De la estructura a la imagen: Visualizando datos JSON a través de la programación

From Structure to Image: Visualizing JSON Data Through Programming

Joel Tapia Flores[1], Misael López Ramírez[2]
[1,2] Departamento de estudios Multidisciplinarios, universidad de Guanajuato, Yuriria Guanajuato, México
{ j.tapiaflores, lopez.misael }@ugto.mx[1,2]

Resumen

En la actualidad la enseñanza de la programación se ha vuelto una herramienta indispensable en el ramo de la educación. Por ello, muchas investigaciones se centran en los diferentes modelos de su enseñanza. Sin embargo, estas investigaciones solo se centra en encontrar la mejor forma para su enseñanza, o en generar rutas personalizadas que ayuden a mejorar las hábilidades de programación en cada niño y no en la detección de errores que puede haber al momento de programar.

Por lo anterior, nuestra investigación se centra en una herramienta que facilite la comprensión de la estructura interna de las soluciones intermedias generadas por los alumnos y permita realizar análisis estáticos de una manera más intuitiva. Como resultado obtuvimos una herramienta que convierte los Árboles de Sintaxis Abstracta (AST) a un formato de visualización estático, el cuál nos dará como beneficios detectar los posibles errores que se tengan al momento de programar.

Palabras clave: Árboles de Sintaxis Abstracta, Diagramas, Soluciones intermedias, Soluciones Óptimas.

Introducción

Hoy en día la enseñanza de la programación para niños se ha vuelto tan importante como la lectura y la escritura. Por este motivo, diversos autores han optado por utilizar herramientas de programación basadas en bloques(Aivaloglou et al., 2017) como Scratch, (Meerbaum-Salant et al., 2013) y CODE.org (Kalelioɬlu, 2015).

Dichas herramientas son creadas bajo el enfoque de la gamificación, además de adoptar un diseño de aprendizaje que permite a los niño desarrollar sus hábilidades de Pensamiento Computacional (*Computational Thinking* , CT, por sus siglas en Ingles) por medio de la programación.

Para analizar los códigos creados en estas herramientas algunos autores han utilizado los AST, que son representaciones jerarquicas de la estructura de código de un programa. Estos permiten evaluar la forma de aprendizaje de los niños en programación y su desarrollo del CT. Sin embargo, estas investigaciones aunque tratan las diferencias que pueden existir entre los AST óptimos y los AST intermedios no tiene como relevación el encontrar o mencionar de manera precisa los errores encontrados en las soluciones intermedias.

Por lo anterior, nuestra investigación se enfoca en la detección de errores en las Soluciones Intermedias. Para lo cual hemos creado una herramienta que convierte los AST de formato JSON a representaciones gráficas, dichas representaciones nos permitirán encontrar los errores exactos al momento de programar. Para esta investigación, hemos tomamos 3 ejercicios de programación de bloques para analizar su flujo de desarrrollo y las relaciones entre las diferentes partes del código y así poder detectar patrones de errores que son difíciles de apreciar en el códgio fuente, esto nos permitirá comprender mejor la lógica de programación y evaluar la eficiencia de diferentes soluciones.

Marco Teorico

En la tabla número 1 mostramos las diferentes investigaciones que se han hecho de los AST como herramientas de medición para el aprendizaje en la programación

Tabla 1*. Trabajos que utilizan los AST para medir el aprendizaje de programación*

Literatura	Plataforma utilizada	Ejercicios	Resultado
Análisis de árboles de sintaxis abstracta (AST) (Jiang et al., 2020).	CODE.org	• HOC4 • HOC18	Distancia entre la solución objetivo y las soluciones creadas por los estudiantes
Evaluación del proceso de resolución de problemas a escala (Grover et al., 2016).	Blockly games	• Laberinto • Estanque	Comparación concisa y significativa de instantáneas de programas
Generación de Scripts de edición AST más simples considerando copiar y pegar(Higo et al., 2017).	GitHub	• Cambios de código de 1,000 proyectos de GitHub	Técnica de generación de guiones de edición de AST capaz de identificar las operaciones correctas de copiar y pegar

Metodología Propuesta

Para poder visualizar los AST de forma gráfica y detectar sus posibles diferencias y errores proponemos una metodología compuesta de 5 etapas:

- **Etapa 1:** Selección de ejercicios.
- **Etapa 2:** Realización de la solución óptima.
    - Generación de los AST óptimos.
- **Etapa 3:** Generación de los AST de las soluciones intermedias.
- **Etapa 4:** Realización del programa que generara los AST de manera gráfica
- **Etapa 5:** Resultados.

Etapa 1:

En esta primera etapa es seleccionar los ejercicios que servirán de estudio para la presente investigación. El objetivo principal de estos ejercicios es que los niños guíen al Pegman (figura amarilla) hasta la Elfa siguiendo una secuencia de bloques de programación visual. La figura 1 muestra dichos ejercicios.

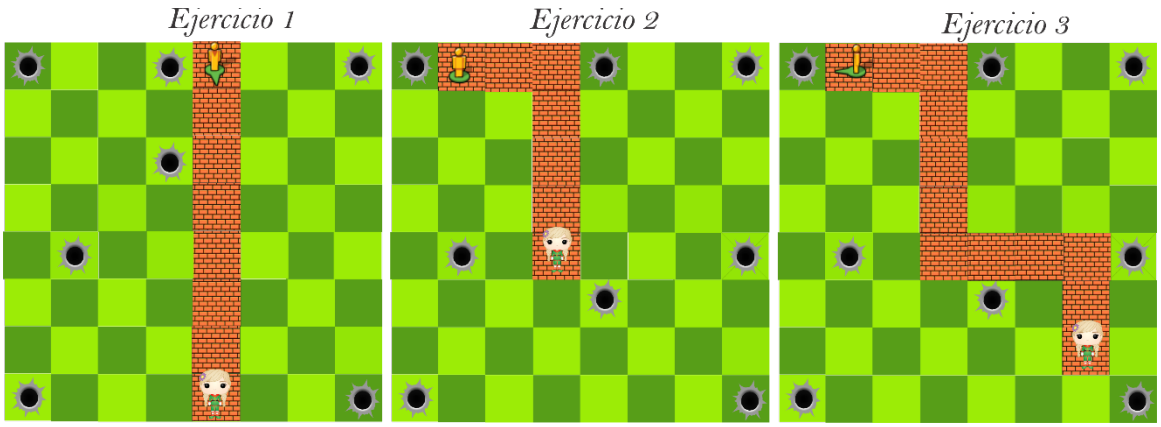


Figura 1. Ejercicios para la generación de los AST.
Fuente: Autoría propia

Etapa 2:

En esta segunda etapa, se elaboraron las soluciones óptimas para los tres ejercicios utilizando la menor cantidad de bloques posible para garantizar una resolución correcta. La optimidad de las soluciones se determinó en base al número total de bloques utilizados. Estas soluciones servirán como referencia para comparar con las respuestas de los estudiantes y así identificar patrones de error y evaluar su comprensión de los conceptos de programación. En la figura 2 mostramos las soluciones óptimas de cada ejercicio.

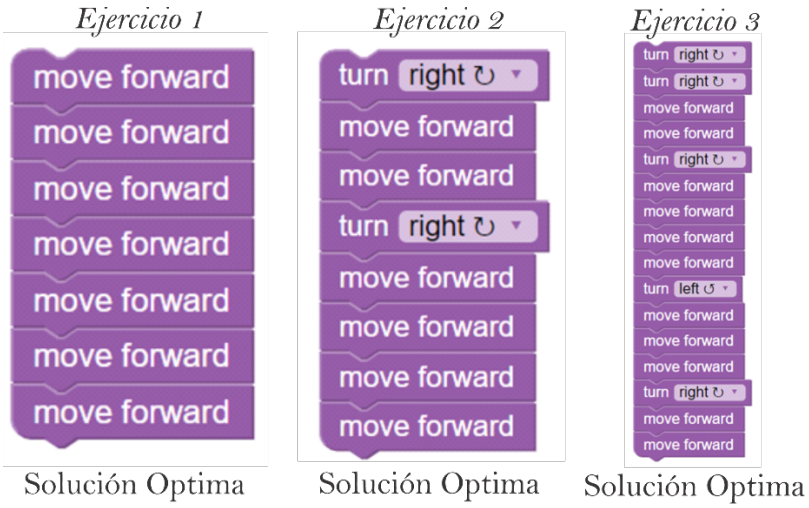


Figura 2. Soluciones optimas de los ejercicios propuestos
Fuente: Autoría propia

Generación de los AST optimos:

Con las soluciones optimas ya realizadas, el segundo paso es convertir los bloques utilizados en lineas de código. Para esto, debemos de ejecutar cada una de nuestras respuestas (ver figura 2) y automaticamente

```
<xml xmlns="http://www.w3.org/1999/xhtml"><block type="move" id="),Zd:m`9uCu1l~BwuCe^" x="19" y="34"><next><block type="move"
id="2U}4huFKRm.fq!k*hS9_"><next><block type="move" id="_Nv128NA%u0!KI%y0$TR"><next><block type="move" id=")fXV},GMzmn+^s$F%L?,">
<next><block type="move" id="O^]7Dpg%v.{i262#/]|z"><next><block type="move" id=":}@%HKlQ7/frsfV#q|(."><next><block type="move"
id="xa)KP9B6F+-qNgu~S+t%"></block></next></block></next></block></next></block></next></block></next></block></next></block></xml>
```

nuestra plataforma web guardará nuestras líneas de código en formato XML. En las figuras 3 mostramos el formato XML de la solución optima del ejercicio 1.

Figura 3. Solución del ejercicio 1 en formato XML
Fuente: Autoría propia

Una vez guardado el código en formato XML, desarrollamos un programa en Python que nos permitio la conversión de las lineas de código a un AST en formato JSON. En la figura 4 mostramos la solución optima del ejercicio 1 en forma de AST.

```
{
    "type": "move",
    "id": "),Zd:m`9uCu1l~BwuCe^",
    "x": "19",
    "y": "34",
    "fields": {},
    "next": {
        "type": "move",
        "id": "2U}4huFKRm.fq!k*hS9_",
        "x": null,
        "y": null,
        "fields": {},
        "next": {
            "type": "move",
            "id": "_Nv128NA%u0!KI%y0$TR",
            "x": null,
            "y": null,
            "fields": {},
            "next": {
                "type": "move",
                "id": "}fXV},GMzmn+^s$F%L7,",
                "x": null,
                "y": null,
                "fields": {},
                "next": {
                    "type": "move",
                    "id": "O^]7Dpg%v.{i262#/||z",
                    "x": null,
                    "y": null,
                    "fields": {},
                    "next": {
                        "type": "move",
                        "id": ":}@%HKlQ7/frsfV#q|(.",
                        "x": null,
                        "y": null,
                        "fields": {},
                        "next": {
                            "type": "move",
                            "id": "xa)KP9B6F+-qNgu~S+t%",
                            "x": null,
                            "y": null,
                            "fields": {}
                        }
                    }
                }
            }
        }
    }
}
```


Figura 4. AST Óptimo del ejercicio 1
Fuente: Autoría propia

Etapa 3:

En esta tercera etapa generaremos los AST de las soluciones intermedias de los niños. Para esto repetiremos los pasos anteriores de la etapa 2. Primero el niño colocara los bloques necesarios para la solución del problema, luego ejecutara dicha solución y la plataforma se encargará de guardarlos en formato XML tantas veces como el niño las genere. Una vez que el niño haya realizado los tres ejercicios, nos dispondremos a generar los AST de todas las soluciones intermedias.

Etapa 4:

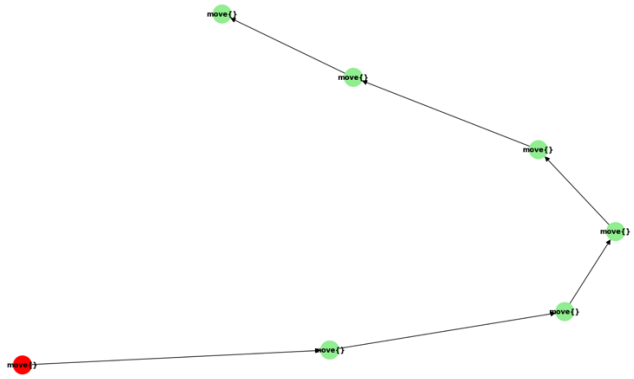
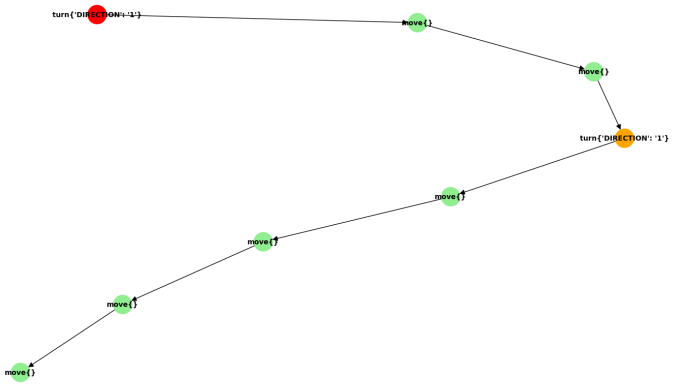
Ahora que ya generamos todos los AST en formato JSON, en esta cuarta etapa desarrollamos un programa en Phyton que nos permitió visualizar los AST de manera gráfica. Para ello nos apoyamos de la librería Json que nos permite leer y escribir datos en formato JSON, también nos apoyamos de la librería networkx la cual permite crear, modificar y analizar grafos, como última librería utilizamos Matplotlib la cual es una biblioteca de visualización de datos en 2D y 3D, la cual nos permite crear cualquier tipo de gráfico.

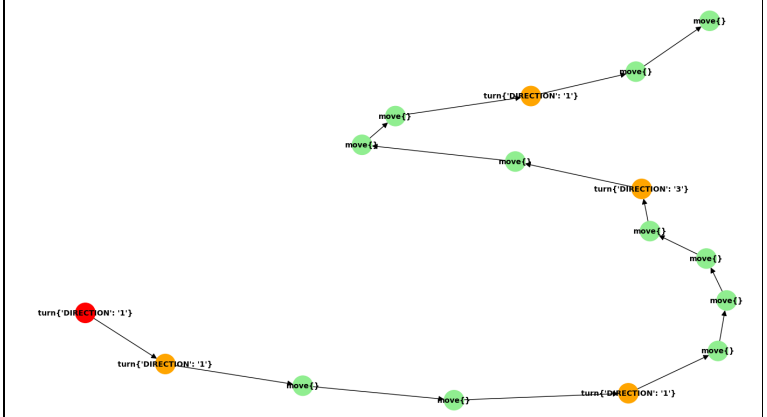
Con estas bibliotecas instaladas en nuestro entorno de programación nos dispusimos a programar. Con la funcion *def add_node_and_edges* (graph, node, parent = none) agregamos los nodos y las aristas recursivamente. Con la línea de código *graph.add_node(node["id"], label=node_label)* nos permitió agregar el nodo al grafo. Los resultados de nuestro programa los mostramos en la sección de resultados.

Resultados

Los resultados obtenidos al momento de ejecutar nuestro código fue la conversión de los AST de formato JSON a un formato gráfico. Esto nos ayudará a identificar fácilmente las diferencias entre las soluciones intermedias y las optimas. En la tabla 2 mostramos los bloques de de las soluciónes optimas asi como los AST optimos de los tres ejercicios, pero ahora de manera gráfica.

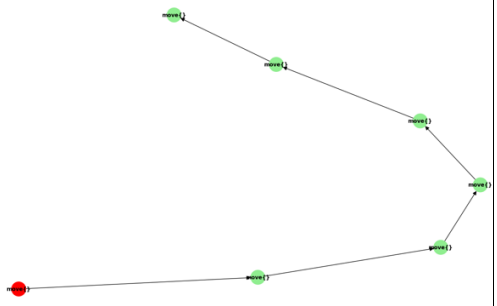
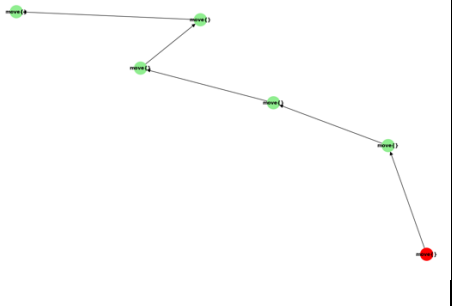
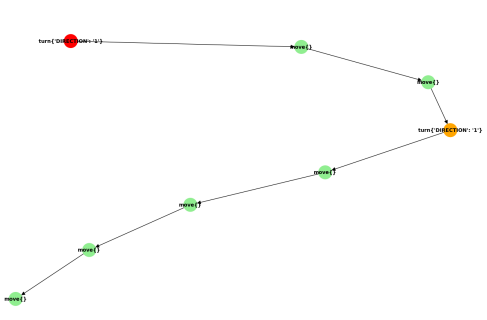
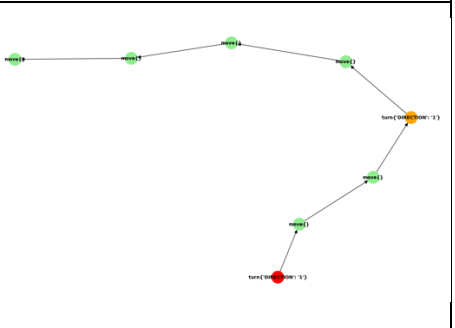
***Tabla 2**. Bloques y AST óptimos de formato gráfico para los tres ejercicios*

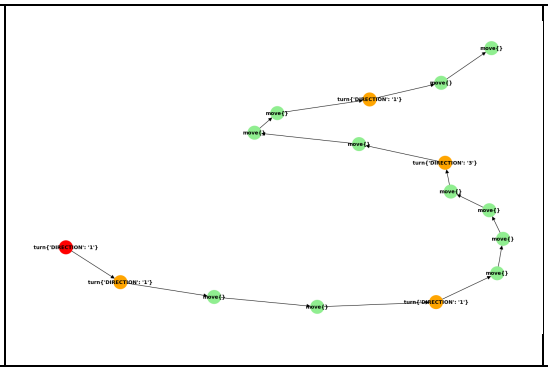
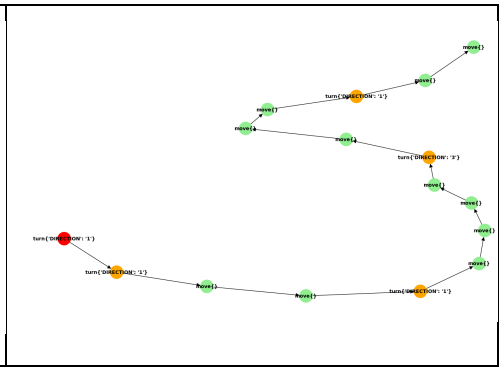
Ejercicio	Bloques	AST óptimo
1		
2		

3		

Como vemos en la tabla 2 ahora podemos ver convertidos los bloques de código en AST de formato gráfico, esto, nos permitira comparar de una manera más sencilla las soluciones intermedias con las soluciones óptimas. En la tabla 3 mostramos las soluciones optimas y las primeras soluciones intermedias de uno de nuestros participantes.

***Tabla 3**. Ejercicios con soluciones optimas e intermedias*

Ejercicio	AST Óptimo	AST Intermedio
1		
2		

3		

Con lo mostrado en la tabla 3 podemos realizar un analisis acerca de las soluciones intermedias. Pero primero se puede ver que nuestros AST tienen 3 colores diferentes:

- **Los nodos con color Rojo** nos indica donde donde comienza nuestros AST.
- **Los nodos con color Verde** nos indican los bloques de movimiento.
- **Los nodos con color Amarillo** nos indican los bloques de giro
    - **Direction 1:** Giro a la derecha
    - **Direction 3:** Giro a la izquierda

Con los datos anteriores ahora podemos realizar un analisis de uno nuestro usuario respecto a las soluciones óptimas:

- Ejercicio 1: para el ejercicio 1 podemos ver que la solución optima cuenta con un nodo de inicio el cual es un bloque move, en la solución intermedia tambien cuenta con este mismo bloque. El número total de nodos verdes para la solución optima es de 6 mientras que para la solución intermedia es de 5, lo que nos quiere decir que a nuestro usuario le falto un bloque de movimiento para llegar a la solucion óptima del problema.
- Ejercicio 2: para el ejercicio 2 vemos que ambas soluciones cuentan con un nodo de inicio, así mismo cuenta con 6 bloques de movimiento y un nodo de giro los cuales tienen la misma dirección de giro (a la derecha) lo que nos indica que el usuario llego a la solución óptima.
- Ejercicio 3: para el ejercicio 3 tambien el usuario también logro llegar a la solución optima ya que la estructura de los AST son las mismas.

Conclusiones

Como conclusiones destacamos que nuestro programa funciona de manera adecuada el cual nos permitirá realizar analisis acerca de las estructuras de las soluciones intermedias respecto a la soluciones óptimas y así poder encontrar los errores y diferencias que tiene el niño respecto a las respuestas optimas. En trabajos a futuro nuestra meta es no sólo generar los AST de manera gráfica si no tambien realizar un análisis automatico de las diferencias encontradas entre los AST intermedios y los optimos.

Referencias

Aivaloglou, E., Hermans, F., Moreno-Leon, J., & Robles, G. (2017). A dataset of scratch programs: Scraped, shaped and scored. *IEEE International Working Conference on Mining Software Repositories*, 511–514. https://doi.org/10.1109/MSR.2017.45

Grover, S., Bienkowski, M., Niekrasz, J., & Hauswirth, M. (2016). Assessing problem-solving process at scale. *L@S 2016 - Proceedings of the 3rd 2016 ACM Conference on Learning at Scale*, 245–248. https://doi.org/10.1145/2876034.2893425

Higo, Y., Ohtani, A., & Kusumoto, S. (2017). *Generating Simpler AST Edit Scripts by Considering Copy-and-Paste*.

Jiang, B., Wu, S., Yin, C., & Zhang, H. (2020). Knowledge Tracing within Single Programming Practice Using Problem-Solving Process Data. *IEEE Transactions on Learning Technologies*, *13*(4), 822–832. https://doi.org/10.1109/TLT.2020.3032980

Kalelioʇlu, F. (2015). A new way of teaching programming skills to K-12 students: Code.org. *Computers in Human Behavior*, *52*, 200–210. https://doi.org/10.1016/j.chb.2015.05.047

Meerbaum-Salant, O., Armoni, M., & Ben-Ari, M. (Moti). (2013). Learning computer science concepts with Scratch. *Computer Science Education*, *23*(3), 239–264. https://doi.org/10.1080/08993408.2013.832022