

La clase Lesson se compone de las propiedades Id; Name para el nombre de la lección; AgeMin y AgeMax para el rango de edad al que va dirigida; Type, tipo de lección (secuencías, sprites, eventos, bucles, funciones...); Difficulty, indica la dificultad (baja, media, alta); y por útimo una lista de tipo Activity que contiene las actividades que forman la lección.

Esta clase Lesson posse un constructor que requiere de cada una de las propiedades nombradas anteriormente excepto el Id que se inicializa dentro del mismo como nueva instacia de tipo Guid.

```
namespace EvaluationDSL.Entities;

81 usages  serele  1 exposing API
public class Lesson
{
    3 usages
    public Guid Id { get; }
    3 usages
    public string Name { get; set; }
    2 usages
    public int AgeMin { get; set; }
    2 usages
    public int AgeMax { get; set; }
    2 usages
    public Type Type { get; set; }
    2 usages
    public Difficulty Difficulty { get; set; }
    4 usages
    public List<Activity> Activities { get; set; }

    50 usages  serele
    public Lesson(string name, int ageMin, int ageMax, Type type, Difficulty difficulty, List<Activity> activities)
    {
        Id = Guid.NewGuid();
        Name = name;
        AgeMin = ageMin;
        AgeMax = ageMax;
        Type = type;
        Difficulty = difficulty;
        Activities = activities;
    }
}
```

Figura 4.4: Clase Lesson

Para la clase Activity solo han sido necesarias las propiedades Id y Status. La propiedad Status se utiliza para indicar en qué estado se encuentra la actividad, siendo por defecto NotStarted tal como se inicializa en el constructor, es decir que la actividad no se ha comenzado a realizar.

```
namespace EvaluationDSL.Entities;

172 usages  serele *
public class Activity
{
    4 usages  12 tests OK
    public string Id { get; set; }
    29 usages  12 tests OK
    public Status Status { get; set; }

    93 usages  12 tests OK  serele *
    public Activity(string id)
    {
        Id = id;
        Status = Status.NotStarted;
    }
}
```

Figura 4.5: Clase Activity

La clase Course además de la propiedad Id, está compuesta de la propiedad Name y una lista de tipo Lesson para las lecciones que contiene el curso. El constructor pide como parámetros Name y la lista de Lessons.

```
namespace EvaluationDSL.Entities;

16 usages  serele
public class Course
{
    3 usages  1 test OK
    public Guid Id { get; }
    2 usages  3 tests OK
    public string Name { get; set; }
    2 usages  3 tests OK
    public List<Lesson> Lessons { get; set; }

    5 usages  3 tests OK  serele
    public Course(string name, List<Lesson> lessons)
    {
        Id = Guid.NewGuid();
        Name = name;
        Lessons = lessons;
    }
}
```

Figura 4.6: Clase Course

Por último, la clase User posee las propiedades Id, Name, Age, Gender, Courses (cursos que ha realizado) y Evaluations (evaluaciones aplicadas)

```
namespace EvaluationDSL.Entities;

8 usages  serele
public class User
{
    3 usages  1 test OK
    public Guid Id { get; }
    3 usages  2 tests OK
    public string Name { get; set; }
    3 usages  2 tests OK
    public int Age { get; set; }
    3 usages  2 tests OK
    public Gender Gender { get; set; }
    3 usages  2 tests OK
    public List<Course> Courses { get; set; }
    3 usages  2 tests OK
    public List<Evaluation> Evaluations { get; set; }

    4 usages  2 tests OK  serele
    public User(string name, int age, Gender gender, List<Course> courses, List<Evaluation> evaluations)
    {
        Id = Guid.NewGuid();
        Name = name;
        Age = age;
        Gender = gender;
        Courses = courses;
        Evaluations = evaluations;
    }
}
```

Figura 4.7: Clase User

4.3. CODE.ORG SPY

CODE.ORG SPY es la herramienta de extracción de datos de las actividades realizadas por los usuario en la plataforma code.org es una extensión para el navegador Chrome.

Está codificada en javascript. Su función es generar ficheros json mediante el parseo del DOM de code.org. Estos ficheros json contienen la información necesaria para posteriormente realizar el proceso de evaluación. Repositorio de código: https://github.com/serele/code.org-spy

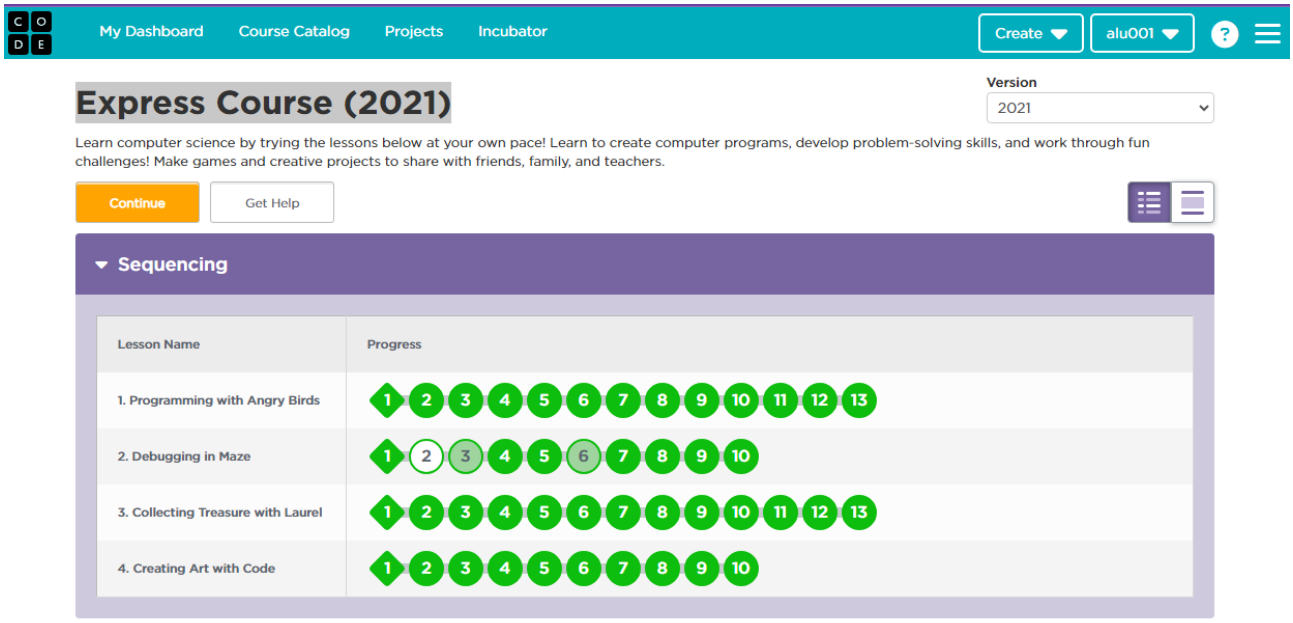


Figura 4.8: Express Course (2021) en code.org

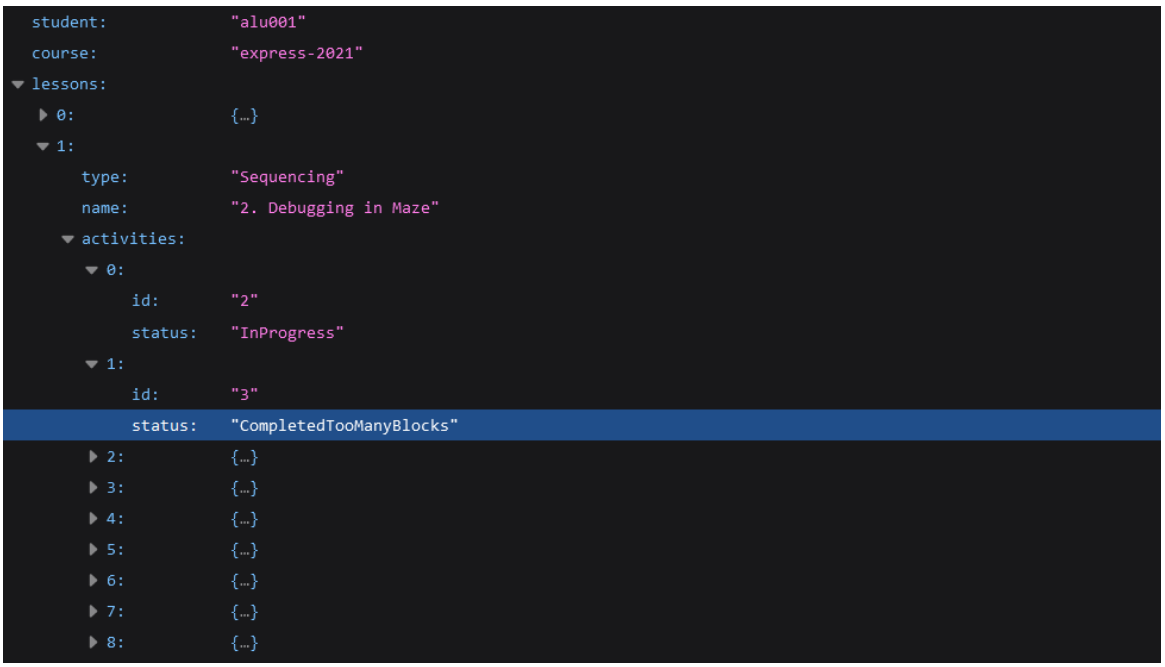


Figura 4.9: Vista de fichero json generado a partir de Express Course (2021)

4.4. Verficación y pruebas

xUnit.net es una herramienta de desarrollo guiado por pruebas para .NET Framework, Core y 5+. Es un conjunto de paquetes NuGet que se pueden instalar en cualquier proyecto.

El proyecto Tests se ha añadido a la solución EvaluationDSL mediante la siguiente orden:

```
dotnet new xunit -o EvaluationDSL.Tests
```

Se han definido tests unitarios con xUnit para todas las clases y sus constructores y funciones de manera que se ha conseguido un 100 % de cobertura, evitando así errores que se puedan ocasionar.

Hay por cada una de las entidades anteriormente mencionadas (ver Figura 4.2) se ha creado una clase de Tests, en las que se comprueba que el Id sea único, que el constructor establece los valores de las propiedades correspondientes de manera adecuada, comprobar que dos objetos con diferentes valores en sus propiedades no sean iguales, y en el caso de la clase Evaluation comprobar que el resultado de la función que calcula la nota sea el correcto.

Figura 4.10: Cobertura de pruebas desde el IDE Rider

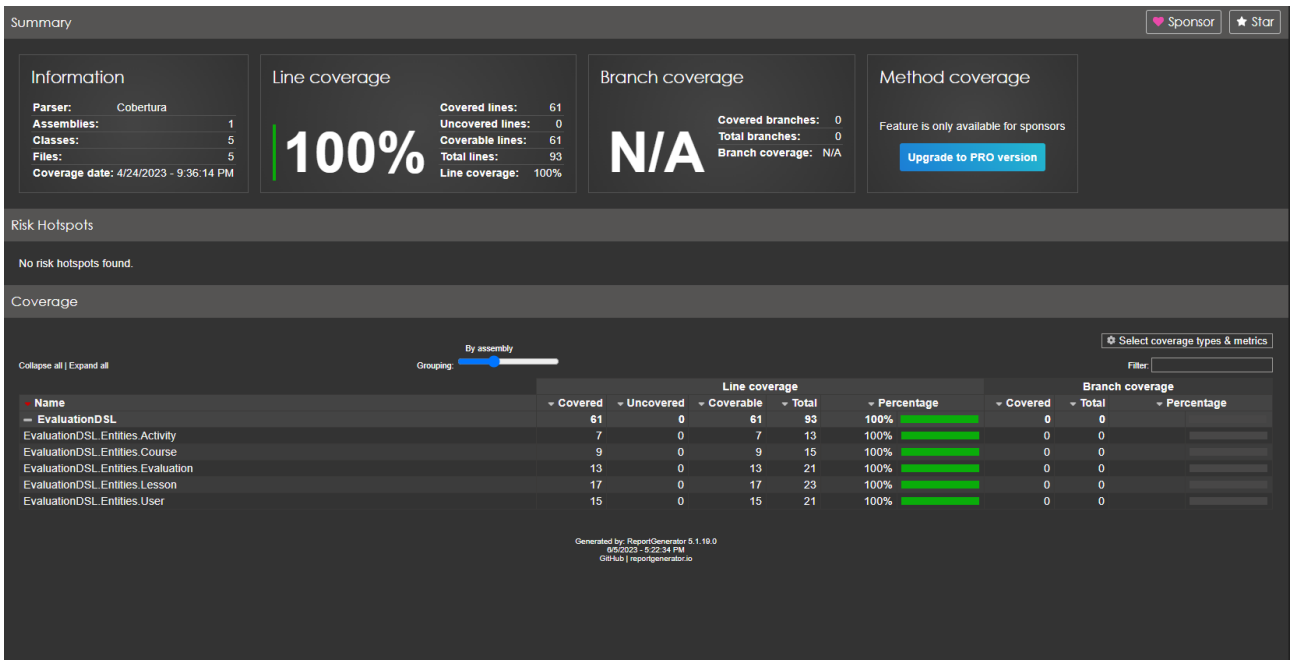


Figura 4.11: Cobertura de pruebas desde ReportGenerator

La figura 4.11 muestra una vista del informe de cobertura generado con la herramienta ReportGenerator. Para una vista más detallada visitar el siguiente enlace al GitHub del proyecto.

A continuación se describirá una prueba real siguiendo todos los pasos a partir de la realización de un curso como usuario de la plataforma code.org.

Tomaremos como ejemplo un alumno llamado alu001. Como vemos en la figura 4.12, dicho alumno ha realizado el curso llamado Express Course (2021) y en este curso ha contestado a las actividades que pertenecen a las lecciones 1 a la 6.

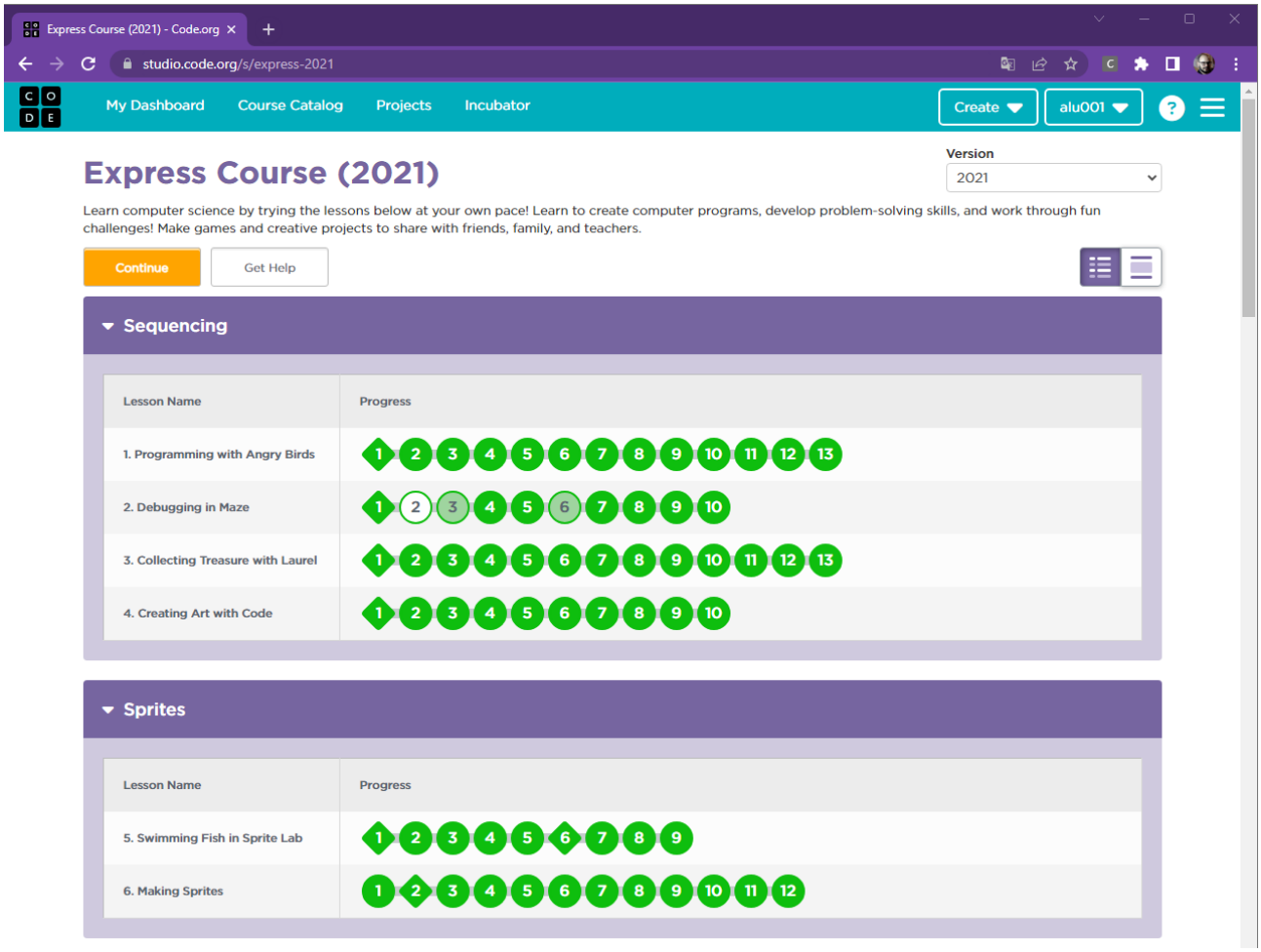


Figura 4.12: Lecciones de alu001 para el curso Express Course (2021)

En la columna Progress aparecen las actividades correspondientes a cada lección, pudiendo además ser conceptos teóricos (los cuales no se evalúan) o actividades propiamente dichas.

Estas actividades se representan de diferentes formas dependiendo del estado de la actividad:

- Borde gris y fondo blanco la actividad no ha sido iniciada (no se evalúa)
- Borde verde y fondo blanco la actividad está en progreso (se evalúa con un 0)

- Borde verde y fondo verde tenue la actividad ha sido completada en demasiados pasos (se evalúa con un 5)
- Borde verde y fondo verde intenso la actividad ha sido completada correctamente (se evalúa con un 10)

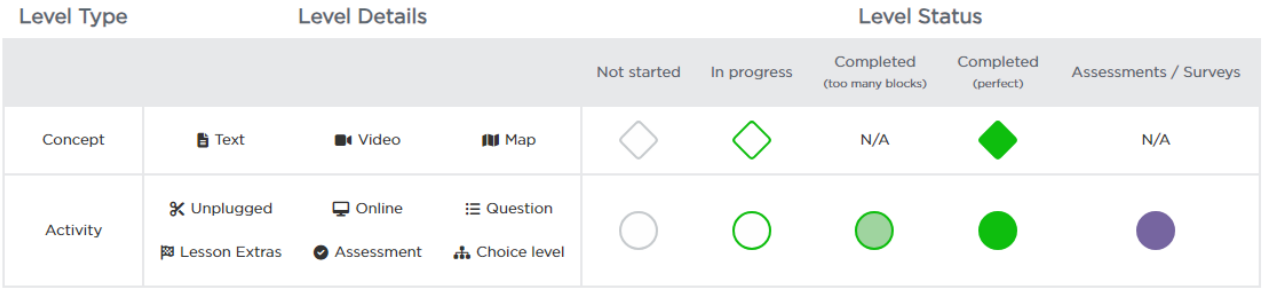


Figura 4.13: Leyenda de code.org

Vamos a hacer uso de la herramienta CODE.ORG SPY, la extensión de Chrome que generará el fichero json. Hacemos click en el botón con la letra C, luego en Descargar JSON, y guardamos el fichero.

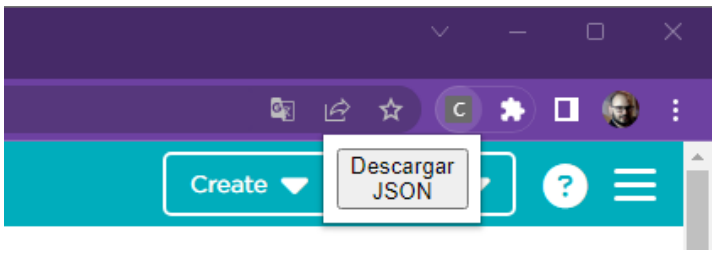


Figura 4.14: Extensión CODE.ORG SPY

Este fichero es el que utilizaremos para el proceso de evaluación mediante el DSL. Tras ejecutar la herramienta EvaluationDSL sobre el fichero json que acabamos de generar obtenemos la siguiente salida:

```
STUDENT: alu001
COURSE: express-2021

LESSON: 1. Programming with Angry Birds
SCORE: 10

LESSON: 2. Debugging in Maze
SCORE: 8.75

LESSON: 3. Collecting Treasure with Laurel
SCORE: 10

LESSON: 4. Creating Art with Code
SCORE: 10

LESSON: 5. Swimming Fish in Sprite Lab
SCORE: 10

LESSON: 6. Making Sprites
SCORE: 10

LESSON: 7. Sprites in Action
SCORE: NaN

LESSON: 8. Virtual Pet with Sprite Lab
SCORE: NaN
```

Figura 4.15: Resultados EvaluationDSL

Como se puede observar los resultados coinciden con la anteriormente descrito, realizándose el cálculo de para la puntuación de las actividades de cada lección. A partir de la lección 6 no se obtiene ninguna nota puesto que las actividades de estas lecciones no han sido iniciadas por el usuario.

Capítulo 5

Conclusiones y líneas futuras

A través de este proyecto, se ha logrado alcanzar los siguientes resultados y conclusiones:

- Diseño y desarrollo del DSL: Se ha logrado diseñar y desarrollar un lenguaje de dominio específico que permite evaluar de manera efectiva las competencias y habilidades relacionadas con el pensamiento computacional de la plataforma code.org.
- Integración en la plataforma code.org: Se ha logrado integrar con éxito el DSL en la plataforma code.org (mediante una extensión de Chrome), proporcionando a los educadores una herramienta adicional para evaluar las habilidades de pensamiento computacional de los estudiantes.
- Evaluación de la efectividad del DSL: Se han desarrollado pruebas unitarias para evaluar la efectividad del DSL.

Líneas Futuras:

- Ampliar las funcionalidades del DSL: En futuras investigaciones, se podrían incorporar nuevas características y funcionalidades al DSL, de manera que pueda evaluar una gama más amplia de habilidades de pensamiento computacional. Esto permitiría una evaluación más completa y precisa de las capacidades de los estudiantes en este ámbito.
- Mejorar la retroalimentación proporcionada por el DSL: Se podría trabajar en el perfeccionamiento de la retroalimentación proporcionada por el DSL, con el objetivo de ofrecer una evaluación más detallada y personalizada de las actividades de pensamiento computacional. Esto podría incluir la incorporación de comentarios específicos y recomendaciones para el desarrollo de habilidades.

- Ampliar la integración con otras plataformas educativas: Además de la plataforma code.org, se podrían explorar oportunidades para integrar el DSL en otras plataformas educativas y entornos de aprendizaje en línea. Esto ampliaría el alcance y el impacto del DSL, permitiendo a un mayor número de educadores y estudiantes beneficiarse de su uso.
- Validación y extensión del estudio experimental: Para respaldar aún más la efectividad del DSL, se podría llevar a cabo una validación adicional del estudio experimental, ampliando la muestra de participantes y considerando diferentes contextos educativos. Además, se podrían explorar otros métodos de evaluación, como la comparación con otros enfoques de evaluación existentes.

Capítulo 6

Summary and Conclusions

The development of the DSL for evaluating computational thinking activities on code.org has been a success. The DSL's design emphasized clarity, high-level abstractions, and flexibility. Its integration into code.org offered a seamless user experience. Future work could focus on expanding the DSL's functionality, creating feedback mechanisms, integrating with other educational platforms, and validating the experimental study in different contexts. Overall, this research contributes to enhancing computational thinking education and assessment through an effective DSL on the code.org platform.

Capítulo 7

Presupuesto

Tarea	Horas	Coste hora
Revisión bibliográfica	25	10 €
Estudio de antecedentes	25	10 €
Diseño y desarrollo de DSL	100	20 €
Diseño y desarrollo de extensión de chrome	20	20 €
TOTAL	170	2900 €

Tabla 7.1: Presupuesto

Bibliografía

[1] N. Ford. Building domain specific languages in c#. *CODE Magazine*, enero-febrero, 2009.

[2] Martin Fowler. *Domain-specific languages*. Pearson Education, 2010.

[3] Yasmin B Kafai and Quinn Burke. *Connected code: Why children need to learn programming*. Mit Press, 2014.

[4] Mariano Luzza, Mario Berón, Germán Antonio Montejano, Mario Peralta, and Pedro Rangel Henriques. Diseño y construcción de lenguajes específicos del dominio. In *XIV Workshop de Investigadores en Ciencias de la Computación*, 2012.

[5] Lauri Malmi, Judy Sheard, Päivi Kinnunen, Simon, and Jane Sinclair. Development and use of domain-specific learning theories, models, and instruments in computing education. *ACM Transactions on Computing Education*, 23(1):1–48, 2022.

[6] Marjan Mernik, Jan Heering, and Anthony M Sloane. When and how to develop domain-specific languages. *ACM computing surveys (CSUR)*, 37(4):316–344, 2005.

[7] Carlos Enrique Montenegro Marín, Paulo Alonso Gaona García, JUAN CUEVA LOVELLE, and Óscar Sanjuán Martínez. Aplicación de ingeniería dirigida por modelos (mda), para la construcción de una herramienta de modelado de dominio específico (dsm) y la creación de módulos en sistemas de gestión de aprendizaje (lms) independientes de la plataforma. *Dyna*, 78(169):43–52, 2011.

[8] Xabier Basogain Olabe, Miguel Ángel Olabe Basogain, and Juan Carlos Olabe Basogain. Pensamiento computacional a través de la programación: Paradigma de aprendizaje. *Revista de educación a distancia (RED)*, (46), 2015.

[9] Ayende Rahien. *DSLs in Boo: Domain Specific Languages in. Net*. Manning Publications Co., 2010.