

UNIVERSIDAD TÉCNICA FEDERICO SANTA MARÍA
DEPARTAMENTO DE INFORMÁTICA
SAN JOAQUÍN - CHILE



“ESTUDIO E IMPLEMENTACIÓN DE UNA SERIE DE
EJERCICIOS HABILITADORES AUTO CORREGIBLES PARA
CURSOS INTRODUCTORIOS DE PROGRAMACIÓN EN LA
PLATAFORMA MOODLE”

MATÍAS IGNACIO MARCHANT AYARZA

MEMORIA PARA OPTAR AL TÍTULO DE
INGENIERO CIVIL EN INFORMÁTICA

Profesor Guía: Federico Meza
Profesor Correferente: Andrea Vásquez

Enero - 2024

DEDICATORIA

A mis padres.

RESUMEN

Resumen— La programación ha ganado una enorme popularidad en la sociedad actual debido a la creciente relevancia de la tecnología, pero aprenderla puede ser difícil para muchas personas debido a su intrínseca dificultad.

En este trabajo de título se ha implementado una serie de ejercicios habilitadores con andamiaje para cursos introductorios de programación en la plataforma Moodle, este tipo de ejercicios son beneficiosos para el aprendizaje ya que reducen la carga cognitiva a la que se ven enfrentados los estudiantes. La solución propuesta fue atractiva para la mayoría de los estudiantes que la probaron, y expresaron deseo de seguir aprendiendo programación usándolos.

Palabras Clave— Andamiaje; Moodle; carga cognitiva; ejercicios de programación

ABSTRACT

Abstract— Programming has gained enormous popularity in today's society due to the increasing relevance of technology, but learning it can be difficult for many people due to its intrinsic difficulty.

In this work, a series of exercises with scaffolding have been implemented for introductory programming courses in the Moodle platform. These types of exercises are beneficial for learning as they reduce the cognitive load that students face. The proposed solution was attractive to the majority of the students who tried it, and they expressed a desire to continue learning programming using them.

Keywords— Scaffolding; Moodle; cognitive load; programming exercises

GLOSARIO

LTS: *Long-Term Support*.

SCT: Sistema de Créditos Transferibles.

SMOJ: *Santa María Online Judge*.

UTFSM: Universidad Técnica Federico Santa María.

TCC: Teoría de la Carga Cognitiva.

VPD: *Visually Paired Distractor*.

ÍNDICE DE CONTENIDOS

RESUMEN	III
ABSTRACT	III
GLOSARIO	IV
ÍNDICE DE FIGURAS	VII
ÍNDICE DE TABLAS	VIII
INTRODUCCIÓN	1
CAPÍTULO 1: DEFINICIÓN DEL PROBLEMA	3
1.1 Situación actual	3
1.2 Dificultades del problema	9
1.3 Soluciones existentes	9
1.3.1 Lenguajes de programación basados en bloques	9
1.3.2 Tipo de pregunta respuestas anidadas (Cloze)	10
1.4 Objetivos	12
1.4.1 Objetivo general	12
1.4.2 Objetivos específicos	12
CAPÍTULO 2: MARCO CONCEPTUAL	13
2.1 Moodle	13
2.1.1 Arquitectura modular de Moodle	13
2.1.2 Módulos de tipo de pregunta	13
2.2 Teoría de la carga cognitiva	15
2.2.1 Teoría de la carga cognitiva en la investigación de la educación computacional	16
2.3 Ejercicios habilitadores	16
2.3.1 Problemas de Parsons	16
2.3.2 Ruteo de código	17
2.3.3 Completar partes	18
2.4 Andamiaje	18
CAPÍTULO 3: PROPUESTA DE SOLUCIÓN	20
3.1 Tipo de pregunta: Completar partes	22
3.2 Tipo de pregunta: ruteo de código	28
3.3 Tipo de pregunta: Problemas de Parsons	34
3.3.1 Parsonsificador	37
CAPÍTULO 4: VALIDACIÓN DE LA SOLUCIÓN	45
4.1 Validación con estudiantes	45

4.1.1	Metodología	45
4.1.2	Cuestionario	45
4.1.3	Encuesta post-cuestionario	50
4.2	Validación con profesores	59
4.2.1	Resultados encuesta	59
CAPÍTULO 5: CONCLUSIONES		63
REFERENCIAS BIBLIOGRÁFICAS		65

ÍNDICE DE FIGURAS

1	Pregunta de control de la unidad de ciclos.	6
2	Pregunta de evaluación formativa de la unidad de <i>strings</i>	7
3	Pregunta de evaluación formativa de la unidad de funciones.	8
4	Interfaz de Scratch y sus partes.	10
5	Ejemplo pregunta Respuestas anidadas (Cloze).	11
6	Pregunta con respuesta correcta.	11
7	Pregunta con respuesta parcialmente correcta.	12
8	Ejemplo de pregunta usando el módulo <i>CodeRunner</i>	14
9	Ejemplo de pregunta usando el módulo <i>Ordering</i>	14
10	Ejemplo de pregunta usando el módulo <i>Gapfill</i>	15
11	Disposición de archivos típica para un <i>question type plugin</i> en Moodle.	21
12	Código correspondiente al control de la figura 1.	22
13	Código a ingresar para realizar la pregunta de la figura 1.	23
14	Código a ingresar usando <i>Gapfill</i> para realizar la pregunta de la figura 1.	24
15	Interfaz para agregar retroalimentación a respuesta ``intentos" usando <i>Gapfill</i>	24
16	Alternativas de respuesta usando <i>plugin Gapfill</i>	25
17	Esquema en la BBDD del tipo de pregunta <i>Gapfill</i>	26
18	Interfaz para agregar retroalimentación a respuesta ``1" usando <i>Gapfill</i> : ru- teo de código.	30
19	Pregunta de figura 3 usando <i>Gapfill</i> : ruteo de código.	31
20	Retroalimentación correcta de ruteo en figura 19.	32
21	Retroalimentación parcialmente correcta de ruteo en figura 19.	33
22	Esquema en la BBDD del tipo de pregunta problema de Parsons.	35
23	Menú de añadir pregunta a cuestionario	36
24	Formulario para agregar bloques VPD (Visually Paired Distractors) y distrac- tores.	37
25	Programa a transformar en problema de Parsons.	38
26	Interfaz de herramienta Parsonsificador.	39
27	Código transformado a problema de Parsons.	40
28	Interfaz para definir un delimitador de código.	40
29	Interfaz para unir bloques de código.	41
30	Interfaz luego de unir bloques de código.	42
31	Interfaz para definir delimitadores de VPD.	43
32	Nuevo bloque de código VPD.	43
33	Código luego de transformar bloque a VPD.	44
34	Problema de Parsons en cuestionario de Moodle.	44
35	Problema de Parsons a realizar usando lápiz y papel.	47
36	Ruteo a realizar usando lápiz y papel.	48
37	Problema de Parsons a realizar usando AULA.	49
38	Ruteo a realizar usando AULA.	50
39	Percepción de dificultad de los estudiantes resolviendo problemas de Par- sons en AULA en comparación a lápiz y papel.	51

40	Percepción de dificultad de los estudiantes resolviendo ruteos en AULA en comparación a lápiz y papel.	52
41	Percepción de dificultad de los estudiantes resolviendo problemas de Parsons en AULA.	53
42	Percepción de dificultad de los estudiantes resolviendo ruteos en AULA. . . .	53
43	Niveles de acuerdo de querer seguir aprendiendo programación usando problemas de Parsons en AULA.	54
44	Niveles de acuerdo de querer seguir aprendiendo programación usando ruteos en AULA.	55
45	Niveles de acuerdo de querer recibir retroalimentación inmediata luego de terminar un cuestionario.	56
46	Cuánto demoraron los estudiantes resolviendo problemas de Parsons en AULA con respecto a usando lápiz y papel.	57
47	Cuánto demoraron los estudiantes resolviendo ruteos en AULA con respecto a usando lápiz y papel.	58
48	Niveles de acuerdo respecto a utilidad para el aprendizaje de los estudiantes de la solución desarrollada.	60
49	Percepción de dificultad de los profesores al crear ejercicios habilitadores en AULA.	60
50	Percepción de dificultad de los profesores al crear problemas de Parsons en AULA sin usar Parsonsificador.	61

ÍNDICE DE TABLAS

1	Servidores de bases de datos y su versión mínima para Moodle 3.9 (LTS) . . .	20
2	Ruteo de código correspondiente a figura 3.	28
3	Ruteo de código correspondiente a figura 3 usando el tipo de pregunta Cloze. .	28
4	Ruteo de código correspondiente a figura 3 modificado para aceptar respuestas vacías.	29
5	Ruteo de código correspondiente a figura 3 modificado para minimizar impacto de respuestas vacías.	29
6	Tabla de ruteo de código correspondiente a figura 3 usando <i>Gapfill</i> : ruteo de código.	30

INTRODUCCIÓN

Aprender a programar es una actividad compleja, según [Robins *et al.*, 2003] mientras se aprende se debe lidiar con varios dominios al mismo tiempo como por ejemplo la orientación general, o sea para qué sirven los programas y qué se puede hacer con ellos; el modelo mental que se tiene sobre el computador mientras ejecuta un programa; la notación, es decir la sintaxis y semántica del lenguaje de programación; los esquemas y patrones que ayudan a resolver problemáticas comunes; planificar, desarrollar, probar, arreglar, etcétera.

[Luxton-Reilly, 2016] al contrario, menciona que aprender a programar es fácil, y aquello que hace que la frase “aprender a programar es difícil” sea considerada conocimiento general es el enfoque que se le da a los cursos introductorios de cubrir mucho contenido a un ritmo rápido, generando expectativas difíciles de alcanzar por los estudiantes lo que lleva altas tasas de deserción. Se indica sobre profesores comparando el aprendizaje en un curso introductorio de programación a escribir poesía en un idioma extranjero.

[Jenkins, 2002] menciona el estilo de aprendizaje como factor cognitivo que puede dificultar el aprendizaje de la programación, existiendo dos tipos, el aprendizaje profundo, en el que se promueve la comprensión y aplicación del tema, y el aprendizaje superficial, en el que el aprendiz mayoritariamente memoriza la información como hechos aislados. Esta clasificación de aprendizajes funciona bien para aprender por ejemplo sobre historia, que requiere aprendizaje superficial para memorizar fechas y después de eso aprendizaje profundo para análisis y comprensión, esto no funciona para aprender a programar, ya que los dos estilos de aprendizaje deben ser aplicados al mismo tiempo, el superficial para aprender patrones y notación, a la par del aprendizaje profundo para desarrollar y aplicar el conocimiento. [Robins *et al.*, 2003] alude como una de las razones que separa al programador novato del experto es el uso del aprendizaje profundo, mientras que los expertos lo aplican correctamente y a la par del superficial, los novatos se limitan al conocimiento superficial y fallan al aplicar conceptos relevantes. En una encuesta que [Petersen *et al.*, 2016] realiza sobre por qué los estudiantes abandonan los cursos introductorios de programación, los estudiantes mencionan que las técnicas de aprendizaje a las que están acostumbrados y usan en otros cursos no son tan efectivas al aprender a programar, y creen que se necesita un enfoque diferente que involucre más práctica.

Lo que lleva a estudiantes a tomar la compleja decisión de abandonar cursos introductorios de programación corresponde principalmente a factores como falta de tiempo y motivación, en combinación a factores secundarios, como darle prioridad a otros cursos, mal manejo del tiempo o la dificultad percibida del curso, lo que influencia la toma de la decisión [Kinnunen y Malmi, 2006]. [Petersen *et al.*, 2016] nombra como punto decisivo el momento en que el estudiante se da cuenta de que está o estará atrasado.

En Chile, la demanda por los estudiantes de matricularse en carreras de tecnología va en crecimiento, tanto así que las matrículas en instituciones de educación superior, en carreras

relacionadas a las Tecnologías de la Información y la Comunicación (TIC) entre los años 2007 y 2023, han tenido un crecimiento del 3,26 % anual en promedio [MIFUTURO, 2023a].

La retención de primer y segundo año de la carrera Ingeniería Civil en Computación e Informática para estudiantes matriculados el año 2021 en Chile corresponde al 82,7 % y 66,5 % respectivamente, esto quiere decir que el 33,5 % de los estudiantes abandonaron la carrera antes de alcanzar el tercer año [MIFUTURO, 2023b]. La tasa de fracaso promedio mundial el año 2017 en cursos introductorios de programación fue del 28 % [Bennedsen y Caspersen, 2019].

Introducir a los estudiantes a la programación sigue siendo un gran desafío, por lo que en la presente memoria se desarrollará una serie de tipos de ejercicios en la plataforma Moodle para cursos introductorios de programación que reduzcan la carga cognitiva a la que se enfrentan los estudiantes.

La estructura de este documento se divide en 5 capítulos. En el Capítulo 1: Definición del problema, se detalla el problema sobre el cual se trabajará, se describe la situación actual de un curso introductorio de programación de la Universidad Técnica Federico Santa María (UTFSM), dando énfasis al uso de la plataforma Moodle para crear cuestionarios en línea, se presentan soluciones existentes y los objetivos que motivan este trabajo. En el Capítulo 2: Marco Conceptual, se describen los conceptos en los que se basa esta memoria, dando a conocer sobre Moodle y sus módulos de tipo de pregunta, sobre la teoría de la carga cognitiva, sobre ejercicios habilitadores y el andamiaje. En el Capítulo 3: Propuesta de Solución, se muestra el desarrollo de cada nuevo tipo de pregunta desarrollado para Moodle, así como también el Parsonsificador. En el Capítulo 4: Validación de la Solución, se valida la solución propuesta en el contexto donde fue planteada, apoyándose de dos encuestas de percepción, una a estudiantes y otra a profesores. Para terminar, en el Capítulo 5: Conclusiones, se resume el aprendizaje final conseguido tras la realización de esta memoria.

CAPÍTULO 1

DEFINICIÓN DEL PROBLEMA

Los estudiantes de cursos introductorios de programación enfrentan una carga cognitiva considerable. Esto se debe a la intrínseca complejidad del tema, que generalmente requiere la práctica de la escritura de código, implicando la consideración de la sintaxis, la semántica y otros elementos [Robins *et al.*, 2003]. En consecuencia, se incrementa la carga cognitiva a la que deben hacer frente.

Es por casos como el descrito antes que se está en constante búsqueda de métodos que mejoren el aprendizaje y retención de la información en los estudiantes, dentro de lo que se ha estudiado se ha descubierto que el aprendizaje de la programación se ve favorecido cuando se enfrenta inicialmente a ejercicios habilitadores con andamiaje; Se trata de ejercicios en que los estudiantes no deben necesariamente escribir código, sino completar partes, ordenar e indentar correctamente un grupo de instrucciones desordenadas, predecir la salida que tendrá un programa o función, rutear código, etc. Este enfoque reduce la carga cognitiva a la que se enfrentan los estudiantes durante el aprendizaje [Van Merriënboer, 1990].

Actualmente en la plataforma Moodle no existen suficientes herramientas para evaluar estos tipos de ejercicios habilitadores, los tipos de pregunta que ofrece la plataforma son alternativa múltiple, verdadero o falso, arrastrar sobre texto, seleccionar palabras faltantes en frases, entre otras.

El impacto inicial de solucionar el problema es permitir a profesores que impartan cursos introductorios de programación, en cualquier lugar del mundo, hacer preguntas con tipos de ejercicios habilitadores que reducen la carga cognitiva del estudiante usando la plataforma Moodle.

1.1. Situación actual

La asignatura Programación (sigla IWI-131) es parte de los planes de estudio de todas las carreras de Ingeniería nivel Civil en la UTFSM, cuenta con 5 créditos SCT, estimándose 120 a 155 horas de trabajo real de un estudiante en un semestre para lograr los resultados de aprendizaje esperados. Para cumplir con lo anterior, la planificación de la asignatura contempla dos instancias de aprendizaje:

- **Clases de cátedra:** Se le dictan a los estudiantes dos veces a la semana. Durante la primera sesión, se imparten clases teóricas para introducir la unidad de aprendizaje, seguidas de clases prácticas en la segunda sesión para reforzar el contenido.
- **Ayudantías:** Se realiza una vez a la semana y es la instancia para llevar a cabo el

aprender-haciendo a través de laboratorios.

Y otras cinco formas de aprendizaje con evaluación:

- **Certámenes:** Las evaluaciones con mayor importancia en la calificación final de la asignatura, son dos cada semestre y se realizan usando lápiz y papel.
- **Controles:** Cuestionarios hechos a través de la plataforma AULA del curso, tienen una duración de 10 minutos y constan de una pregunta que no exige creación de código, sino completar y/o arreglar código existente, son 8 a 10 cada semestre.
- **Tareas:** Trabajo individual que consta en la generación de código para solucionar una problemática presentada, normalmente los estudiantes tienen 10 días para resolverlas, son 4 cada semestre.
- **Evaluaciones formativas:** Cuestionarios hechos a través de la plataforma AULA del curso, constan de preguntas que buscan validar si se entendieron los nuevos contenidos de la unidad, son 8 a 10 cada semestre.
- **Ejercicios de corrección automática:** Ejercicios basados en certámenes de años anteriores que consisten en generación de código para resolver problemáticas, se resuelven en la herramienta Santa María Online Judge (SMOJ)¹ que es externa a la plataforma AULA, son 8 a 10 cada semestre.

Como se mencionó anteriormente, los **Controles** y las **Evaluaciones formativas** se llevan a cabo mediante la plataforma **AULA** basada en Moodle. Al momento de la realización de este trabajo de título, la UTFSM utiliza la versión 3.9 (LTS) de esta plataforma. En una instalación estándar (sin *plugins* adicionales) de la plataforma Moodle en la versión usada por la UTFSM se encuentran los siguientes tipos de preguntas que se pueden incluir en un cuestionario:

- **Opción múltiple:** Permite la selección de una o múltiples respuestas de una lista predefinida.
- **Verdadero/Falso:** Una forma simple de pregunta de alternativa múltiple con las opciones Verdadero o Falso.
- **Emparejamiento:** La respuesta de cada pregunta debe ser seleccionada de un conjunto de respuestas.
- **Respuesta corta:** Permite una respuesta de pocas palabras que se califica comparándola con varias respuestas modelo.

¹<https://smoj.inf.utfsm.cl/>

- **Numérica:** Permite una respuesta numérica, que se califica comparándola con varias respuestas modelo, posiblemente con márgenes de tolerancia.
- **Ensayo:** Permite subir un archivo o texto como respuesta, la calificación de este tipo de preguntas es manual.
- **Calculada:** Es como un tipo de pregunta Numérica pero los números involucrados en la pregunta son seleccionados aleatoriamente de un conjunto cuando se toma el cuestionario.
- **Calculada opción múltiple:** Permite la selección de una o múltiples respuestas que pueden incluir resultados hechos con fórmulas a partir de números seleccionados aleatoriamente de un conjunto cuando se realiza el cuestionario.
- **Calculada simple:** Una versión más simple del tipo de pregunta Calculada.
- **Arrastrar y soltar sobre texto:** Permite arrastrar y soltar palabras sobre espacios en blanco en el texto de la pregunta.
- **Arrastrar y soltar marcadores:** Permite arrastrar y soltar marcadores sobre una imagen de fondo.
- **Arrastrar y soltar sobre una imagen:** Permite arrastrar y soltar imágenes o etiquetas de texto sobre zonas específicas en una imagen de fondo.
- **Respuestas anidadas (Cloze):** Permite la creación de una combinación de tipos de preguntas Opción múltiple, Respuesta corta y Numérica.
- **Emparejamiento de respuesta corta aleatoria:** Es como un tipo de pregunta Emparejamiento, pero extraídas aleatoriamente de un banco de preguntas creadas del tipo Respuesta corta.
- **Elige la palabra perdida:** Permite la selección de palabras que faltan en un texto desde un menú desplegable.

Un ejemplo de pregunta de control de la unidad 4: ciclos que usa el tipo de pregunta con respuestas anidadas (Cloze) para hacer un ejercicio habilitador de tipo completar código es el siguiente:

A continuación se muestra un programa que genera una cantidad n , indicada por el usuario, de números aleatorios entre el 1 y el 100. El programa se detiene cuando ha generado $n/2$ números pares o ha llegado al tope de n números en donde indicará que no pudo generar todos los números pares requeridos.

A continuación se presenta el código de un programa que hace lo indicado, pero está incompleto. Escriba las instrucciones faltantes en cada caso, para que el programa funcione correctamente.

```
from random import randint
n = int(input("Cantidad de números a generar: "))
listo = False
pares = 0
intentos = 0
while   n  listo == False:
    num = randint(1,100)
    print(num)
    if num % 2 == 0:
        pares += 1
    if pares == int(n/2):
        listo = 
    intentos += 1
if listo ==  :
    print("No se completo la cantidad de pares necesarios")
else:
    print("Se logró generar la cantidad de pares necesarios en", intentos, "intentos")
```

Figura 1: Pregunta de control de la unidad de ciclos.

Fuente: Control 4 de la asignatura IWI-131 de la UTFSM del primer semestre del año 2023.

En el ejercicio de la figura 1 los estudiantes deben rellenar cada espacio en blanco en el código con la instrucción correcta, normalmente éstas están escritas correctamente y las opciones ofrecidas dependen del código, por ejemplo si la instrucción correcta es el booleano *True*, entonces las opciones ofrecidas son *True* y *False*, o si la instrucción correcta es usar el signo “mayor qué” ($>$), entonces las opciones ofrecidas pueden ser los signos “mayor qué” ($>$), “menor qué” ($<$), “mayor o igual qué” ($>=$) y “menor o igual qué” ($<=$). También existe la posibilidad de ofrecer parte de la selección de instrucciones incorrectamente escritas y así evaluar el conocimiento del estudiante sobre la sintaxis del lenguaje.

Otro ejemplo de pregunta en que el estudiante no debe escribir código es la siguiente que corresponde a una evaluación formativa de la unidad 5: *strings*:

Contamos con un string con el siguiente formato: **CSTCSTCSTCST**.... del cual no conocemos el largo, pero sabemos que siempre tendrá esa forma. Aquí, **C** corresponde a una letra que indica el **Campamento** del cual proviene un trabajador, **S** es una letra que identifica el **sindicato** al que pertenece el trabajador y **T** es una letra que identifica al **tipo** de trabajador operario de la mina. El string ya se encuentra validado.

Se requiere crear un ciclo que permita recorrer dicho string, de tal forma de que **en cada iteración** se obtengan los datos de un trabajador, es decir, el campamento, sindicato y trabajador, y se muestren en la pantalla.

A continuación se presentan 3 opciones. Indique cuál es la correcta

Seleccione una:

- ☐ a.

```
s = "abcdefghijkl"
i = 0
for i in s:
    cam = s[i]
    sin = s[i+1]
    tra = s[i+2]
    print(cam,sin,tra)
```
- ☐ b.

```
s = "abcdefghijkl"
i = 0
while i<len(s):
    cam = s[i]
    sin = s[i+1]
    tra = s[i+2]
    i+=3
    print(cam,sin,tra)
```
- ☐ c.

```
s = "abcdefghijkl"
i = 0
while i<len(s)-2:
    cam = s[i]
    sin = s[i+1]
    tra = s[i+2]
    i+=3
    print(cam,sin,tra)
```

Comprobar

Figura 2: Pregunta de evaluación formativa de la unidad de *strings*.

Fuente: Evaluación formativa de la asignatura IWI-131 de la UTFSM del primer semestre del año 2023.

En el ejercicio de la figura 2 los estudiantes deben seleccionar la opción con el código que cumpla con lo requerido.

Un ejemplo de ruteo de código usando el tipo de pregunta respuestas anidadas (Cloze) es el siguiente:

Considere el siguiente programa que hace un llamado a una función:

```
def num_digitos(n):
    nd = 1
    while n>9:
        nd += 1
        n = n//10
    return nd

num = 53
resultado = num_digitos(num)
print(resultado)
```

Complete el ruteo de la ejecución del programa.

num	resultado	n	nd
53			
		53	
			2

Pantalla

Figura 3: Pregunta de evaluación formativa de la unidad de funciones.
Fuente: Evaluación formativa de la asignatura IWI-131 de la UTFSM del primer semestre del año 2023.

En esta pregunta el estudiante debe realizar un ruteo de código, es decir, seguir el flujo del programa y anotar el valor de las variables según el resultado de cada instrucción a medida que el programa se ejecuta línea por línea.

Si bien hay una gran cantidad de tipos de preguntas, para reducir la carga cognitiva de los estudiantes a la hora de aprender se necesitan nuevas formas de evaluación en la plataforma.

1.2. Dificultades del problema

Según los datos publicados en la sección de admisión de la página web de la UTFSM², entre los campus San Joaquín, Vitacura y Casa Central Valparaíso se abren alrededor de 1.800 vacantes para que estudiantes de carreras nivel Civil puedan ingresar. Como la asignatura Programación es parte de las mallas curriculares de todas las carreras nivel Civil de la universidad, se espera que cada año 1.800 estudiantes la cursen, lo que descarta cualquier plataforma que cobre licencia de uso por estudiante.

1.3. Soluciones existentes

Las soluciones alternativas que cumplen con el objetivo de poder ser usadas para enseñar programación reduciendo la carga cognitiva al no ser necesario escribir código se listan a continuación.

1.3.1. Lenguajes de programación basados en bloques

Los lenguajes de programación basados en bloques son un tipo de lenguaje de programación que permite a los estudiantes crear algoritmos utilizando bloques de código conectables entre sí en lugar de escribir líneas de código tradicionales. Estos bloques representan operaciones, funciones o acciones específicas y se pueden arrastrar y soltar en un lienzo o espacio de trabajo para construir la lógica del programa. Los lenguajes de programación basados en bloques son especialmente populares en la enseñanza de la programación a principiantes y a jóvenes estudiantes debido a su naturaleza visual y su facilidad de uso en comparación a lenguajes de programación basados en texto [Weintrop y Wilensky, 2015].

Al ser visuales y basados en bloques, hacen que la creación de programas sea una experiencia más interactiva e intuitiva ya que los bloques se encajan como piezas de un rompecabezas, esto a la vez elimina la necesidad de lidiar con sintaxis compleja que suele encontrarse en lenguajes de programación basados en texto ya que las reglas sintácticas se transfieren a reglas de conexión entre bloques. Al enfocarse en la lógica y el flujo del programa, en lugar de la sintaxis, los estudiantes pueden desarrollar habilidades sólidas de resolución de problemas y pensamiento lógico.

Dentro de los desafíos de la programación basada en bloques es que los estudiantes no perciben esta forma de programación como “programación de verdad”, y también existe una pregunta abierta sobre si las habilidades obtenidas tras el aprendizaje con herramientas basadas en bloque se transfieren a la programación basada en texto [Weintrop, 2019].

²<https://usm.cl/admision/admision-centralizada-pdt/>

Un ejemplo popular de lenguaje de programación basado en bloques es **Scratch**, cuyo objetivo es proveer un ambiente de programación con suficiente andamiaje tal que principiantes puedan empezar a programar con nulo o muy poco conocimiento y a la vez ser capaces de crear programas sofisticados [Weintrop, 2019].

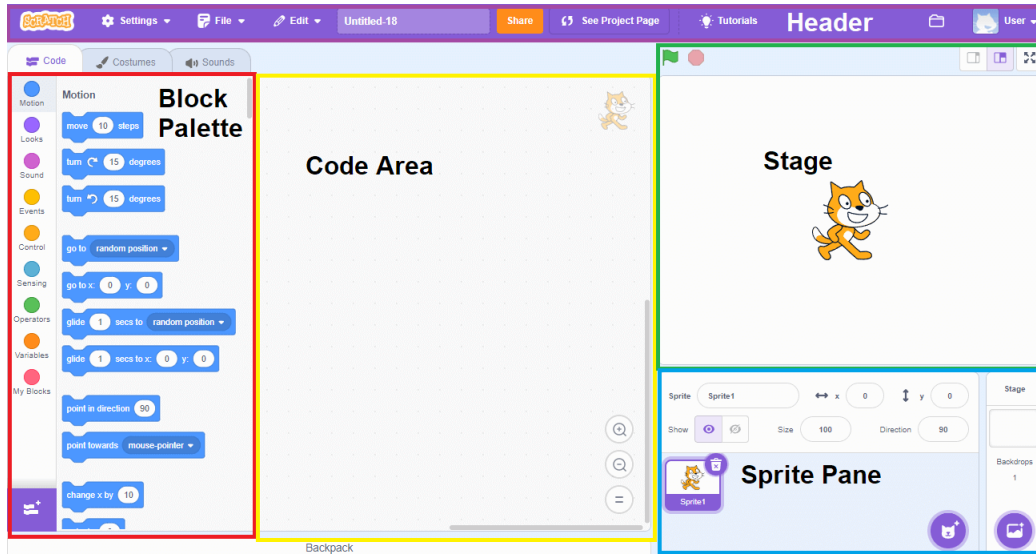


Figura 4: Interfaz de Scratch y sus partes.
Fuente: Wiki de Scratch.

1.3.2. Tipo de pregunta respuestas anidadas (Cloze)

Respuestas anidadas (Cloze) es un tipo de pregunta oficial de Moodle, es decir viene incluido en una instalación estándar, y consiste en un tipo de pregunta que se compone de una combinación de opción múltiple, respuesta corta y numérica. Para crear un tipo de pregunta respuestas anidadas (Cloze) se debe seguir la siguiente estructura:

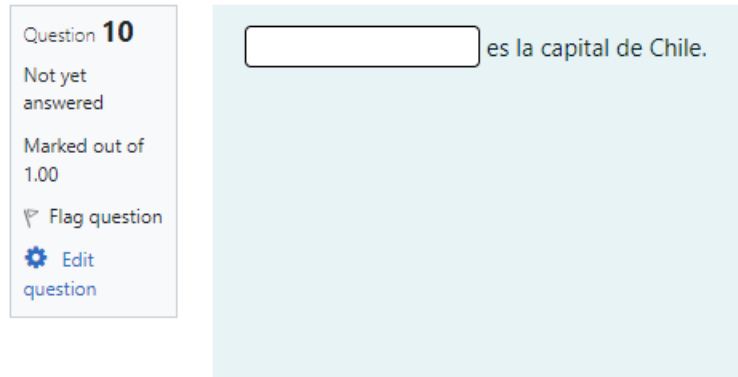
- Se comienza una sub-pregunta con un corchete de apertura (`{`).
- Opcionalmente se sigue con un número entero que representa el peso de la sub-pregunta sobre el total de la calificación de la pregunta.
- Luego se sigue con el tipo de pregunta encerrado por dos puntos, **:MULTICHOICE:**, **:SHORTANSWER:** o **:NUMERICAL:**. Existen más opciones que modifican la interfaz de la pregunta, por ejemplo si se desea una fila horizontal o vertical de *checkboxes*.
- Se debe definir cada posible respuesta con el símbolo `~` que actúa como separador de respuestas.
- Se puede definir el porcentaje del total del peso sobre la calificación que obtendría una respuesta encerrando un número entre símbolos de porcentaje (`%`).

- El símbolo de igualdad (=) se usa para identificar la respuesta correcta.
- Opcionalmente se puede agregar un símbolo # que se usa para entregar un mensaje de retroalimentación.
- Finalmente se cierra la sub-pregunta con un corchete de cierre (}).

Con este tipo de pregunta es posible hacer ejercicios habilitadores ya que se pueden crear preguntas de tipo completar código entregando las instrucciones correctamente escritas y así reducir la carga cognitiva que enfrenta el estudiante ya que no debe lidiar con la sintaxis del lenguaje de programación. Un ejemplo de sub-pregunta es el siguiente:

{1:SHORTANSWER:=Santiago#Correcto! %50 %Gran Concepción#No, esa es la segunda ciudad más grande en Chile.} es la capital de Chile

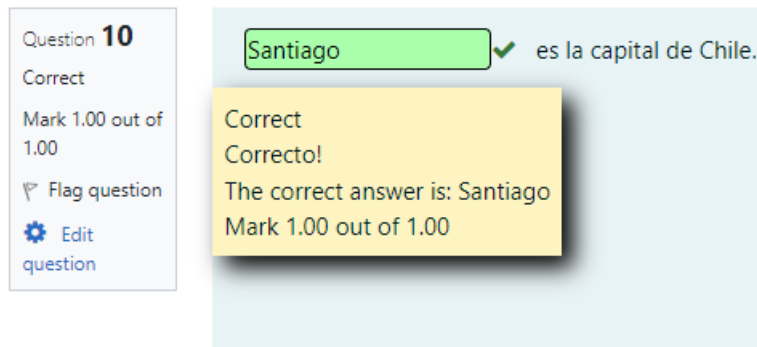
Se vería representada de la siguiente forma:



The screenshot shows a Moodle question interface. On the left, a sidebar contains the following information: 'Question 10', 'Not yet answered', 'Marked out of 1.00', a 'Flag question' button, and an 'Edit question' button. The main area of the question is a light blue box containing a text input field followed by the text 'es la capital de Chile.'

Figura 5: Ejemplo pregunta Respuestas anidadas (Cloze).
Fuente: Elaboración propia.

y la retroalimentación de cada respuesta se puede ver en las siguientes imágenes:



The screenshot shows the same Moodle question interface as Figure 5, but with the answer 'Santiago' entered in the input field. The input field is highlighted with a green border and a green checkmark icon to its right. A yellow feedback box is overlaid on the question area, containing the text: 'Correct', 'Correcto!', 'The correct answer is: Santiago', and 'Mark 1.00 out of 1.00'. The sidebar on the left now shows 'Correct' and 'Mark 1.00 out of 1.00'.

Figura 6: Pregunta con respuesta correcta.
Fuente: Elaboración propia.

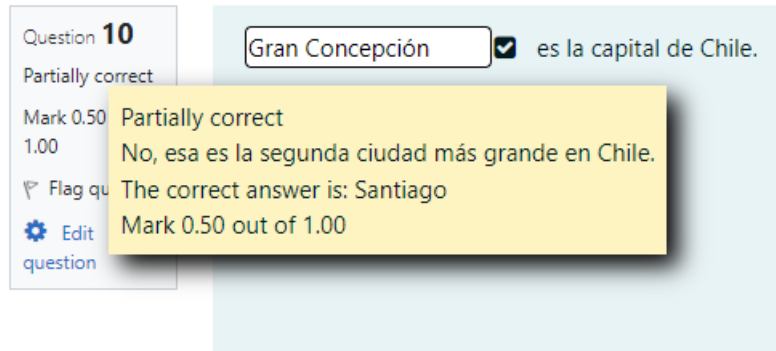


Figura 7: Pregunta con respuesta parcialmente correcta.
Fuente: Elaboración propia.

1.4. Objetivos

1.4.1. Objetivo general

- Implementar una serie de tipos de ejercicios habilitadores auto corregibles en Moodle que ayuden al proceso de aprendizaje del estudiante en cursos introductorios de programación.

1.4.2. Objetivos específicos

- Investigar tipos de ejercicios habilitadores que reducen la carga cognitiva necesaria para el aprendizaje.
- Seleccionar aquellos tipos de ejercicios habilitadores que puedan ser implementados en Moodle.
- Implementar los tipos de ejercicios habilitadores seleccionados de modo que sean auto corregibles.
- Evaluar los mecanismos implementados con estudiantes de la asignatura IWI-131 Programación de la UTFSM.

CAPÍTULO 2

MARCO CONCEPTUAL

2.1. Moodle

Moodle es una plataforma de aprendizaje diseñada para proporcionarle a educadores, administradores y estudiantes un sistema integrado único, robusto y seguro para crear y administrar ambientes de aprendizaje personalizados en línea. Moodle es un acrónimo de “*Modular Object-Oriented Dynamic Learning Environment*” (Entorno de Aprendizaje Dinámico Orientado a Objetos y Modular).

En Moodle los profesores pueden crear y organizar contenido educativo, interactuar con estudiantes, asignar tareas, administrar exámenes, facilitar discusiones en línea y realizar un seguimiento del progreso del estudiante. Ofrece herramientas y funcionalidades que promueven la interacción, el compromiso y la colaboración en el proceso de aprendizaje. Dada su naturaleza de código abierto, Moodle tiene una comunidad activa de desarrolladores y usuarios que contribuyen con mejoras a la plataforma, lo que ha contribuido a su popularidad y adopción en todo el mundo.

2.1.1. Arquitectura modular de Moodle

Moodle sigue una arquitectura de diseño modular, es decir separa la funcionalidad de la aplicación en módulos independientes que contienen lo justo y necesario para solucionar un aspecto de la funcionalidad final, la mayoría de las interacciones de los usuarios en Moodle son sobre estos módulos, más comúnmente llamados *plugins*.

2.1.2. Módulos de tipo de pregunta

Los módulos de tipo de pregunta extienden la funcionalidad de los cuestionarios de Moodle al proveer nuevas opciones para agregar. Al momento de realización de este trabajo de título, alrededor de 40 tipos de pregunta han sido publicados en la página web oficial de Moodle para su versión 3.9 LTS. Dentro de los tipos de pregunta más descargados se encuentran:

- **CodeRunner** es un tipo de pregunta en que los estudiantes responden escribiendo código, generalmente una función, y luego se califica ejecutando la respuesta y evaluando la similitud entre los resultados obtenidos y casos de prueba predefinidos. Según los autores esta extensión ha sido usada por más de siete años por la Universidad de

Canterbury en Nueva Zelanda, y soporta los lenguajes de programación Python, C, JavaScript, PHP, Octave y Matlab.

Write a function `date_string(day_num, month_name, year_num)` that returns a string in the format `"day month, year"`. See the examples below for more information.

For example:

Test	Result
<code>print(date_string(1, "December", 1984))</code>	1 December, 1984
<code>print(date_string(29, "February", 2020))</code>	29 February, 2020

Answer: (penalty regime: 10, 20, ... %)

```
1 def date_string(day_num, month_name, year_num):
2     """Returns a string in the format 'day month, year' """
3     return "{} {}, {}".format(day_num, month_name, year_num)
```

Precheck Check

Pre-check only

Test	Expected	Got
✓ <code>print(date_string(1, "December", 1984))</code>	1 December, 1984	1 December, 1984 ✓
✓ <code>print(date_string(29, "February", 2020))</code>	29 February, 2020	29 February, 2020 ✓

Figura 8: Ejemplo de pregunta usando el módulo *CodeRunner*.

Fuente: Figura presente en el sitio web Moodle

https://moodle.org/plugins/qtype_coderunner.

- **Ordering** es un tipo de pregunta que muestra varios bloques de texto en un orden aleatorio que deben ser arrastrados a un orden secuencialmente correcto.

Question 1
Not yet answered
Marked out of 1.00

Put the following events into the order that they happened.

The kids saw a man with a wristwatch climb out of a grave.

The kids made a fire.

The kids went hiking.

The kids found footprints.

Byron gave Ruth Rose

Lucy took the kids to the hotel.

Jack's wife called him.

Myrna gave Josh a whistle.

Figura 9: Ejemplo de pregunta usando el módulo *Ordering*.

Fuente: Figura presente en el sitio web Moodle

https://moodle.org/plugins/qtype_ordering.

- **Gapfill** es un tipo de pregunta de relleno de espacios en blanco con una sintaxis de creación de preguntas muy simple, permite hacer preguntas con respuestas de arrastrar y soltar, *dropdowns* o relleno de espacios en blanco.

Feline	Canine

The sat on the

Figura 10: Ejemplo de pregunta usando el módulo *Gapfill*.

Fuente: Figura presente en el sitio web Moodle https://moodle.org/plugins/qtype_gapfill.

2.2. Teoría de la carga cognitiva

La teoría de la carga cognitiva (TCC) fue propuesta por primera vez por [Sweller, 1988] y puede ser definida como un marco teórico que explica cómo el aprendizaje se ve afectado por las limitaciones de la arquitectura cognitiva humana, siendo fundamental la relación entre la memoria de trabajo o memoria a corto plazo, la memoria a largo plazo y sus características [Duran *et al.*, 2022].

La TCC postula que la memoria de largo plazo tiene un almacenamiento prácticamente ilimitado, al contrario de la memoria de trabajo cuyo almacenamiento es limitado y de corta duración. Para lograr el aprendizaje la información debe ser procesada por la memoria de trabajo para una posterior construcción de conocimiento en la memoria de largo plazo.

Según la TCC, la carga cognitiva se refiere a la demanda sobre los recursos disponibles de la memoria de trabajo a la hora de realizar una tarea determinada, esta se puede dividir en dos tipos [Sweller, 2010]:

- **Carga cognitiva intrínseca:** Se trata de la complejidad de los conceptos o información que se deben aprender o procesar. Dada una tarea particular y una persona con cierto nivel de conocimiento, la carga cognitiva intrínseca es fija y no puede ser cambiada de otra forma que no sea cambiando la tarea o los niveles de conocimiento de la persona.
- **Carga cognitiva extrínseca:** Se refiere a la forma en que se presenta la información. Esta carga sí puede ser reducida y la cantidad de carga cognitiva extrínseca a la que se enfrenta una persona depende de la efectividad del diseño de la instrucción. Según [Sorva, 2012] algunos ejemplos de factores que generan carga cognitiva extrínseca son: información innecesaria en los materiales de aprendizaje, contenido de difícil acceso mientras se resuelve un problema, ruido de fondo molesto, instrucciones redundantes que deben escanearse en la búsqueda de contenido nuevo.

También se habla de la **Carga cognitiva relevante**, que no se considera como fuente de carga cognitiva, sino que se refiere a los recursos de la memoria de trabajo que se dedican para

lidar con la carga cognitiva intrínseca asociada a la información. Por lo que si durante el aprendizaje se tiene una carga cognitiva extrínseca baja, existe más espacio en la memoria de trabajo para dedicar a la carga cognitiva relevante y así mejorar el aprendizaje. Mientras que si se aumenta la carga cognitiva extrínseca, hay menos espacio en la memoria de trabajo para que la carga cognitiva relevante gestione la carga cognitiva intrínseca y el aprendizaje es menos eficiente ya que el estudiante dedica más recursos de la memoria de trabajo para lidiar con elementos extrínsecos.

2.2.1. Teoría de la carga cognitiva en la investigación de la educación computacional

[Duran *et al.*, 2022] realiza una revisión de mapeo sistemática sobre la TCC en cuatro revistas y seis conferencias que publican sobre la investigación en educación computacional y tras el análisis de 299 publicaciones se reporta un crecimiento en el uso de la palabra «carga cognitiva» en publicaciones, desde 17 artículos el año 2010 a 48 artículos el año 2020, casi triplicando su valor en una década y siendo a menudo utilizada como una justificación para decisiones de diseño como lo son por ejemplo problemas de Parsons, *worked examples*, entre otros.

Si bien se menciona que no es común que se profundice sobre la TCC, sí existen artículos que lo hacen como por ejemplo [Ericson *et al.*, 2017] que expresa hipótesis en términos de la TCC y compara resultados de aprendizaje, percepción de carga cognitiva, entre otros, de dos formas de práctica, usando problemas de Parsons y escribiendo código.

2.3. Ejercicios habilitadores

Los ejercicios habilitadores son tipos de ejercicios en los que no necesariamente se debe escribir código, sino ordenar o completar código correctamente escrito, así restringiendo las posibles soluciones y reduciendo la carga cognitiva extrínseca ya que se debe considerar en menor medida la sintaxis del lenguaje de programación.

2.3.1. Problemas de Parsons

Los problemas de Parsons fueron introducidos por [Parsons y Haden, 2006] como un nuevo tipo de ejercicio de programación con la motivación de hacer menos tediosa la práctica cuando se está aprendiendo a programar. En los problemas de Parsons los estudiantes deben posicionar en el orden correcto bloques de código separados y desordenados provenientes de un programa correctamente escrito. Existen diferentes variantes de los problemas de Parsons:

- **Problemas de Parsons:** es el más básico y los estudiantes sólo deben ordenar un programa que no incluye indentaciones.
- **Problemas bidimensionales de Parsons:** en que los estudiantes también deben indentar correctamente los bloques de código.
- **Problemas de Parsons con distractores:** en que no sólo se provee un programa correctamente escrito, también se debe identificar qué bloques de código están sintácticamente mal escritos o no pertenecen al programa final.
- **Problemas de Parsons con distractores emparejados:** en este tipo de problema de Parsons existe un nuevo bloque que contiene varios bloques de código y el estudiante debe seleccionar el correcto.

En [Ericson *et al.*, 2017] se compara la eficiencia, efectividad y la carga cognitiva del aprendizaje entre resolver problemas de Parsons bidimensionales con distractores, versus arreglar código con los mismos distractores, versus escribir el código equivalente a través de dos sesiones. En la primera sesión se separaron a 106 estudiantes en tres grupos, se les entrega material de estudio y luego se les realiza un *test* según el tipo de problema del grupo al que pertenecen, una semana después en la segunda sesión se les realiza un *test* isomórfico³ al primero para evaluar la retención del conocimiento.

Los resultados de la comparación muestran que los estudiantes en el grupo de problema de Parsons toman significativamente menos tiempo en completar el *test* en comparación a los otros dos grupos, que toman un tiempo parecido, mientras que no hay una diferencia significativa entre el desempeño de los tres grupos, ni tampoco en la encuesta de carga cognitiva. Concluyendo que usar problemas de Parsons bidimensionales con distractores es una forma de aprendizaje más eficiente, pero igual de efectiva en comparación a arreglar o escribir código.

2.3.2. Ruteo de código

Los ejercicios de ruteo de código son un tipo de ejercicio de programación en que los estudiantes deben ejecutar mentalmente cada paso de un programa y ser capaces de escribir el valor de todas las variables en cada paso y/o predecir el resultado de un programa.

[Kumar, 2013] reconoce la facilidad de calificar problemas de ruteo de código ya que su metodología es automatizable usando preguntas de alternativa múltiple, por ende busca conocer si resolver problemas de ruteo de código ayudan a mejorar las habilidades de escritura de código. Tras un análisis de desempeño usando sentencias de control *if-else* entre dos *tests*

³Se denomina a un *test* isomórfico a otro cuando sus problemas tienen la misma estructura pero diferentes características superficiales como por ejemplo el nombre de las variables

de escritura de código, con una serie de problemas de ruteo de código entremedio, distingue una mejoría en los resultados y la sintaxis de los *tests post-ruteos* de código.

Usando una metodología similar de evaluación, pero esta vez determinando el rendimiento de los estudiantes en la escritura de *loops* usando la sentencia *for*, [Kumar, 2015] descubre una mejoría en la semántica de los estudiantes que lograron una calificación mayor o igual a 90 % en los problemas de ruteo de código, asociándolo a un mejor modelo mental de programación en comparación a los otros estudiantes, pudiendo obtener aprendizaje en escritura de código de forma pasiva en la lectura de código correctamente escrito.

Combinando los resultados de los estudios anteriores, se considera el ruteo de código como un tipo de ejercicio que ayuda a mejorar los componentes de semántica y de sintaxis en las habilidades de escritura de código de los estudiantes.

2.3.3. Completar partes

Los ejercicios de completar partes son un tipo de ejercicio de programación en el que se proporciona a los estudiantes un fragmento de código en el que algunas partes han sido omitidas. Los estudiantes deben completar las partes faltantes del código para que funcione correctamente y cumpla con los requisitos establecidos en el enunciado del problema. De esta forma los estudiantes pueden practicar la sintaxis del lenguaje de programación, entender cómo se aplican ciertos conceptos y ganar experiencia en la resolución de problemas a nivel de código.

Este tipo de ejercicio tiende a tener una menor carga cognitiva en comparación a escribir código desde cero ya que gran parte del contexto del programa ya está dado, lo que ayuda a comprender mejor el propósito y la estructura general del código, de esta manera el estudiante puede centrarse en las áreas críticas del código que necesitan ser completadas, así olvidándose de la preocupación respecto a la sintaxis del resto del código.

2.4. Andamiaje

El término de andamiaje en la enseñanza fue introducido por [Wood *et al.*, 1976] definiéndolo como el “proceso que permite a un niño o principiante resolver un problema, realizar una tarea o lograr una meta que estaría más allá de sus esfuerzos sin ayuda.” Siendo el andamiaje el control por parte de un adulto de los elementos que están más allá de la capacidad del aprendiz, permitiéndole concentrarse y completar sólo los elementos que están en el rango de su capacidad.

La Zona de Desarrollo Próximo (ZDP) de Vygotsky ha sido definida como “la distancia entre el nivel de desarrollo real determinado mediante la resolución independiente de problemas y el nivel de desarrollo potencial determinado mediante la resolución de problemas bajo la

guía de un adulto o en colaboración con compañeros más capaces.” [Vygotsky y Cole, 1978]. Se considera que la noción de la ZDP está en el centro del concepto de andamiaje [Verenikina, 2003].

Un aspecto importante de la enseñanza con andamiajes es que el andamiaje es temporal, a medida que el aprendiz adquiere más habilidades y conocimientos que individualmente no podría haber logrado, el andamiaje se ajusta y disminuye gradualmente para permitir una mayor autonomía y desarrollo. El objetivo final es fomentar la independencia y la capacidad de resolver problemas por sí mismos, lo que les permite a los aprendices alcanzar su máximo potencial dentro de su ZDP [Van Der Stuyf, 2002].

El andamiaje es aplicado de diferentes formas para mejorar la enseñanza de la programación, en [Xie *et al.*, 2019] se logran buenos resultados al crear y aplicar un currículo basado en el andamiaje de conocimientos para un curso introductorio de programación, apoyando una transición gradual de habilidades de bajo nivel, como por ejemplo el ruteo de código, a habilidades de escritura de código de alto nivel. En [Margulieux *et al.*, 2019] se identifican cinco conceptos básicos de la programación y se diseña un currículo junto a ejercicios de ejemplo con etiquetas que indican cómo se emplean esos conceptos básicos que sirven de andamiaje mientras son usados durante el desarrollo del curso. En [Ericson *et al.*, 2017] y [Kumar, 2015] se utilizan ejercicios habilitadores, problemas de Parsons y ruteos de código respectivamente, como andamiaje para obtener resultados más eficientes en el aprendizaje de los estudiantes.

CAPÍTULO 3

PROPUESTA DE SOLUCIÓN

Al momento de realización de este trabajo de título, la UTFSM usa Moodle en su versión 3.9 (LTS) para crear la plataforma Aula USM⁴, donde se administra la mayoría de los cursos ofrecidos por la universidad cada semestre. Para desarrollar sobre la plataforma Moodle es necesario levantar un entorno de desarrollo local con las siguientes características:

- PHP versión mínima 7.2.0, también son compatibles las versiones 7.3.x ó 7.4.x.
- Apache versión 2.0.
- Javascript ES6.

y uno de los siguientes servidores de bases de datos:

Tabla 1: Servidores de bases de datos y su versión mínima para Moodle 3.9 (LTS)
Fuente: https://docs.moodle.org/dev/Moodle_3.9_release_notes.

Base de datos	Versión mínima	Versión recomendada
PostgreSQL	9.5	Más reciente
MySQL	5.6	Más reciente
MariaDB	10.2.29	Más reciente
Microsoft SQL Server	2012	Más reciente
Oracle Database	11.2	Más reciente

Una disposición de archivos típica y básica de un tipo de pregunta llamado questionName que se aloja en el directorio moodle/question/type/questionName es la siguiente:

⁴aula.usm.cl

```
moodle/question/type/questionName/
├── amd/
│   ├── build/
│   │   ├── .min.js file
│   │   └── .min.js.map file
│   └── src/
│       └── .js file
├── classes/
│   └── output/
│       └── renderer.php
├── db/
│   └── install.xml
├── lang/
│   └── en/
│       └── qtype_questionName.php
├── pix/
│   └── icon.svg
├── edit_questionName_form.php
├── question.php
├── questiontype.php
├── styles.css
└── version.php
```

Figura 11: Disposición de archivos típica para un *question type plugin* en Moodle.

Fuente: Elaboración propia

Donde:

- La carpeta **amd/** aloja todos los archivos JavaScript que necesite el tipo de pregunta.
- El archivo **classes/output/renderer.php** contiene toda la lógica correspondiente a cómo se muestra la pregunta en los cuestionarios, ya sea cuando el estudiante interactúe con ella y pueda responder, como también cuando el cuestionario haya sido enviado y se desee mostrar la retroalimentación.
- El archivo **db/install.xml** contiene el esquema de las tablas creadas específicamente para este tipo de pregunta en la base de datos si es necesario.
- La carpeta **lang/** contiene los textos que se muestran al utilizar el tipo de pregunta, separados en carpetas según el idioma.
- El archivo **pix/icon.svg** representa el ícono del tipo de pregunta.
- El archivo **edit_questionName_form.php** contiene toda la lógica correspondiente al formulario que se muestra a la hora de crear una pregunta de este tipo.

- El archivo **question.php** contiene toda la lógica correspondiente a cada instancia del tipo de pregunta, en este archivo se define cuál es el estado inicial de la pregunta según cada intento y cómo calificarla.
- El archivo **questiontype.php** contiene toda la lógica correspondiente al tipo de pregunta, en este archivo se define cómo se guardan a la base de datos la información ingresada por el usuario en el formulario a la hora de crear una pregunta de este tipo.
- El archivo **styles.css** contiene toda la lógica correspondiente a la presentación del contenido de la página específicos al tipo de pregunta.
- El archivo **version.php** contiene la información correspondiente a la versión del tipo de pregunta.

3.1. Tipo de pregunta: Completar partes

Actualmente se hacen preguntas de tipo completar partes usando el tipo de pregunta respuestas anidadas (Cloze) en Moodle. Para demostrar un ejemplo de cómo se hace esta pregunta se tendrá el siguiente código correspondiente al control 4 de la unidad de ciclos visto en la figura 1:

```
1  from random import randint
2  n = int(input("Cantidad de números a generar: "))
3  listo = False
4  pares = 0
5  intentos = 0
6  while intentos < n and listo == False:
7      num = randint(1,100)
8      print(num)
9      if num % 2 == 0:
10         pares += 1
11     if pares == int(n/2):
12         listo = True
13     intentos += 1
14 if listo == False:
15     print("No se completo la cantidad de pares necesarios")
16 else:
17     print("Se logró generar la cantidad de pares necesarios en", intentos, "intentos")
```

Figura 12: Código correspondiente al control de la figura 1.

Fuente: Control 4 de la asignatura IWI-131 de la UTFSM del primer semestre del año 2023.

Para lograr que el estudiante deba rellenar las mismas sentencias que en el ejemplo de la figura 1, el código ingresado al formulario de creación de pregunta en Moodle debe ser el siguiente:

```
1 from random import randint
2 n = int(input("Cantidad de números a generar: "))
3 listo = False
4 pares = 0
5 intentos = 0
6 while {1:MCS:=intentos~listo} {1:MCS:=<~>~<~>=} n {1:MCS:=and~or} listo == False:
7     num = randint(1,100)
8     print(num)
9     if num % 2 == 0:
10         pares += 1
11     if pares == int(n/2):
12         listo = {1:MCS:=True~False}
13         intentos += 1
14 if listo == {1:MCS:=False~True}:
15     print("No se completo la cantidad de pares necesarios")
16 else:
17     print("Se logró generar la cantidad de pares necesarios en", intentos, "intentos")
```

Figura 13: Código a ingresar para realizar la pregunta de la figura 1.

Fuente: Control 4 de la asignatura IWI-131 de la UTFSM del primer semestre del año 2023.

Notar los cambios hechos en las líneas 6, 12 y 14, que según lo visto en la sintaxis del tipo de pregunta respuestas anidadas (Cloze) en la sección "Soluciones existentes" incluye cinco campos de selección múltiple en un menú *dropdown*.

Una alternativa que si bien simplifica la creación de este tipo de preguntas, pero tiene menos características (como no poder determinar exactamente qué alternativas van en una selección, sino que sus opciones se escogen a partir de todas las respuestas y los distractores) es usar el *plugin Gapfill*⁵, que no es un tipo de pregunta incluida en una instalación de Moodle estándar sino que es creado y mantenido por usuarios externos a Moodle. Al usar este tipo de pregunta se tienen las siguientes ventajas:

- Sintaxis más simple para la creación de las preguntas.
- Proceso más intuitivo para agregar retroalimentación personalizada a cada respuesta.

El código que se debe ingresar al formulario en Moodle para la misma pregunta anterior, pero esta vez usando el tipo de pregunta *Gapfill* luce de la siguiente manera:

⁵https://moodle.org/plugins/qtype_gapfill

```
1 from random import randint
2 n = int(input("Cantidad de números a generar: "))
3 listo = False
4 pares = 0
5 intentos = 0
6 while @intentos@ @<@ n @and@ listo == False:
7     num = randint(1,100)
8     print(num)
9     if num % 2 == 0:
10         pares += 1
11     if pares == int(n/2):
12         listo = @True@
13     intentos += 1
14 if listo == @False@:
15     print("No se completo la cantidad de pares necesarios")
16 else:
17     print("Se logró generar la cantidad de pares necesarios en", intentos, "intentos")
```

Figura 14: Código a ingresar usando *Gapfill* para realizar la pregunta de la figura 1.
Fuente: Elaboración propia.

Con los siguientes distractores: listo, >, <=, >=, or, False, True.

Para agregar retroalimentación a cada respuesta, en el formulario de creación de pregunta se presiona el botón *Gap settings* y al hacer click sobre una de las respuestas aparece el menú de la figura 15:

The screenshot shows the Moodle question editor interface. The main text area contains the following code with gap markers: `programa funcione correctamente.`
`from random import randint`
`n = int(input("Cantidad de números a generar: "))`
`listo = False`
`pares = 0`
`intentos = 0`
`while @intentos@ @<@ n @and@ listo == False:`
 `num = randint(1,100)`
 `print(num)`
 `if num % 2 == 0:`
 `pares += 1`
 `if pares == int(n/2):`
 `listo = @True@`
 `intentos += 1`
`if listo == @False@:`
 `print("No se completo la cantidad de pares necesarios")`
`else:`
 `print("Se logró generar la cantidad de pares necesarios en", intentos, "intentos")`

The 'Distractors' field contains: listo, >, <=, >=, or, False, True

The 'Penalty factor' is set to 0.00

The 'Add Gap settings: intentos' dialog box is open, showing two sections: 'Feedback for correct.' and 'Feedback for incorrect.', each with a rich text editor. The 'Feedback for correct.' section is currently empty. The 'Feedback for incorrect.' section is also empty. The dialog box has an 'OK' button at the bottom right.

Figura 15: Interfaz para agregar retroalimentación a respuesta "intentos" usando *Gapfill*
Fuente: Elaboración propia.

Las desventajas de usar el tipo de pregunta *Gapfill* sobre el tipo de pregunta respuestas anidadas (Cloze) son:

- Como se ve en la figura 16 las alternativas que ofrece *Gapfill* en una sub-pregunta de selección múltiple con interfaz *dropdown* son las respuestas de todas las sub-preguntas y además los distractores que el usuario puede agregar, en cambio, usando el tipo de pregunta respuestas anidadas (Cloze) se puede definir cuáles son las alternativas incluidas en la selección de cada sub-pregunta.
- Es posible definir el porcentaje de impacto de cada sub-pregunta sobre la calificación total de la pregunta usando respuestas anidadas (Cloze), mientras que usando *Gapfill* cada sub-pregunta equivale lo mismo sobre la calificación total.

A continuación se muestra un programa que genera una cantidad n , indicada por el usuario, de números aleatorios entre el 1 y el 100. El programa se detiene cuando ha generado $n/2$ números pares o ha llegado al tope de n números en donde indicará que no pudo generar todos los números pares requeridos.

A continuación se presenta el código de un programa que hace lo indicado, pero está incompleto. Escriba las instrucciones faltantes en cada caso, para que el programa funcione correctamente.

```
from random import randint
n = int(input("Cantidad de números a generar: "))
listo = False
pares = 0
intentos = 0
while   n  listo == False:
    num = randint(1,100)
    print(num)
    if num % 2 == 0:
        pares += 1
    if pares == int(n/2):
        listo = 
        intentos += 1
    if listo ==  :
        print("No se completo la cantidad de números")
    else:
        print("Se logró generar la cantidad de números en", intentos, "intentos")
```

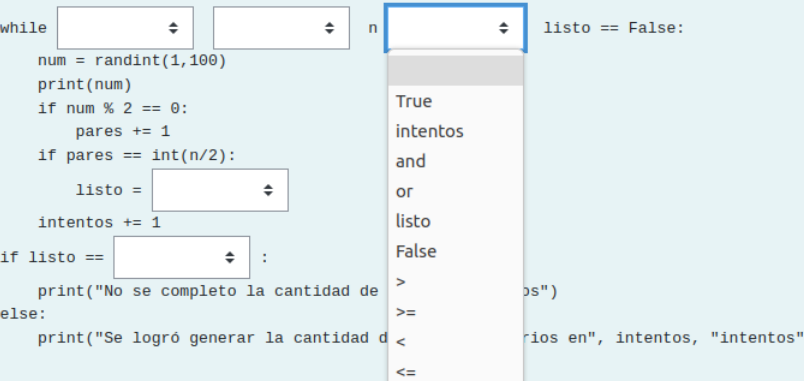


Figura 16: Alternativas de respuesta usando *plugin Gapfill*.

Fuente: Elaboración propia.

La tabla en la base de datos del tipo de pregunta *Gapfill* se ve de la siguiente manera:

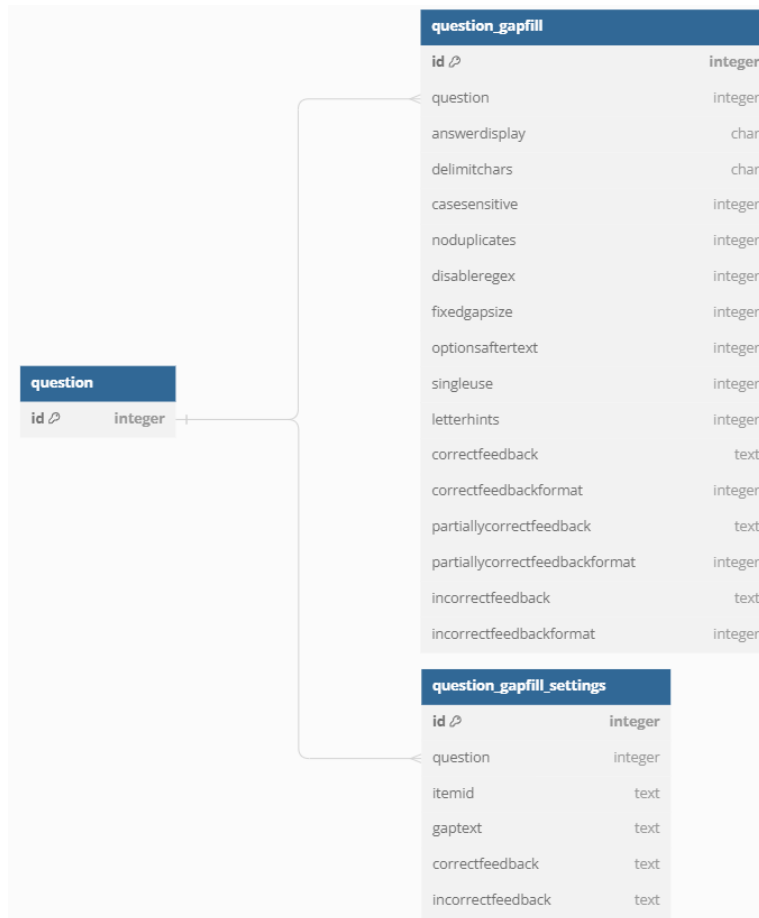


Figura 17: Esquema en la BBDD del tipo de pregunta *Gapfill*.
Fuente: Elaboración propia en herramienta dbdiagram.io.

Donde se ha simplificado la tabla *question* que es la pregunta base de Moodle y sólo se ha indicado su llave primaria. En las tablas que soportan el *plugin Gapfill* cada campo corresponde a:

Para la tabla *question_gapfill*

- **id:** Identificador de la pregunta de tipo *Gapfill*.
- **question:** Llave foránea que referencia a la pregunta base *question*.
- **answerdisplay:** Toma los valores 1 ó 0 e indica si las respuestas deben mostrarse con un orden particular o no respectivamente.
- **delimitchars:** Los caracteres que delimitan los campos que serán completados por el usuario. Su valor predeterminado es '['].

- **casesensitive:** Toma los valores 1 ó 0 e indica si las respuestas deben ser sensibles a mayúsculas o no respectivamente.
- **noduplicates:** Toma los valores 1 ó 0 e indica si se descartan respuestas duplicadas o no respectivamente, es útil cuando más de un campo acepta la misma respuesta.
- **disableregex:** Toma los valores 1 ó 0 e indica si se deshabilita el procesamiento de expresiones regulares o no respectivamente.
- **fixedgapsize:** Toma los valores 1 ó 0 e indica si todos los campos en blanco se muestran con el mismo tamaño o no respectivamente.
- **optionsaftertext:** Toma los valores 1 ó 0 e indica si las opciones se muestran encima o bajo del texto respectivamente.
- **singleuse:** Toma los valores 1 ó 0 e indica si los bloques arrastrables se quitan de la lista cuando sean soltados en un espacio o no respectivamente.
- **letterhints:** Toma los valores 1 ó 0 e indica si en el modo interactivo se van agregando letras de la respuesta correcta cuando se da una respuesta equivocada.
- **correctfeedback:** Retroalimentación correcta.
- **correctfeedbackformat:** El formato en que se muestra la retroalimentación correcta.
- **partiallycorrectfeedback:** Retroalimentación parcialmente correcta.
- **partiallycorrectfeedbackformat:** El formato en que se muestra la retroalimentación parcialmente correcta.
- **incorrectfeedback:** Retroalimentación incorrecta.
- **incorrectfeedbackformat:** El formato en que se muestra la retroalimentación incorrecta.

Y para la tabla *question_gapfill_settings*

- **id:** Identificador de la retroalimentación personalizada de una respuesta.
- **question:** Llave foránea que referencia a la pregunta base *question*.
- **itemid:** Identificador correspondiente a la respuesta.
- **gaptext:** Respuesta incluyendo a los delimitadores.
- **correctfeedback:** Retroalimentación a mostrar cuando la respuesta entregada por el estudiante es correcta.
- **incorrectfeedback:** Retroalimentación a mostrar cuando la respuesta entregada por el estudiante es incorrecta.

3.2. Tipo de pregunta: ruteo de código

Actualmente se hacen preguntas de ruteo de código usando el tipo de pregunta respuestas anidadas (Cloze). Para demostrar un ejemplo de cómo se hace esta pregunta se tendrá la siguiente tabla correspondiente al ruteo de código de la evaluación formativa de la unidad de funciones vista en la figura 3:

Tabla 2: Ruteo de código correspondiente a figura 3.

Fuente: Elaboración propia.

num	resultado	n	nd
53			
		53	
			1
			2
		5	
	2		

Para lograr que el estudiante deba rellenar las mismas sentencias que en el ejemplo de la figura 1, la tabla ingresada en el formulario de creación usando el tipo de pregunta respuestas anidadas (Cloze) debe ser la siguiente:

Tabla 3: Ruteo de código correspondiente a figura 3 usando el tipo de pregunta Cloze.

Fuente: Elaboración propia.

num	resultado	n	nd
53			
		53	
			{:NM:=1}
			2
		{:NM:=5}	
	{:NM:=2}		

En el caso que se desee completar la tabla con espacios en blanco, para que el estudiante pueda responder sobre cualquier lugar en la tabla, se debe definir qué respuesta constituye una respuesta vacía, ya que el tipo de pregunta respuestas anidadas (Cloze) no las permite debido a que se generaría un error de sintaxis. Si en esta pregunta se definiera el carácter guión (" - ") como respuesta vacía, la tabla ingresada en el formulario de creación debe ser la siguiente:

Tabla 4: Ruteo de código correspondiente a figura 3 modificado para aceptar respuestas vacías.

Fuente: Elaboración propia.

num	resultado	n	nd
53	{:SA:=-}	{:SA:=-}	{:SA:=-}
{:SA:=-}	{:SA:=-}	53	{:SA:=-}
{:SA:=-}	{:SA:=-}	{:SA:=-}	{:NM:=1}
{:SA:=-}	{:SA:=-}	{:SA:=-}	2
{:SA:=-}	{:SA:=-}	{:NM:=5}	{:SA:=-}
{:SA:=-}	{:NM:=2}	{:SA:=-}	{:SA:=-}

Debido a que el tipo de pregunta respuestas anidadas (Cloze) no está diseñado para aceptar respuestas vacías, una idea para lograr minimizar el impacto sobre la calificación de cada espacio que debe permanecer en blanco (o sea con el carácter guión) es otorgar un número bajo al peso que tienen sobre la calificación y un número muy alto al peso de las sub-preguntas que no deban permanecer en blanco, como en la siguiente tabla:

Tabla 5: Ruteo de código correspondiente a figura 3 modificado para minimizar impacto de respuestas vacías.

Fuente: Elaboración propia.

num	resultado	n	nd
53	{:SA:=-}	{:SA:=-}	{:SA:=-}
{:SA:=-}	{:SA:=-}	53	{:SA:=-}
{:SA:=-}	{:SA:=-}	{:SA:=-}	{999:NM:=1}
{:SA:=-}	{:SA:=-}	{:SA:=-}	2
{:SA:=-}	{:SA:=-}	{999:NM:=5}	{:SA:=-}
{:SA:=-}	{999:NM:=2}	{:SA:=-}	{:SA:=-}

Considerando las ventajas del tipo de pregunta *Gapfill* vistas en la sección “Tipo de pregunta: Completar partes”, se modificó el mismo plugin para desarrollar la pregunta de ruteo de código basada en *Gapfill*, estas modificaciones consideran:

- Permitir la creación de una tabla llena de espacios en blanco sin que tengan una contribución a la calificación final.
- Nueva variable de penalización cuya cantidad es determinada por el profesor por rellenar un espacio que debe estar en blanco.

Para hacer esto posible es necesario agregar un nuevo campo a la tabla *question_gapfill* de la figura 17 llamado **penaltyfactor**, que guardará el descuento que se le hará a la calificación

de la pregunta por cada espacio respondido que debía permanecer vacío. Tras esta modificación, al momento de crear una pregunta de tipo *Gapfill*: ruteo de código, se debe ingresar la siguiente tabla al formulario de creación de pregunta:

Tabla 6: Tabla de ruteo de código correspondiente a figura 3 usando *Gapfill*: ruteo de código.

Fuente: Elaboración propia.

num	resultado	n	nd
53	@@	@@	@@
@@	@@	53	@@
@@	@@	@@	@1@
@@	@@	@@	2
@@	@@	@5@	@@
@@	@2@	@@	@@

Al igual que cómo se mostró en la sección anterior, a cada espacio por rellenar se le puede agregar retroalimentación cuando el estudiante responde correctamente tanto como para cuando responde incorrectamente

Question text: Complete el ruteo de la ejecución del programa.

num	resultado	n	nd
53	@@	@@	@@
@@	@@	53	@@
@@	@@	@@	@1@
@@	@@	@@	2
@@	@@	@5@	@@
@@	@2@	@@	@@

Add Gap settings: 1

Feedback for correct: Correcto!

Feedback for incorrect: Incorrecto.

OK

Figura 18: Interfaz para agregar retroalimentación a respuesta "1" usando *Gapfill*: ruteo de código.

Fuente: Elaboración propia.

Una vez creada la pregunta, luce de la siguiente forma para que el estudiante pueda responder:

Considere el siguiente programa que hace un llamado a una función:

```
def num_digitos(n):
    nd = 1
    while n > 9:
        nd += 1
        n = n // 10
    return nd

num = 53
resultado = num_digitos(num)
print(resultado)
```

Complete el ruteo de la ejecución del programa.

num	resultado	n	nd
53			
		53	
			2

Pantalla

Figura 19: Pregunta de figura 3 usando *Gapfill*: ruteo de código.
Fuente: Elaboración propia.

y su retroalimentación respondiendo todo correctamente es la siguiente:

Question **8**

Correct

Mark 1.00 out of 1.00

Flag question

Edit question

Considere el siguiente programa que hace un llamado a una función:

```
def num_digitos(n):
    nd = 1
    while n>9:
        nd += 1
        n = n//10
    return nd

num = 53
resultado = num_digitos(num)
print(resultado)
```

Complete el ruteo de la ejecución del programa.

num	resultado	n	nd
53			
		53	
			1
			2
		5	
	2		

Pantalla

2

Figura 20: Retroalimentación correcta de ruteo en figura 19.
Fuente: Elaboración propia.

Para dar un ejemplo de retroalimentación para una respuesta incorrecta y también una demostración del *penaltyfactor* al rellenar un espacio que debía permanecer en blanco:

Question **8**

Partially correct

Mark 2.90 out of 4.00

Flag question

Edit question

Considere el siguiente programa que hace un llamado a una función:

```
def num_digitos(n):
    nd = 1
    while n>9:
        nd += 1
        n = n//10
    return nd

num = 53
resultado = num_digitos(num)
print(resultado)
```

Complete el ruteo de la ejecución del programa.

num	resultado	n	nd
53			
		53	
			53 ✖ @1@ Incorrecto
			2
fill ✖		5	
	2		

Pantalla

2 ✔

Figura 21: Retroalimentación parcialmente correcta de ruteo en figura 19.

Fuente: Elaboración propia.

Muestra una calificación 2.90 de 4.00, si se considera que:

- Se tienen cuatro espacio que deben rellenarse, tres en la tabla de ruteo de código y uno de la tabla 'Pantalla', por lo que cada respuesta correcta aporta $\frac{4}{4}$ ó 1,00 de calificación. Este valor será dado por VRC (Valor Respuesta Correcta).
- Visto en la figura 18 se sabe que el *penalty factor* es 0.10, este valor estará dado por la variable PF (Penalty Factor).
- Se considerará CF como la calificación final.
- Se considerará i como la cantidad de respuestas correctas. $i = 3$ en la figura 21.
- Se considerará j como la cantidad de espacios que deben estar vacíos pero están rellenos. $j = 1$ en la figura 21.

Entonces la calificación final se calcula usando la siguiente fórmula:

$$CF = i * VRC - j * VRC * PF$$

$$CF = 3 * 1,00 - 1 * 1,00 * 0,10$$

$$CF = 3,00 - 0,10$$

$$CF = 2,90$$

3.3. Tipo de pregunta: Problemas de Parsons

Debido a que los problemas de Parsons no existen como tipo de pregunta oficial de Moodle o creada por usuarios externos es necesario su desarrollo para poder agregarlos a cuestionarios. Para comenzar es recomendado usar el módulo *Moodle plugin skeleton generator*⁶, herramienta que ayuda a desarrolladores al generar el directorio con los archivos necesarios en su forma más básica para crear plugins de distinto tipo en Moodle.

Como se desea desarrollar un *plugin* de problema de Parsons con andamiaje que pueda ser utilizado para cualquier curso introductorio de programación es necesario considerar lo siguiente:

- Durante el desarrollo se debe minimizar la cantidad de caracteres reservados para no restringir el lenguaje de programación de los programas a transformar a Problema de Parsons.
- Se debe permitir al usuario definir aquellos caracteres reservados necesarios para dar espacio a que pueda ser utilizado sobre cualquier lenguaje de programación.
- Se debe permitir la creación de bloques de código distractores.
- Se debe permitir la creación de bloques distractores pareados de código.
- Los bloques de código deben poder ser compuestos por una línea de código o más.

Para lograr lo anterior será necesario definir campos en la base de datos que representen los delimitadores para separar cada bloque de código, para separar cada distractor y para separar cada distractor pareado, por lo que el esquema en la base de datos del problema de Parsons luciría de la siguiente forma:

⁶https://moodle.org/plugins/tool_pluginskel

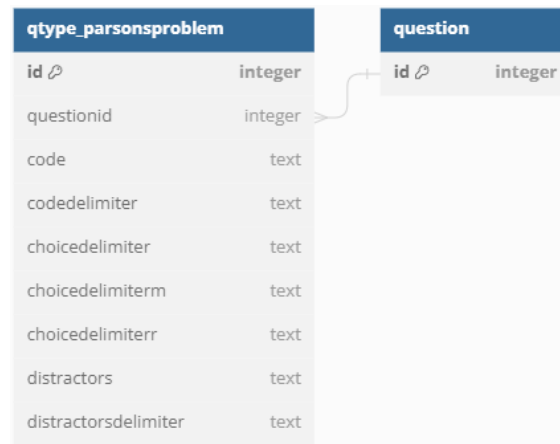


Figura 22: Esquema en la BBDD del tipo de pregunta problema de Parsons.
Fuente: Elaboración propia en herramienta dbdiagram.io.

Donde se ha simplificado la tabla *question* que es la pregunta base de Moodle y sólo se ha indicado su llave primaria. En la tabla que soporta el *plugin* problema de Parsons cada campo corresponde a:

- **id**: Identificador de la pregunta de tipo problema de Parsons.
- **questionid**: Llave foránea que referencia a la pregunta base *question*.
- **code**: Programa que será separado en bloques para formar un problema de Parsons.
- **codedelimiter**: Delimitador usado para separar cada bloque del problema de Parsons.
- **choicedelimiter**: Delimitador de la izquierda para crear un bloque distractor pareado.
- **choicedelimiterm**: Delimitador del medio para crear un bloque distractor pareado.
- **choicedelimiterr**: Delimitador de la derecha para crear un bloque distractor pareado.
- **distractors**: Distractores separados por el delimitador *distractorsdelimiter*.
- **distractorsdelimiter**: Delimitador usado para separar cada bloque distractor.

A la hora de crear una pregunta el profesor debe dirigirse al cuestionario de Moodle al que desea agregar el problema de Parsons y seleccionarlo:

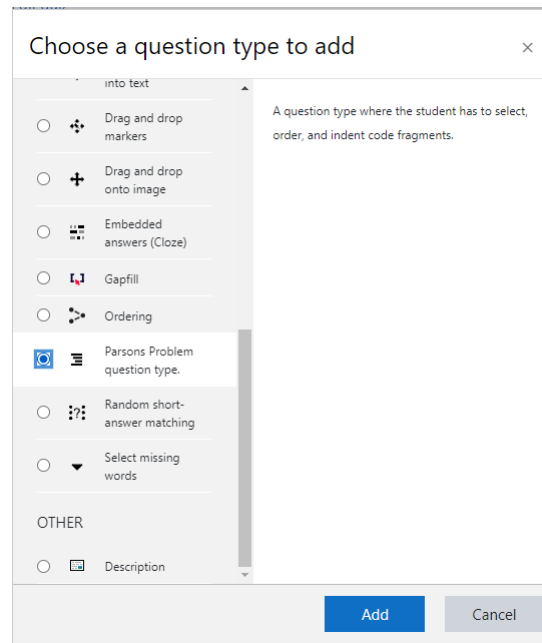


Figura 23: Menú de añadir pregunta a cuestionario
Fuente: Elaboración propia.

Una vez seleccionado el problema de Parsons, se deben llenar los siguientes campos en el formulario:

- **Category** - Se debe seleccionar a qué **categoría** pertenece esta nueva pregunta, ya sea de grupos predefinidos por Moodle como son el curso o el cuestionario o seleccionar una de las categorías definidas anteriormente por el usuario.
- **Question name** - Se debe definir el **nombre de la pregunta** para identificarla más fácilmente.
- **Question text** - Se debe definir el **enunciado de la pregunta** que se mostrará al estudiante a la hora de realizar el cuestionario.
- **Default mark** - Se debe definir la **puntuación por defecto**, es decir el impacto que tendrá esta pregunta sobre la calificación final del cuestionario.
- **General feedback** - (opcional) Se puede definir la **retroalimentación general** que se mostrará al estudiante luego de terminar el cuestionario independiente de si su respuesta es correcta, parcialmente correcta o incorrecta.
- **ID number** - (opcional) Se puede definir un **número de identificación** para tener otra manera de distinguir esta pregunta.
- **Insert the code** - Se debe ingresar el **código que será utilizado para generar el problema de Parsons**

- *Insert code delimiter* - Se debe ingresar el **delimitador de código** que servirá para separar y generar cada bloque de código, si se deja vacío se usará un salto de línea como delimitador predeterminado.

Si se desean agregar bloques distractores pareados de código se debe abrir el menú *Visually paired distractors delimiter* donde se puede ingresar los delimitadores izquierda, centro y derecha. De igual forma si se desean agregar bloques distractores se debe abrir el menú *Distractors* donde se puede ingresar el delimitador de los distractores y en la siguiente entrada los distractores deseados separados por el delimitador definido anteriormente.

The screenshot shows a Moodle form with two main sections. The first section, titled 'Visually paired distractors delimiter', contains three input fields: 'Insert left side visually paired distractor delimiter. The first choice will always be the correct one', 'Insert middle visually paired distractor delimiter', and 'Insert right side visually paired distractor delimiter'. Each field has a question mark icon to its right. The second section, titled 'Distractors', contains two input fields: 'Insert distractors delimiter.' and 'Insert distractors separated by your choice of distractors delimiter.'. The second field also has a question mark icon to its right.

Figura 24: Formulario para agregar bloques VPD (Visually Paired Distractors) y distractores.
Fuente: Elaboración propia.

3.3.1. Parsonsificador

Dada la complejidad que hay en la creación de un problema de Parsons en base a un programa es que se hace necesaria la creación de una herramienta externa que facilite su creación. Para cumplir con lo anterior se desarrolla una página web usando la librería *React* de JavaScript por las siguientes razones:

- Posee buena documentación oficial, además de una comunidad activa lo que facilita la resolución de problemas.
- Se ha usado anteriormente por lo que se está familiarizado con el lenguaje y la librería.
- Se fomenta la creación de componentes reutilizables, por lo que simplifica la estructura del código y su mantenibilidad.

Para mostrar un ejemplo del uso del Parsonsificador se tendrá el siguiente código:

```
1  n = int(input('Ingrese número de personas: '))
2  total = 0
3  mayor = 0
4  n_mayores = 0
5  i = 0
6  while i < n:
7      edad = int(input('Ingrese edad: '))
8      if edad > 18:
9          total = total + edad
10         n_mayores = n_mayores + 1
11     if edad > mayor:
12         mayor = edad
13     i += 1
14 prom = total/n_mayores
15 prom = round(prom)
16 print('Promedio de mayores de edad:', prom)
17 print('Mayor edad:', mayor)
```

Figura 25: Programa a transformar en problema de Parsons.

Fuente: Elaboración propia.

Del cual se desea que estén juntas las líneas de código 2, 3, 4 y 5 que son las inicializaciones de variables, y también las líneas 16 y 17 que corresponden a las dos salidas por pantalla. Como distractores se desean los bloques:

```
mayor = mayor + edad
```

```
if edad < mayor:
```

Y que se cree un VPD junto a la línea 12 que esté acompañada de:

```
edad = mayor
```

Para lograr lo anterior usaremos la herramienta Parsonsificator cuya interfaz luce de la siguiente forma:

Parsonsificator

El Parsonsificator se creó con el objetivo de facilitar la creación de problemas de Parsons en Moodle usando el plugin [gtype_parsonspproblem](#) debido a su complicada sintaxis para unir bloques o crear distractores pareados.

Para comenzar, los pasos recomendados a seguir son los siguientes:

1. Copiar y pegar tu código en el formulario "Ingresa acá tu código".
2. Automáticamente se generará cómo se vería tu problema de Parsons con bloques separados por saltos de línea.

Instrucciones si deseas unir bloques de código: ▼

Instrucciones si deseas crear bloques pareados de código: ▼

Ingresa acá tu código

Código

Ingresa delimitadores

Codedelimiter	VPD Izquierda	VPD Medio	VPD Derecha
Opcional	Opcional	Opcional	Opcional
También puedes presionar 'Unir bloques'	También puedes presionar 'Crear VP Distractores'		

Unir bloques Transformar bloque a VPD

Figura 26: Interfaz de herramienta Parsonsificador.
Fuente: Elaboración propia.

El primer paso que debemos completar es copiar y pegar nuestro código en el formulario, lo que instantáneamente nos mostrará cómo luciría un problema de Parsons con este código siendo separado por saltos de línea.

The interface shows a code editor at the top with the following Python code:

```
if edad > mayor:
    mayor = edad
i += 1
prom = total/n_mayores
prom = round(prom)
print('Promedio de mayores de edad:', prom)
print('Mayor edad:', mayor)
```

Below the code editor is a section titled "Ingresa delimitadores" with four input fields: "Codedelimiter", "VPD Izquierda", "VPD Medio", and "VPD Derecha". Each field has an "Opcional" button below it. Below these fields are two large blue buttons: "Unir bloques" and "Transformar bloque a VPD".

At the bottom, the code is displayed in a Parsons problem format, where each line of code is in its own box, ready to be dragged and dropped into the correct order:

```
n = int(input('Ingrese número de personas: '))
total = 0
mayor = 0
n_mayores = 0
i = 0
while i < n:
    edad = int(input('Ingrese edad: '))
    if edad > 18:
```

Figura 27: Código transformado a problema de Parsons.
Fuente: Elaboración propia.

Como deseamos que existan fragmentos de código de más de una línea de código es necesario definir un delimitador de código, para que la herramienta haga el trabajo por nosotros presionamos el botón **Unir bloques** lo que haría que se muestre una nueva ventana modal que nos permite definir un delimitador de código:

The "Unir bloques" modal window is displayed over the main interface. It contains a text input field labeled "Define tu Codedelimiter" with the value "/" entered. Below the input field are two buttons: "Cerrar" (Close) and "Guardar" (Save).

Figura 28: Interfaz para definir un delimitador de código.
Fuente: Elaboración propia.

En este caso como en ninguna parte de nuestro código usamos los caracteres “/*\” entonces los usaremos como delimitador de código, luego tras presionar el botón **Guardar** nuestro código es modificado y se abre la posibilidad de unir bloques de código:

The interface is titled "Ingresa acá tu código" and contains a text area with the following Python code:

```
n = int(input('Ingrese número de personas: '))/*\total = 0/*\mayor = 0/*\n_mayores = 0/*\i = 0/*\while i < n:/*\  edad =
int(input('Ingrese edad: '))/*\  if edad > 18:/*\    total = total + edad/*\    n_mayores = n_mayores + 1/*\  if edad >
mayor:/*\    mayor = edad/*\  i += 1/*\prom = total/n_mayores/*\prom = round(prom)/*\print('Promedio de mayores
de edad:', prom)/*\print('Mayor edad:', mayor)
```

Below the code editor is a section titled "Ingresa delimitadores" with four input fields:

- Code delimitar: /*\ (with a note: "También puedes presionar 'Unir bloques'")
- VPD Izquierda: Opcional (with a note: "También puedes presionar 'Crear VP Distractores'")
- VPD Medio: Opcional
- VPD Derecha: Opcional

At the bottom of this section are two buttons: "Cancelar" (red) and "Confirmar unión" (green).

Below the delimiter section is a list of code blocks, each with a "Unir" button:

- Block 1: `n = int(input('Ingrese número de personas: '))`
- Block 2: `total = 0`
- Block 3: `mayor = 0`
- Block 4: `n_mayores = 0`
- Block 5: `i = 0`

Figura 29: Interfaz para unir bloques de código.

Fuente: Elaboración propia.

Luego para unir fragmentos de código simplemente se presionan los botones **Unir** entre los bloques que se desean juntar, lo que nuevamente modificará el código que inicialmente copiamos y pegamos en la herramienta:

Ingresa acá tu código

```
n = int(input('Ingrese número de personas: '))\ntotal = 0\nmayor = 0\nn_mayores = 0\ni = 0\nwhile i < n:\n    edad = int(input('Ingrese edad: '))\n    if edad > 18:\n        total = total + edad\n        n_mayores = n_mayores + 1\n        mayor = edad\n    i += 1\nprom = total/n_mayores\nprint('Promedio de mayores de edad:', prom)
```

Ingresa delimitadores

Codelimitar	VPD Izquierda	VPD Medio	VPD Derecha
/\n	Opcional	Opcional	Opcional
También puedes presionar 'Unir bloques'			
También puedes presionar 'Crear VP Distractores'			

Unir bloques Transformar bloque a VPD

```
n = int(input('Ingrese número de personas: '))\n\ntotal = 0\nmayor = 0\nn_mayores = 0\ni = 0\n\nwhile i < n:\n\n    edad = int(input('Ingrese edad: '))\n\n    if edad > 18:
```

Figura 30: Interfaz luego de unir bloques de código.
Fuente: Elaboración propia.

Finalmente para crear bloques distractores pareados presionamos el botón **Transformar bloque a VPD**, lo que de igual forma a los distractores muestra una nueva ventana modal que nos permite definir los tres delimitadores necesarios, en el ejemplo usaremos los delimitadores [, ** y] como delimitadores izquierda, centro y derecha respectivamente.

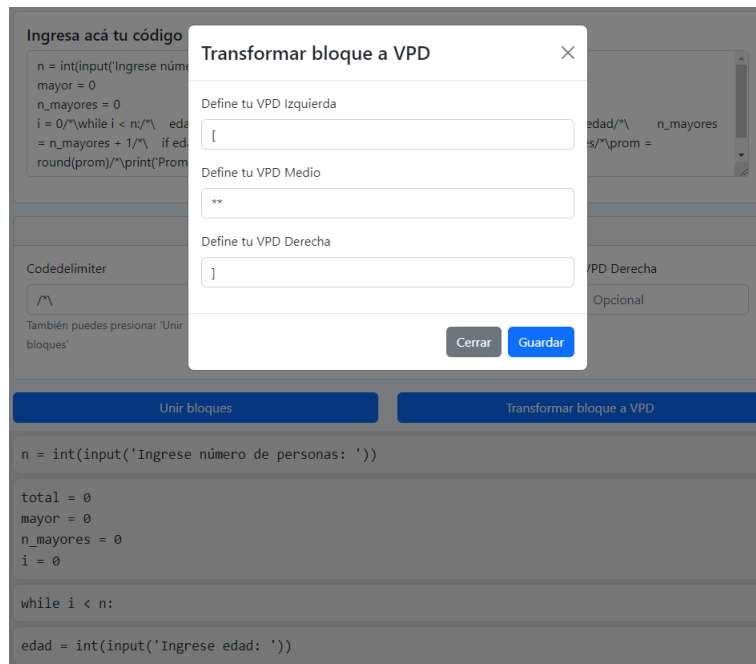


Figura 31: Interfaz para definir delimitadores de VPD.
Fuente: Elaboración propia.

Luego de presionar el botón **Guardar**, aparece una nueva opción en los bloques de código para generar bloques VPD, para crear el VPD que deseamos se presiona el botón **Crear VPD** asociado al bloque que buscamos modificar, lo que transforma el bloque a VPD con texto 'Modifica acá *i*' con *i* el número del índice del bloque de código.

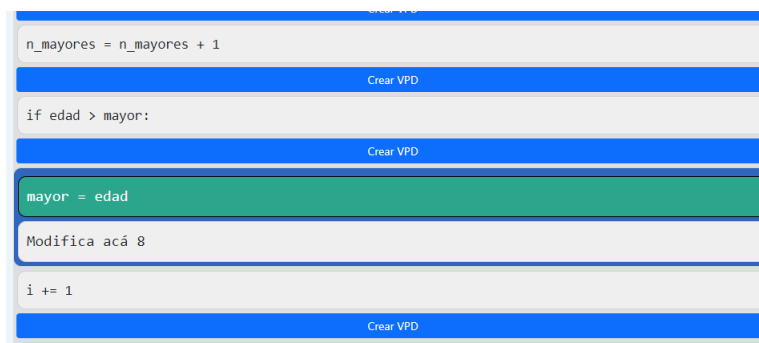


Figura 32: Nuevo bloque de código VPD.
Fuente: Elaboración propia.

Tras presionar el botón **Confirmar nuevos VPD** nuestro código cambia y el nuevo VPD se refleja en el código, por lo que fácilmente se puede cambiar el texto 'Modifica acá *i*'.

Ingresa acá tu código

```
n = int(input('Ingrese número de personas: '))\total = 0
mayor = 0
n_mayores = 0
i = 0\while i < n:\    edad = int(input('Ingrese edad: '))\    if edad > 18:\        total = total + edad\        n_mayores =
n_mayores + 1\    if edad > mayor:[        mayor = edad*Modifica acá 8]\    i += 1\prom = total/n_mayores\prom
= round(prom)\print('Promedio de mayores de edad:', prom)
print('Mayor edad:', mayor)
```

Figura 33: Código luego de transformar bloque a VPD.
Fuente: Elaboración propia.

Una vez finalizado el traspaso de código a problema de Parsons se tienen todos los datos necesarios para copiar y pegar en el formulario de creación de problema de Parsons en Moodle, lo que logra que nuestra pregunta luzca de la siguiente forma en un cuestionario:

Question 11

Not yet answered

Marked out of 1.00

Flag question

Edit question

Seleccione, ordene e indente los siguientes bloques de código para generar un programa que solicita la cantidad de personas a considerar y, para cada una, pide su edad. Calcula el promedio de quienes son mayores a 18 años, y determina la mayor edad entre todas las personas. Al final imprime el promedio calculado y la mayor edad.

n_mayores = n_mayores + 1

prom = total/n_mayores

print('Promedio de mayores de edad:', prom)

print('Mayor edad:', mayor)

if edad < mayor

n = int(input('Ingrese número de personas: '))

if edad > 18:

edad = mayor 8

mayor = edad

while i < n:

mayor = mayor + edad

if edad > mayor:

total = 0

mayor = 0

Figura 34: Problema de Parsons en cuestionario de Moodle.
Fuente: Elaboración propia.

CAPÍTULO 4

VALIDACIÓN DE LA SOLUCIÓN

Existirán dos instancias de validación sobre diferentes partes de lo desarrollado:

- Con la ayuda de estudiantes de la UTFSM se realizará una comparación entre el uso tradicional de los ejercicios habilitadores usando lápiz y papel y una nueva forma de aplicarlos usando AULA.
- Con la ayuda de profesores de un curso introductorio de programación de la UTFSM se evaluará el proceso de creación de los ejercicios habilitadores usando la solución implementada en AULA.

Que se detallan a continuación:

4.1. Validación con estudiantes

4.1.1. Metodología

Para validar la solución implementada se contará con un horario de ayudantía de duración 1 hora y 10 minutos en el cual se separará a los estudiantes en dos grupos, durante 25 minutos un grupo usará la plataforma AULA y el otro usará lápiz y papel para realizar ejercicios habilitadores. Luego, en los siguientes 25 minutos se intercambiarán, y quienes comenzaron en AULA ahora usarán lápiz y papel y el otro grupo viceversa. Al terminar se les pedirá a todos los estudiantes responder una encuesta para conocer su experiencia resolviendo ejercicios habilitadores usando AULA y lápiz y papel.

Los participantes pertenecen a los paralelos del ramo Programación (IWI-131) de la UTFSM que poseen en su mayoría a estudiantes de primer año de las carreras Ingeniería en Diseño de Productos⁷ e Ingeniería en Aviación Comercial⁸.

4.1.2. Cuestionario

Al momento de realizar este cuestionario los estudiantes cuentan con conocimientos obtenidos por las clases de programas secuenciales, condicionales y recién comienzan a aprender

⁷<https://usm.cl/admision/carreras/ingenieria-en-diseno-de-productos/>

⁸<https://usm.cl/admision/carreras/ingenieria-en-aviacion-comercial/>

sobre ciclos en Python. Debido a que se cuenta con un tiempo restringido se les pedirá a los estudiantes completar un problema de Parsons y un ruteo por cada método. Como los estudiantes se encuentran en la unidad de ciclos los ejercicios escogidos estarán relacionados a ese tema.

Cuando los estudiantes deban responder los ejercicios usando lápiz y papel se les entregarán hojas con los problemas escogidos de las figuras 35 y 36:

al ingresar al cuestionario:

Pregunta 1
Sin responder aún
Puntúa como 1,00
⚙ Marcar pregunta
🔗 Editor pregunta

Seleccione, ordene e indente los siguientes bloques de código para generar un programa que solicite la cantidad de personas a considerar y, para cada una, pide su edad. Calcula el promedio de quienes son mayores a 18 años, y determina la mayor edad entre todas las personas. Al final imprime el promedio calculado y la mayor edad.

```
mayor = mayor + edad  
  
n_mayores = n_mayores + 1  
  
n = int(input('Ingrese número de personas: '))  
  
total = 0  
mayor = 0  
n_mayores = 0  
i = 0  
  
if edad > 18:  
  
if edad < mayor:  
  
prom = round(prom)  
  
mayor = edad  
  
while i < n:  
  
print('Promedio de mayores de edad:', prom)  
print('Mayor edad:', mayor)  
  
edad = int(input('Ingrese edad: '))  
  
prom = total/n_mayores
```

Figura 37: Problema de Parsons a realizar usando AULA.
Fuente: Elaboración propia.

Pregunta 2

Sin responder aún

Puntúa como 1,00

🚩 Marcar pregunta

⚙️ Editar pregunta

Realice el ruteo del siguiente programa e indique qué es lo que imprime. Cada vez que el valor de una variable cambie, escríbalo en una nueva fila de la tabla. Suponga que la persona que usa el programa ingresa el valor 5.

n	f1	f2	fn	i

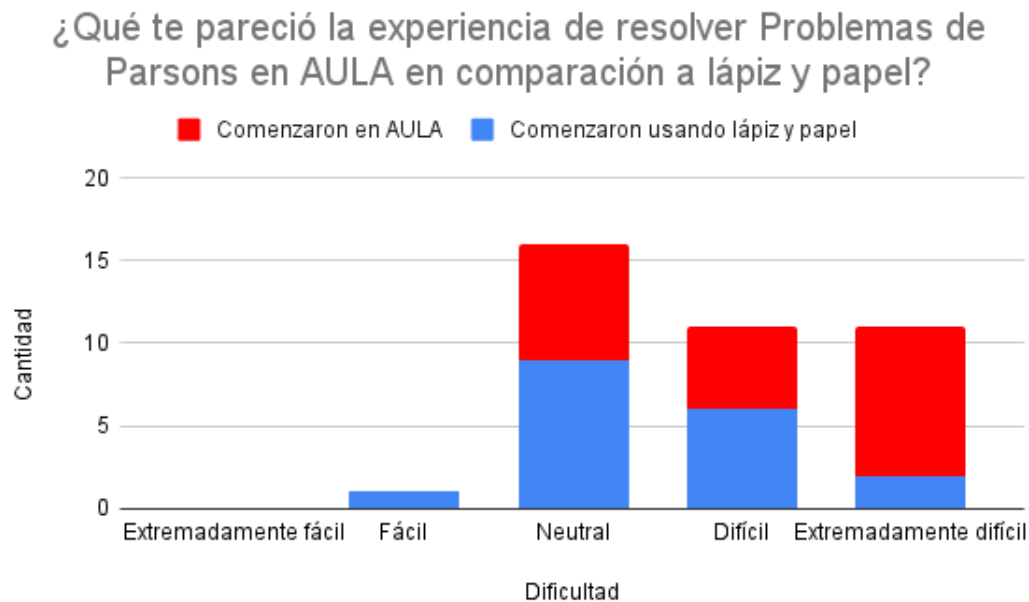


Figura 39: Percepción de dificultad de los estudiantes resolviendo problemas de Parsons en AULA en comparación a lápiz y papel.

Fuente: Elaboración propia.

Aquellos estudiantes que comenzaron el cuestionario usando AULA sintieron una dificultad mayor en comparación a aquellos que comenzaron usando lápiz y papel, que en su mayoría percibieron una dificultad neutral y difícil.

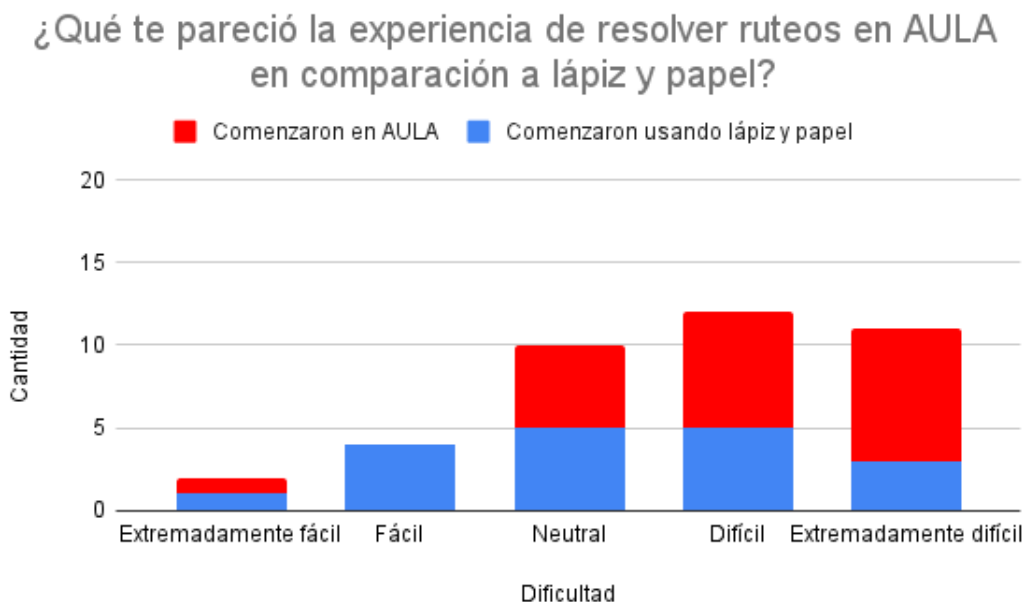


Figura 40: Percepción de dificultad de los estudiantes resolviendo ruteos en AULA en comparación a lápiz y papel.

Fuente: Elaboración propia

De igual manera que en la primera pregunta, los estudiantes que comenzaron en AULA perciben una dificultad mayor en comparación a quienes comenzaron usando lápiz y papel, aunque a diferencia de la pregunta anterior existe una mayor cantidad de estudiantes con una percepción fácil y extremadamente fácil del ejercicio de ruteo.

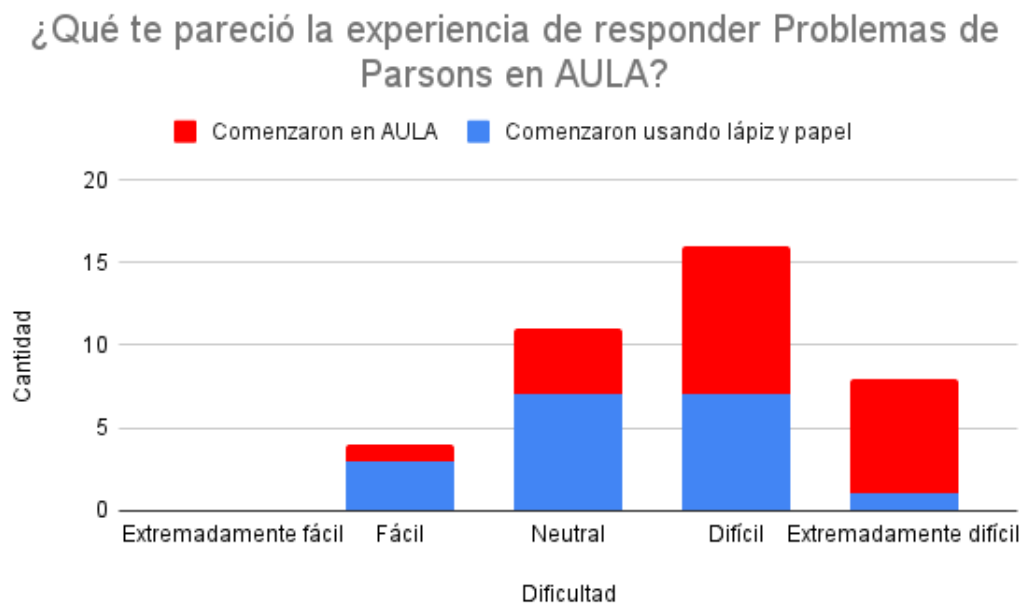


Figura 41: Percepción de dificultad de los estudiantes resolviendo problemas de Parsons en AULA.

Fuente: Elaboración propia.

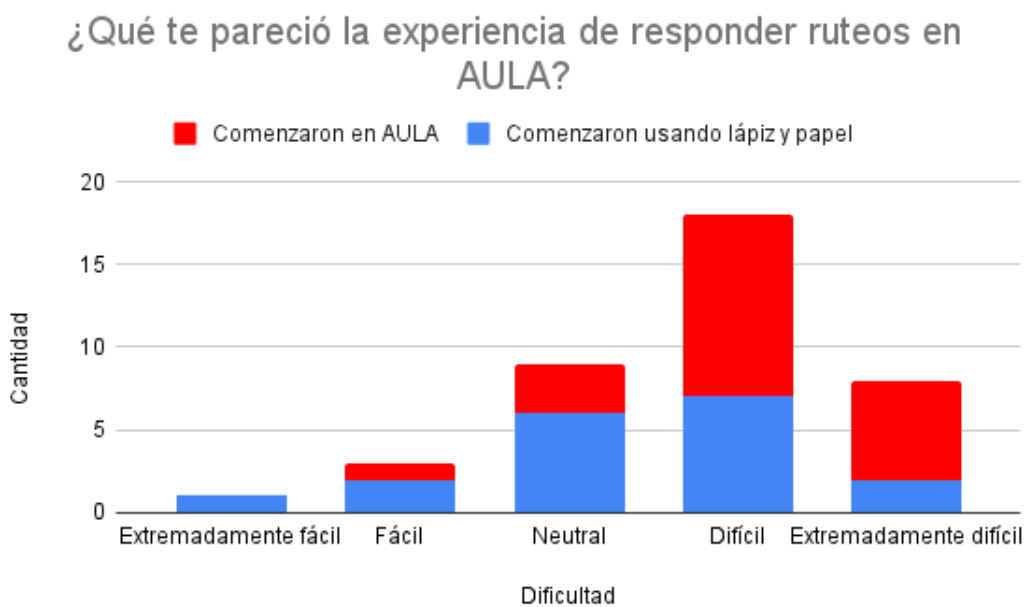


Figura 42: Percepción de dificultad de los estudiantes resolviendo ruteos en AULA.

Fuente: Elaboración propia.

En su mayoría los estudiantes perciben difícil de resolver ambos tipos de ejercicios habilitadores en AULA.

La segunda pregunta busca saber si los ejercicios habilitadores en AULA fueron lo suficientemente atractivos para que los estudiantes deseen seguir usándolos para aprender a programar.

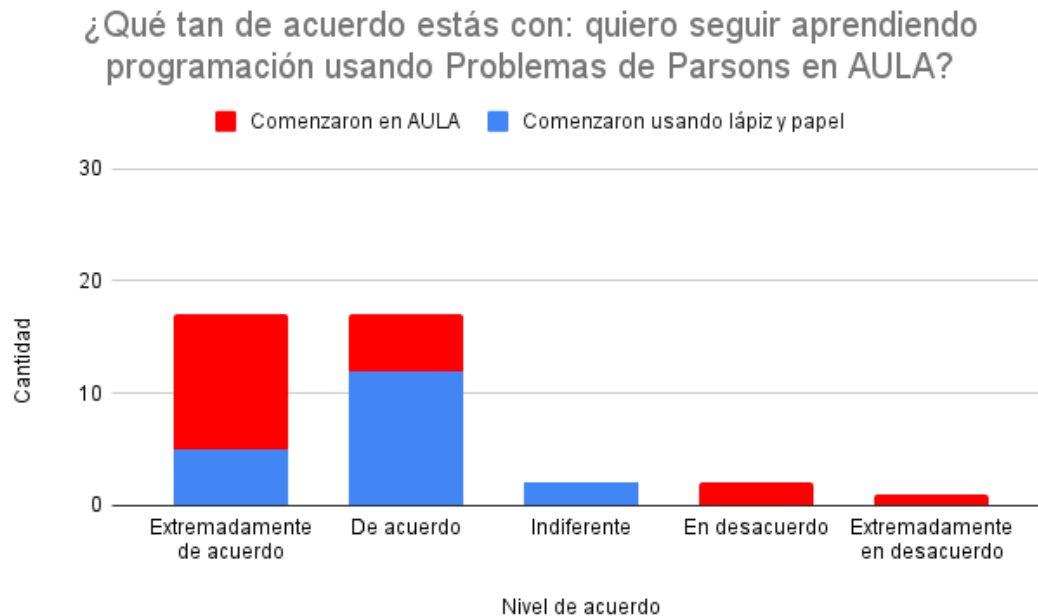


Figura 43: Niveles de acuerdo de querer seguir aprendiendo programación usando problemas de Parsons en AULA.

Fuente: Elaboración propia.

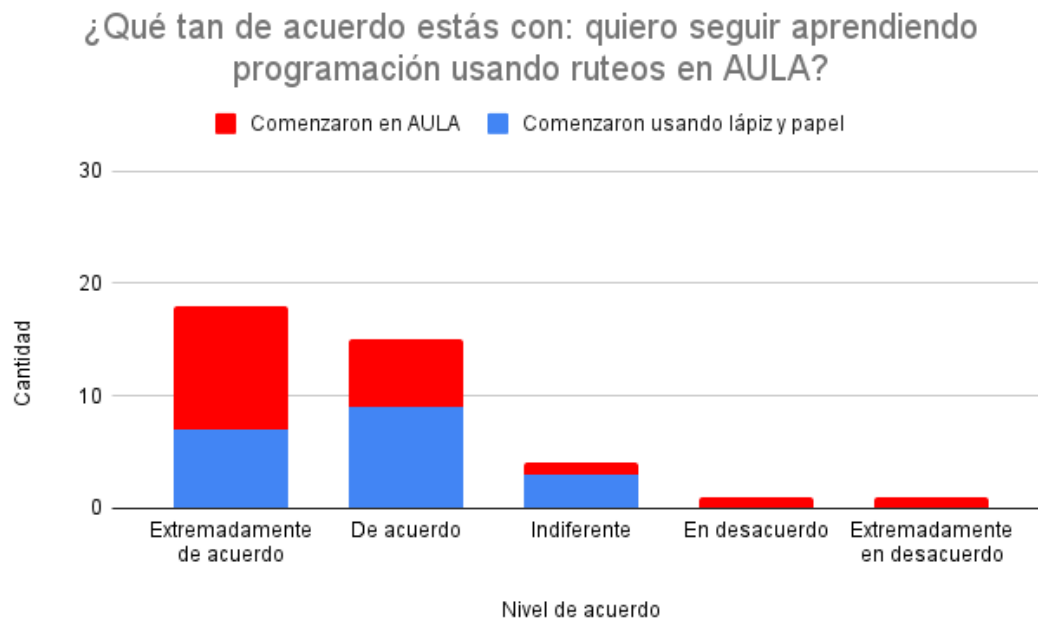


Figura 44: Niveles de acuerdo de querer seguir aprendiendo programación usando ruteos en AULA.

Fuente: Elaboración propia.

En su mayoría y para ambos tipos de ejercicios habilitadores los estudiantes desean seguir usándolos para aprender programación, con una preferencia mayor aquellos estudiantes que comenzaron el cuestionario en AULA en comparación a quienes empezaron usando lápiz y papel.

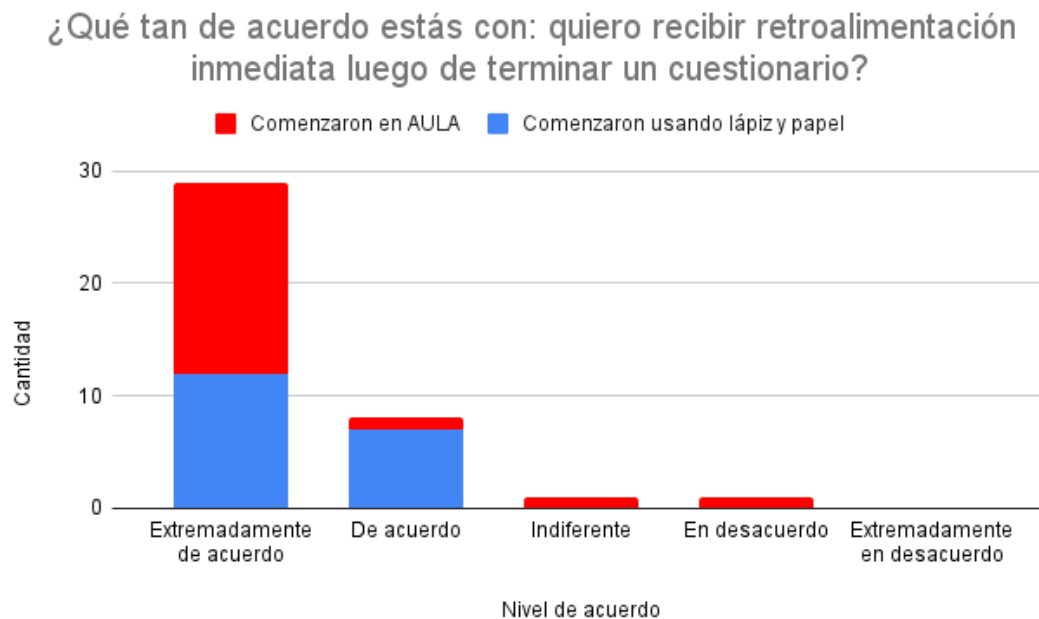


Figura 45: Niveles de acuerdo de querer recibir retroalimentación inmediata luego de terminar un cuestionario.

Fuente: Elaboración propia.

En lo que los estudiantes están muy de acuerdo es en querer recibir retroalimentación inmediata luego de terminar un cuestionario y eso se aprecia en la figura 45.

La tercera pregunta se hizo con el objetivo de saber la diferencia de tiempo empleado para resolver los ejercicios en AULA con respecto a usar lápiz y papel

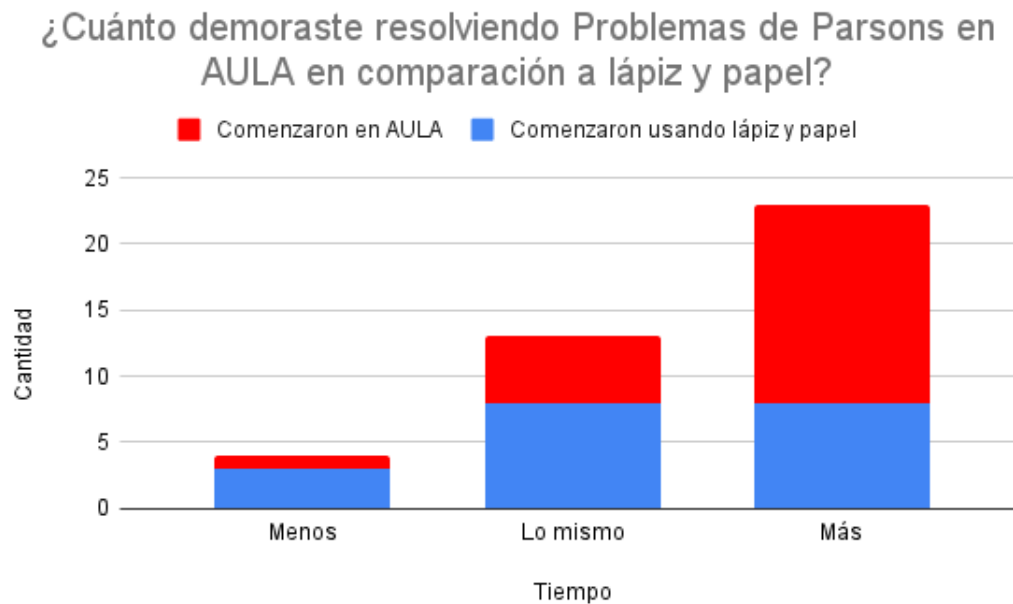


Figura 46: Cuánto demoraron los estudiantes resolviendo problemas de Parsons en AULA con respecto a usando lápiz y papel.

Fuente: Elaboración propia.

La cantidad de estudiantes que empezaron usando lápiz y papel y que tardaron lo mismo, o más, para resolver problemas de Parsons en AULA, es la misma. A diferencia de los estudiantes que comenzaron usando AULA, la mayoría de ellos tardaron más en completar los problemas en AULA en comparación con el uso de lápiz y papel.

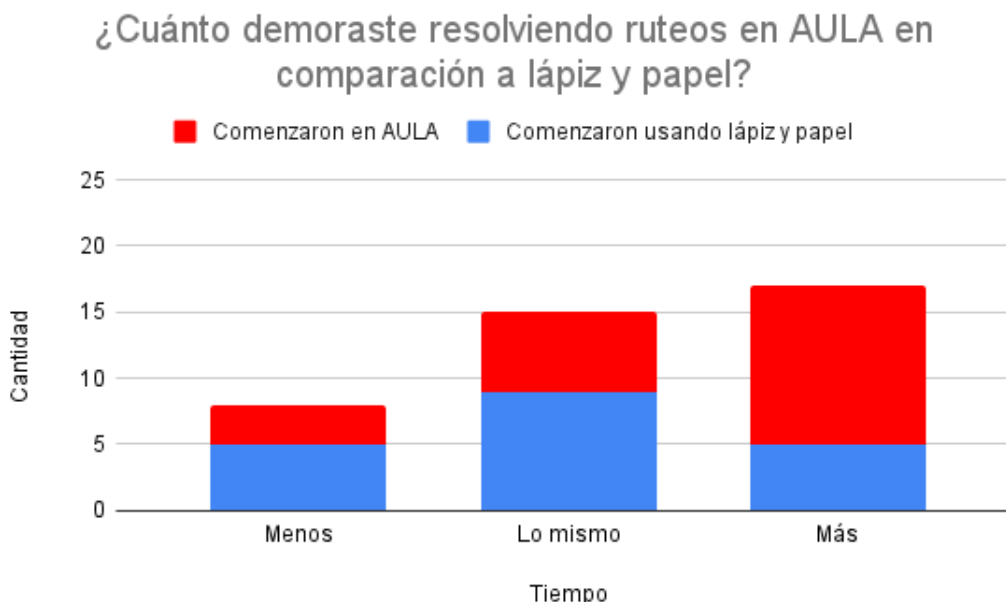


Figura 47: Cuánto demoraron los estudiantes resolviendo ruteos en AULA con respecto a usando lápiz y papel.

Fuente: Elaboración propia.

En los ruteos, la mayoría de los estudiantes que comenzaron usando lápiz y papel demoraron el mismo tiempo en AULA que usando lápiz y papel. Mientras que la mayoría de quienes comenzaron en AULA fueron más rápidos para resolver ruteos usando lápiz y papel.

La última pregunta “¿Cómo se podría usar esta herramienta para apoyar el proceso de aprendizaje de un/a estudiante de primer año de la UTFSM que está en IWI-131?” es de respuesta libre y las observaciones más destacadas son:

- “Quizás con una guía de qué podemos obtener con los comandos que estemos creando a partir del lenguaje que nos enseñan.”
Los ejercicios habilitadores en AULA pueden usarse para que luego de que los estudiantes aprendan un patrón puedan usarlo sin tener la necesidad de escribir un programa desde cero, sino que pueden completar un ruteo de un programa que utilice el patrón para predecir su salida, y luego ordenar un problema de Parsons para identificar si lograron aplicar el patrón.
- “El hecho de tener retro alimentación ayuda mucho.”
- “para tener una retroalimentación inmediata, ya que los contenidos los encuentro muy difíciles así tengo una explicación rápida de lo que no sé.”

La retroalimentación inmediata es de suma importancia para los estudiantes ya que permite una identificación rápida de errores y hace que el proceso de aprendizaje fluya mejor ya que conocen el error y pueden corregirlo inmediatamente.

- *“Un poco más de aprendizaje en clase que depender de los videos, ya que la clase la clase sirve de practica, ósea está bien, pero yo creo que un poco más en la clase.”*

Ya no sería necesario usar otra plataforma o imprimir hojas con ejercicios habilitadores, sino que basta con crear un cuestionario en AULA para proyectarlos durante la clase o para que los estudiantes los realicen en clases prácticas.

4.2. Validación con profesores

Para validar el proceso de creación de los ejercicios habilitadores se solicitó la colaboración de dos profesores de la asignatura Programación (IWI-131) de la UTFSM que tuvieran experiencia en la plataforma AULA.

La actividad a completar es crear un ejercicio de ruteo de código usando el *plugin* Gapfill y un problema de Parsons usando el módulo desarrollado en un cuestionario en AULA, como ayuda se les proveyó videos tutorial⁹ para el proceso de creación de preguntas. Luego de que completaran la actividad se les pidió responder una encuesta para conocer su experiencia en el proceso de creación de los ejercicios en un cuestionario de AULA.

4.2.1. Resultados encuesta

Las primeras dos preguntas buscan saber la opinión de los profesores con respecto a la validez de los nuevos tipos de ejercicios en AULA como ayuda para el aprendizaje de los estudiantes. Ambas respuestas son positivas.

⁹<https://www.youtube.com/watch?v=59ydsxvHDyM> y https://www.youtube.com/watch?v=_3UqHh92ILI

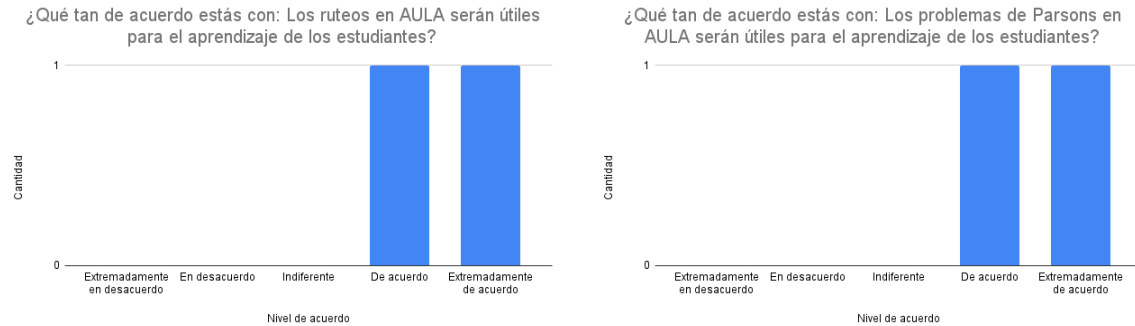


Figura 48: Niveles de acuerdo respecto a utilidad para el aprendizaje de los estudiantes de la solución desarrollada.

Fuente: Elaboración propia.

Las siguientes dos preguntas fueron planteadas para conocer la dificultad percibida por los profesores a la hora de crear los nuevos tipos de pregunta en AULA, en el caso del problema de Parsons, utilizando el Parsonsificador.

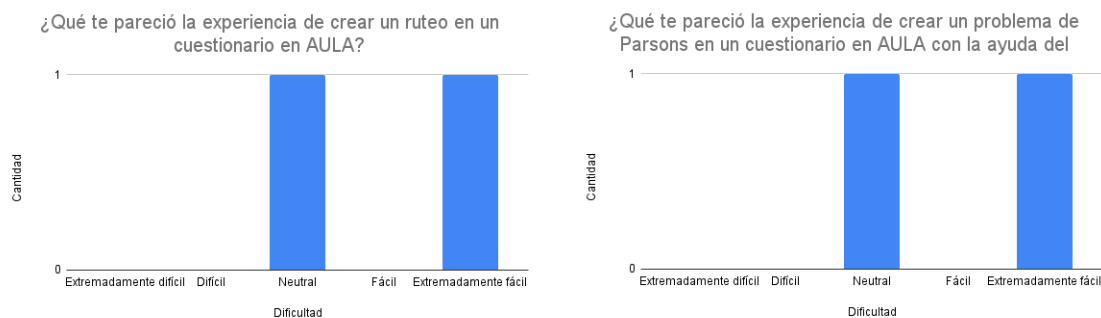


Figura 49: Percepción de dificultad de los profesores al crear ejercicios habilitadores en AULA.

Fuente: Elaboración propia.

Para un profesor la dificultad fue neutral en ambos tipos de pregunta, mientras que para el otro fue extremadamente fácil.

También fue planteada la pregunta *¿Qué tan difícil crees que hubiera sido crear un problema de Parsons sin la ayuda del Parsonsificador?* para saber si la complejidad percibida sobre la creación del problema de Parsons fue correcta a la hora de tomar la decisión para desarrollar la herramienta web.

Siendo 1 Extremadamente difícil y 10 Extremadamente fácil ¿Qué tan difícil crees que hubiera sido crear un problema de Parsons sin la

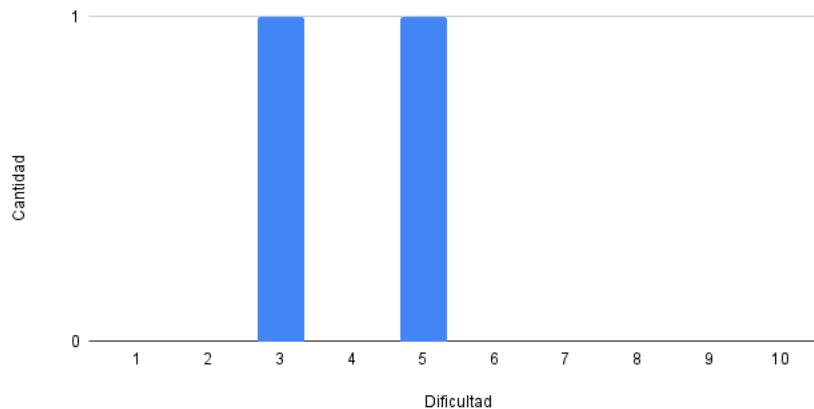


Figura 50: Percepción de dificultad de los profesores al crear problemas de Parsons en AULA sin usar Parsonsificador.

Fuente: Elaboración propia.

En promedio, los profesores consideran difícil la creación de problemas de Parsons en AULA sin la ayuda del Parsonsificador, lo que valida su creación.

Las siguientes dos preguntas fueron de respuesta abierta, la primera *¿Tienes algún comentario, sugerencia y/o queja sobre los nuevos tipos de pregunta en aula y/o el Parsonsificador?* buscaba darle espacio al profesor para que diera su opinión sobre parte del proceso de creación de las preguntas, las respuestas fueron las siguientes:

- *En la pregunta de ruteo, al crear la tabla, si en una columna no se coloca el espacio en blanco para rellenar, la columna queda muy pequeña. Sé que es algo normal de aula, pero lo comento por si es posible modificar. Además, al mostrar las respuestas correctas, estas podrían mostrarse solas, por ejemplo mostrar 5 en lugar de @5@.*

Ambas sugerencias respecto a cómo luce la pregunta tras su creación son realizables.

- *Me gustó mucho la herramienta.*

La segunda pregunta *¿Cómo se podría usar esta herramienta para apoyar el proceso de aprendizaje de un/a estudiante de primer año de la UTFSM que está en IWI-131?* buscaba que el profesor diera su opinión del uso de los nuevos tipos de pregunta en base a su experiencia siendo docentes de un curso introductorio de programación, cuyas respuestas fueron:

- *Creo que sería positivo tanto para generar evaluaciones formativas, como para generar una base de problemas para la ejercitación de los estudiantes.*

- *Generalmente realizamos controles para cada unidad del curso. Con este tipo de herramientas, podríamos agregar ejercicios de ruteo y/o parsons en las evaluaciones.*

Ambas respuestas mencionan como válida la solución desarrollada para las evaluaciones en AULA que se hacen a estudiantes durante el curso.

CAPÍTULO 5

CONCLUSIONES

En la presente memoria, se implementó una serie de ejercicios habilitadores con andamiaje auto corregibles en la plataforma Moodle, con la finalidad de ser usados en cursos introductorios de programación y apoyar el proceso de aprendizaje del estudiante. Para lograrlo se revisó cómo son aplicados los ejercicios habilitadores en la asignatura Programación (IWI-131) de la UTFSM y se propusieron alternativas de uso para los existentes y un nuevo tipo de pregunta de problemas de Parsons.

En los ejercicios de completar partes a los estudiantes se les entrega un programa en el que algunas partes del código han sido recortadas y deben ser rellenadas, para la creación de este tipo de ejercicio en Moodle se recomienda el uso del tipo de pregunta *Gapfill* cuando las respuestas del ejercicio estén compuestas sólo por rellenados en blanco donde el estudiante deba escribir, debido a la sintaxis más simple para la creación de las preguntas y el proceso más intuitivo para agregar retroalimentación a cada respuesta. En cualquier otro caso, ya sea en el que se desee usar una combinación de respuestas como por ejemplo rellenados en blanco y además el uso de menús *dropdowns*, o sólo *dropdowns*, entonces se recomienda el uso del tipo de pregunta respuestas anidadas (Cloze) que si bien tiene una sintaxis más compleja, soporta la combinación de tipos de respuesta y es posible definir cuáles son las alternativas incluidas en la selección de las respuestas con menú *dropdown*.

En los ejercicios de ruteo de código a los estudiantes se les entrega un programa y deben ejecutar mentalmente cada línea de código y ser capaces de escribir el valor de todas las variables en cada paso y/o predecir el resultado del programa, para la creación de este tipo de ejercicio en Moodle se recomienda el uso del tipo de pregunta *Gapfill* adaptada para ruteos de código, debido a su simple sintaxis, poder crear una tabla con espacios de respuesta en blanco sin que tengan una contribución a la calificación final, que es parte importante de una pregunta de ruteo de código y parte del andamiaje que se puede agregar o quitar, y poder agregar un factor de penalización que resta de la calificación final cuando el estudiante responde sobre un espacio que debía permanecer en blanco. En casos donde se desea combinar tipos de respuestas entonces se recomienda el uso del tipo de pregunta respuestas anidadas (Cloze).

En los ejercicios de problemas de Parsons los estudiantes deben ordenar un programa correctamente escrito separado en bloques con órdenes aleatorios, para la creación de este tipo de ejercicio en Moodle se desarrolló un nuevo tipo de pregunta con el objetivo de que funcionara para cualquier lenguaje de programación y con diferentes formas de andamiaje para cambiar la dificultad de la pregunta, como lo son los bloques distractores, los bloques distractores pareados y los bloques de código de más de una línea. Debido a la flexibilidad que se busca para que funcione con cualquier lenguaje de programación al no tener caracteres reservados es que se vuelve compleja la creación de problemas de Parsons en Moodle,

por lo que también se desarrolló el Parsonsificador, herramienta web que facilita y simplifica la creación de problemas de Parsons en Moodle.

Para evaluar los tipos de pregunta se realizó una encuesta de percepción a estudiantes que en el transcurso de una hora realizaron ejercicios habilitadores usando la solución propuesta en Moodle, como también usando lápiz y papel. La alta dificultad de los ejercicios seleccionados fue reflejada en la baja cantidad de estudiantes que lograron completarlos en ambos métodos y las respuestas a la encuesta post-cuestionario, que muestran una percepción de dificultad alta al realizar los ejercicios, aún así, los estudiantes expresaron el deseo de continuar aprendiendo programación usando los ejercicios habilitadores implementados en Moodle.

Para evaluar el proceso de creación de los nuevos tipos de pregunta se realizó una actividad a dos profesores que consistía en la creación de una pregunta de ruteo usando el *plugin* Gapfill modificado para ruteos y una pregunta de problema de Parsons usando el *plugin* desarrollado y la ayuda del Parsonsificador, al término de la actividad se les solicitó rellenar una encuesta para conocer su experiencia y su opinión respecto a la solución propuesta. Ambos profesores consideraron que los nuevos tipos de pregunta serán útiles para el aprendizaje de los estudiantes y en promedio consideraron fácil la creación de los nuevos tipos de pregunta.

REFERENCIAS BIBLIOGRÁFICAS

- [Bennedsen y Caspersen, 2019] Bennedsen, J. y Caspersen, M. E. (2019). Failure rates in introductory programming: 12 years later. *ACM inroads*, 10(2):30–36.
- [Duran et al., 2022] Duran, R., Zavgorodniaia, A., y Sorva, J. (2022). Cognitive load theory in computing education research: A review. *ACM Transactions on Computing Education (TOCE)*, 22(4):1–27.
- [Ericson et al., 2017] Ericson, B. J., Margulieux, L. E., y Rick, J. (2017). Solving parsons problems versus fixing and writing code. En *Proceedings of the 17th Koli Calling International Conference on Computing Education Research*, pp. 20–29.
- [Jenkins, 2002] Jenkins, T. (2002). On the difficulty of learning to program. En *Proceedings of the 3rd Annual Conference of the LTSN Centre for Information and Computer Sciences*, volumen 4, pp. 53–58. Citeseer.
- [Kinnunen y Malmi, 2006] Kinnunen, P. y Malmi, L. (2006). Why students drop out cs1 course? En *Proceedings of the second international workshop on Computing education research*, pp. 97–108.
- [Kumar, 2013] Kumar, A. N. (2013). A study of the influence of code-tracing problems on code-writing skills. En *Proceedings of the 18th ACM conference on Innovation and technology in computer science education*, pp. 183–188.
- [Kumar, 2015] Kumar, A. N. (2015). Solving code-tracing problems and its effect on code-writing skills pertaining to program semantics. En *Proceedings of the 2015 acm conference on innovation and technology in computer science education*, pp. 314–319.
- [Luxton-Reilly, 2016] Luxton-Reilly, A. (2016). Learning to program is easy. En *Proceedings of the 2016 ACM Conference on Innovation and Technology in Computer Science Education*, pp. 284–289.
- [Margulieux et al., 2019] Margulieux, L. E., Morrison, B. B., y Decker, A. (2019). Design and pilot testing of subgoal labeled worked examples for five core concepts in cs1. En *Proceedings of the 2019 ACM Conference on Innovation and Technology in Computer Science Education*, pp. 548–554.
- [MIFUTURO, 2023a] MIFUTURO (2023a). Bases de datos de estudiantes matriculadas y matriculados en educación superior. <https://www.mifuturo.cl/bases-de-datos-de-matriculados/>. Accedido: 14-12-2023.
- [MIFUTURO, 2023b] MIFUTURO (2023b). Buscador estadísticas por carrera. <https://www.mifuturo.cl/buscador-de-estadisticas-por-carrera/>. Accedido: 14-12-2023.

- [Parsons y Haden, 2006] Parsons, D. y Haden, P. (2006). Parson's programming puzzles: a fun and effective learning tool for first programming courses. En *Proceedings of the 8th Australasian Conference on Computing Education-Volume 52*, pp. 157–163.
- [Petersen et al., 2016] Petersen, A., Craig, M., Campbell, J., y Tafliovich, A. (2016). Revisiting why students drop cs1. En *Proceedings of the 16th Koli Calling International Conference on Computing Education Research*, pp. 71–80.
- [Robins et al., 2003] Robins, A., Rountree, J., y Rountree, N. (2003). Learning and teaching programming: A review and discussion. *Computer science education*, 13(2):137–172.
- [Sorva, 2012] Sorva, J. (2012). *Visual program simulation in introductory programming education*. Aalto University.
- [Sweller, 1988] Sweller, J. (1988). Cognitive load during problem solving: Effects on learning. *Cognitive science*, 12(2):257–285.
- [Sweller, 2010] Sweller, J. (2010). Element interactivity and intrinsic, extraneous, and germane cognitive load. *Educational psychology review*, 22:123–138.
- [Van Der Stuyf, 2002] Van Der Stuyf, R. R. (2002). Scaffolding as a teaching strategy. *Adolescent learning and development*, 52(3):5–18.
- [Van Merriënboer, 1990] Van Merriënboer, J. J. (1990). Strategies for programming instruction in high school: Program completion vs. program generation. *Journal of educational computing research*, 6(3):265–285.
- [Verenikina, 2003] Verenikina, I. (2003). Understanding scaffolding and the zpd in educational research.
- [Vygotsky y Cole, 1978] Vygotsky, L. y Cole, M. (1978). *Mind in Society: Development of Higher Psychological Processes*. Harvard University Press.
- [Weintrop, 2019] Weintrop, D. (2019). Block-based programming in computer science education. *Communications of the ACM*, 62(8):22–25.
- [Weintrop y Wilensky, 2015] Weintrop, D. y Wilensky, U. (2015). To block or not to block, that is the question: students' perceptions of blocks-based programming. En *Proceedings of the 14th international conference on interaction design and children*, pp. 199–208.
- [Wood et al., 1976] Wood, D., Bruner, J. S., y Ross, G. (1976). The role of tutoring in problem solving. *Journal of child psychology and psychiatry*, 17(2):89–100.
- [Xie et al., 2019] Xie, B., Loksa, D., Nelson, G. L., Davidson, M. J., Dong, D., Kwik, H., Tan, A. H., Hwa, L., Li, M., y Ko, A. J. (2019). A theory of instruction for introductory programming skills. *Computer Science Education*, 29(2-3):205–253.