UCLouvain

epl

École polytechnique de Louvain

Integrating Parsons problems in a CS1 programming course autograder

Author: **Corentin Lengelé**
Supervisors: **Kim Mens, Olivier Goletti, Anthony Gego**
Reader: **Olivier Bonaventure**
Academic year 2023–2024
Master [60] in Computer Science

Contents

Abstract

Computer science courses often suffer from the perception that learning programming is difficult. A significant issue comes from the disparity between the expectations placed on students and the limited time available for learning. Recognizing the need for improvement, CS1 instruction increasingly prioritize student-centered and practice-oriented approaches. Inginious, an automatic grading platform, addresses this need but does not support Parsons problems, an interactive exercise type, effective for teaching programming to novices. Integrating Parsons problems into Inginious has the potential to enrich the platform and further enhance the learning experience. This work aims to develop a plugin for Inginious to support Parsons problems, focusing on interface design and feedback provision. Additionally, a tutorial course for instructors will provide documentation about the plugin and contribute to the validation process. This validation, conducted through instructions and a survey, seeks to gather information to evaluate whether this integration is beneficial for instructors and students.

Chapter 1

Introduction

Despite the increasing demand for computational skills, computer science courses often yield high failure rates and foster a perception of programming as a difficult domain. A key issue contributing to this challenge is the disparity between the expectations placed on students and the limited timeframe available for learning. Traditional programming instruction tends to require mastering complex concepts and syntax within a short period, leaving little room for foundational understanding [1]. To address these challenges and enhance CS1 instruction, there is a growing recognition of the need for student-centered and practice-oriented approaches [2]. Automatic grading platforms, like Inginious, have emerged as a solution to promote these approaches by allowing students to practice coding independently while still receiving instructor support.

However, Inginious does not support Parsons problems, a valuable exercise type known for its effectiveness in teaching programming skills to novices. These problems involve rearranging given code blocks to form a correct solution [3]. This type of exercises serves as an intermediary between multiple-choice questions and coding exercises. While some benefits of Parsons problems have been assessed by research, gaps still remain in fully understanding their learning effects [4]. Introducing Parsons problems in Inginious has the potential to enrich the platform and enhance the learning experience of CS1 students, while also providing opportunities for future research to fill some of the identified gaps. The purpose of this work is to integrate Parsons problems into the Inginious autograder through the development of a plugin, with a special intention given to the interface design as well as the configurable feedback given to students. Another goal is to design a tutorial course for instructors using Inginious, teaching them how to use Parsons problems on the platform. This course will also help to conduct the validation, supplemented by a survey, in order to gather information to evaluate whether this integration is beneficial for instructors, students, or both.

The following chapters will provide a review of the related literature, explain the methodology used in designing and implementing the plugin, detail the validation process and its results, present the integration outcomes, and suggest directions for future research and improvements.

ChatGPT has been employed to verify the grammatical accuracy and refine the formulation of this article, although all content was manually generated.

Chapter 2

Related work

In recent years, there has been a growing recognition of the challenges faced by students in introductory computer sciences courses (CS1), particularly in the programming courses. Among CS1 students, the majority are considered novice programmers when beginning their degree [5]. Despite the increasing demand for computational skills in various fields, the introductory programming course often poses significant difficulties for students, leading to high rates of failure and a perception of programming as a difficult domain to learn. One of the central issues contributing to the perceived difficulty of the CS1 course is the misalignment between the expectations placed on students to learn all concepts and the limited time frame available for learning [1]. The traditional approach to teaching programming often prioritizes the mastery of complex concepts and syntax within a relatively short period of time, leaving little room for students to develop a solid foundation in programming principles.

To address these challenges and improve the teaching of the CS1 programming courses, there has been a significant focus on creating automatic grading platforms, which offers students the opportunity to actively engage in practical programming learning through coding. These platforms have emerged as a crucial tool in providing a student-centered approach to learning programming, allowing students to practice programming independently and receive immediate feedback on their work. This autonomy for practicing serves to enhance their understanding of programming concepts at their own pace through iterative trial and errors. The development of such platforms was not simply a desire but a requirement due to the high number of students, where the need for practice was undeniable but the volume of students made manual correction impractical [6]. The practice side of the course is student-centered, with the teaching staff serving more as guides and mentors rather than authority figures [2]. This allows them to spend more time focusing on complex subjects and providing personalized guidance where it is needed during the theoretical side of the course. Additionally, this approach enables the teaching staff to seek a good balance between student-centered and teacher-centered methods, optimizing the overall learning experience.

2.1 The INGInious autograding platform

Inginious [7] is an automatic grading platform developed by the UCLouvain. It is designed to enhance the effectiveness of evaluating code submissions in educational contexts. Composed of two integral components - the front-end and the back-end - the platform offers instructors a solution to manage coding assignments and enables them to provide pertinent feedback to students. Various types of problems such as coding exercises, multiple-choice questions or mathematical problems are already integrated into the platform.

The front-end serves as the user interface part of Inginious. It provides a web interface

that students can access to interact with the system. It is designed to be simple and intuitive, allowing students to submit their code solutions and track their progress. Another page is accessible to users with a teaching role, serving as a course administration page where instructors can write and manage tasks directly within the web browser, making it easy to create and update assignments.

The back-end is the core processing component of Inginious. It operates as an independent library responsible for managing the execution of student submissions. This includes creating, managing and deleting Docker containers in which student code is run. Containers provide a secure and isolated environment for executing untrusted code, ensuring the safety of the system by preventing malicious code to access or affect parts of the system. It also provides control over resource allocation, allowing administrators to limit the CPU, memory, and bandwidth allocated to each submission.

The platform features a plugin system that enables the extension of its functionality. The administrators are allowed to incorporate additional features with a flexible and modular framework for extending the functionalities. Integrating new problem types is a quite easy process that does not require modifications to the code base. This modular approach allows administrators to tailor INGInious to their specific use cases and preferences.

Automation is a central aspect of Inginious. Once the tasks and environments are set up by the instructors, the system automates the grading of the student code submissions. This automated grading eliminates the need for manual interventions, allowing the instructors to focus on other aspects of the course.

2.2 Parsons problems

A Parsons problem [3] is a type of programming puzzle where the solution is cut into multiple blocks and provided out of order, the task is to arrange these blocks to recreate a correct solution. Figure 2.1 shows a simple example of Parson problem. The blocks could represent code fragments, algorithm steps, math proofs, execution traces, or any related components. It serves an intermediary type of problem bridging the gap between multiple choice questions and coding exercises. Rather than starting with a blank page, students must analyze, manipulate and rearrange a predetermined set of blocks to assemble the correct solution.

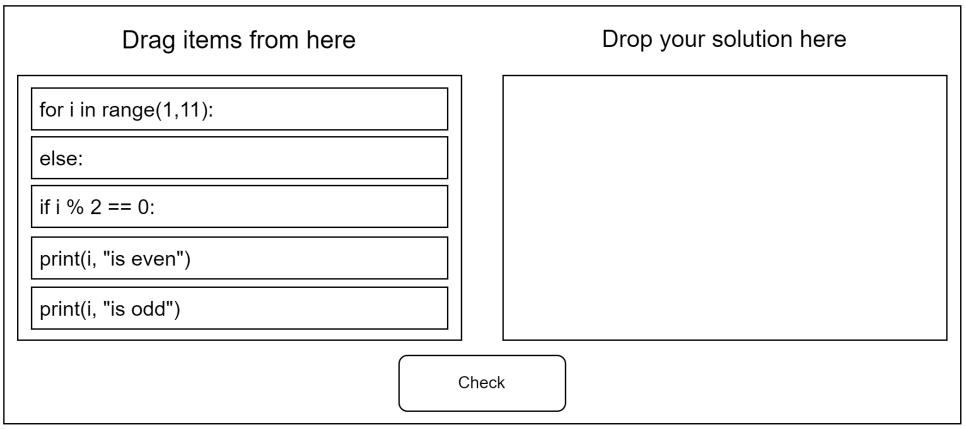


Figure 2.1: Example of Parsons problem

There exists different variations of Parsons problems. Figure 2.2 shows an example of these variations: distractors, faded, adaptive, optional. Basic problems are classic parsons problems but the other ones have some properties adding new mechanisms.

- **Distractors**: Distractors are incorporated into the block set to add an extra challenge for the student. These distractors may consist of irrelevant blocks that are intentionally misleading. The student have to figure out which blocks are meant to trick them and leave those out when they arrange the correct sequence.
- **Optional**: Students can encounter finite sets of distractors alongside a genuine code line. They must choose only one block of the set to include in their final solution.
- **Faded**: Some blocks of the problem contain blank spaces that need filling. Students must fill in these blanks while also rearranging the blocks in the correct order.
- **Adaptive**: When certain conditions are met, such as reaching a specific number of failures, it is possible to adjust the problem to make it slightly easier, providing guidance to students when they encounter difficulties solving it.

2.2.1 Benefits

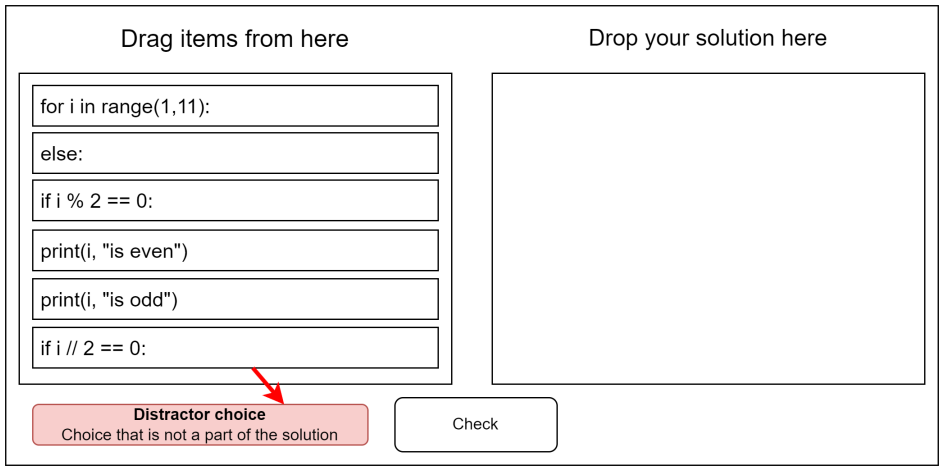
Now that we have established the mechanics of Parsons problems, it is crucial to explore the multitude of benefits such problems bring to the programming course environment. These exercises offer more than just another approach to learning. We will explore some benefits of Parsons problems, highlighting their ability to enhance student learning and comprehension.

Equivalent learning impact with improved time efficiency The expectations set for students in introductory programming courses involve mastering various skills simultaneously, including understanding formal languages, utilizing software tools, and problem-solving. These expectations are considered unrealistic due to the comprehensive nature of the requirements within a single course and the limited time available for students to adequately develop these skills. [1] While Parsons problems do not resolve that issue, it rather serve to reduce its magnitude. Parsons problems provide a time-efficient solution while maintaining an equivalent impact on learning skills.[8] This allows to cover a wide range of topics in a shorter amount of time.

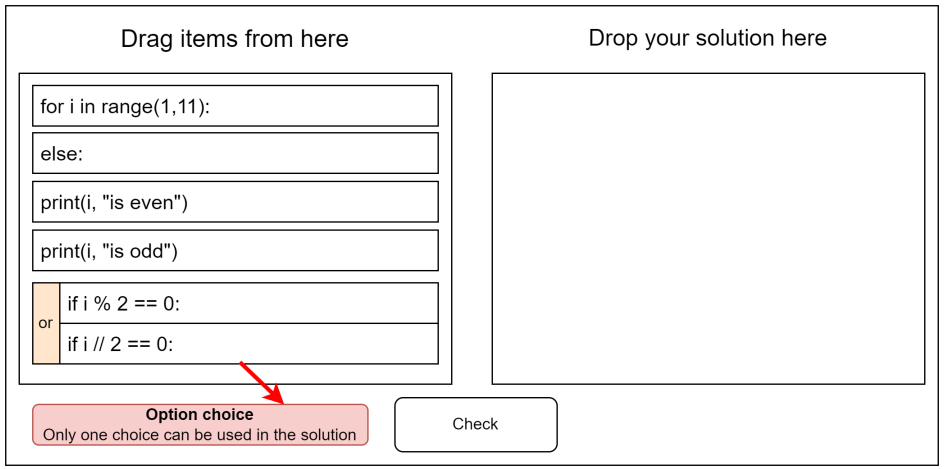
Reduced cognitive load Traditional programming tasks demand students to recall the syntax of the used programming language, making the cognitive load of novices programmer heavy. Problems which have heavy loads do not assist in learning [9]. The Parsons problems approach reduces the strain on students' working memory since they no longer need to memorize or worry about syntax rules and errors. This ultimately leads to a decrease in cognitive effort. Consequently, students can allocate more mental resources to work on fundamental concepts.

Active engagement These problems demand an active participation from students by challenging them to actively construct solutions and organize provided elements. This not only stimulates engagement but also encourages students to consider various approaches to problem-solving, promoting a deeper understanding of programming concepts among students.

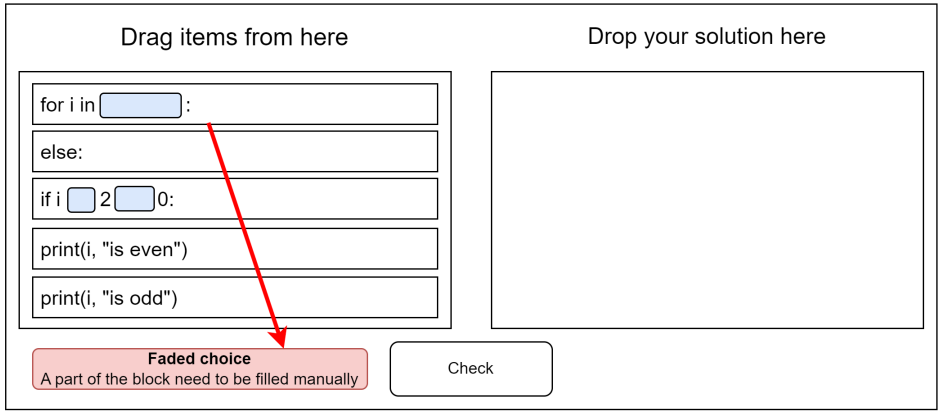
Understanding of programming concepts through examples Parsons Problems help students focus on understanding the logical structure of a program rather than the syntax. By rearranging code fragments, students have to think about the order of execution and the dependencies between different parts of the program, enhancing



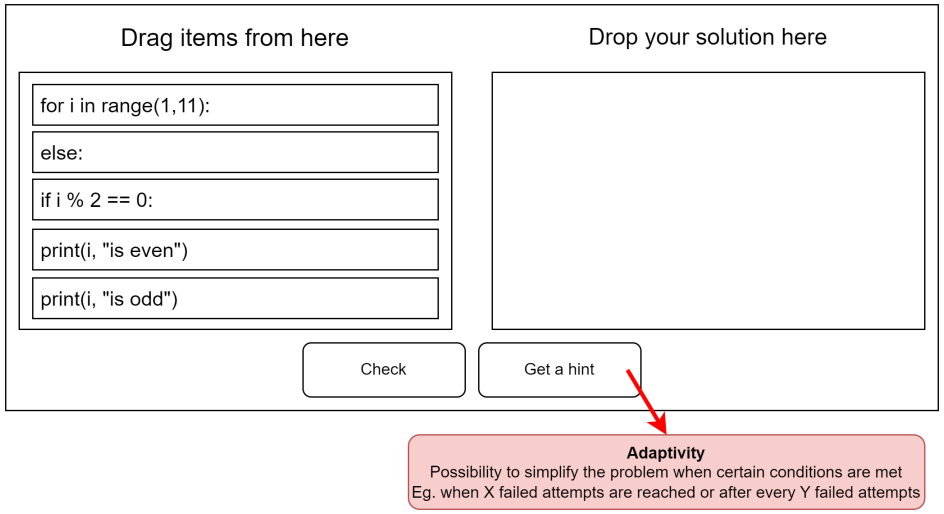
(a) Distractor



(b) Optional



(c) Faded



(d) Adaptive

Figure 2.2: Existing variations of Parsons problems

their comprehension and problem-solving skill. Parsons problems also help student to understand how to write good code through the examples provided in these problems. As students must read and analyze each block to determine where it fits, they gain insight into the structural organization and arrangement of well-written code and learn the ability to discern effective coding practices.

Highlight student's misconceptions Parsons problems can highlight common programming misconceptions [10] and errors. A programming misconception in an incorrect understanding or belief about a fundamental programming concept resulting from incomplete or incorrect information [11]. Having misconceptions can lead to problems in education by causing wrong understanding and slowing down the learning process. By addressing these misconceptions directly, students develop a deeper understanding of programming concepts and learn to identify and correct errors in their thinking.

Instant feedback Providing an instant feedback to students as they solve parsons problems offers several benefits. The main goal is to determine a precise location of the error [12]. They receive direct feedback on their errors through a comparison of their solution with the correct one. It encourages the self-assessment by prompting students to reconsider their problem-solving strategies, simultaneously reducing the frustration of failure as progress becomes evident over time.

2.2.2 Existing uses of Parsons problems

Using Parsons Problems is a strategy that aims at enhancing the teaching and learning process in computer science education. A study conducted in 2020 comparing multiple strategies [13] revealed that tutors at UCLouvain found this strategy engaging for students, giving them the ability to provide more examples in less time but requiring more preparation. The study showed that tutors appreciated the opportunity to tailor exercises to different difficulty levels and to stimulate discussions among students regarding programming structures [13]. Despite its benefits, tutors acknowledged the challenge of identifying the appropriate exercises for Parsons Problems and expressed a desire for more practical experience with the strategy [13]. Additionally, they suggested leveraging Parsons Problems as a means of assessing student knowledge or diagnosing systematic mistakes. Furthermore, tutors recognized the potential for an online platform to facilitate the implementation of Parsons Problems, making it more accessible and user-friendly for both tutors and students [13].

2.2.3 Existing implementation of Parsons Problems

Two notable existing software implementing Parsons problem are the JSParsons [3] library and the integration of Parsons problems within the Runestone Academy platform [14]. Each of these implementations offers unique features and advantages for facilitating the integration of Parsons problems into course materials.

JSParsons is a JavaScript library that facilitates the integration of Parsons problems into websites. With JSParsons, web developers can seamlessly embed Parsons problems directly into their platform. It was the first to propose two-dimensional Parsons problems and it is considered as the most influential technology delivering Parsons problems. [4] This library relies on Python for its implementation and runs the code to verify the solution, it possesses the ability to translate Python code into Java or pseudocode blocks.

Python	for i in range(5):
Java	for (int i = 0; i <5; i++) {
Pseudocode	FOR i from 0 to 4 DO

The Runestone Academy solution differs somewhat: it is a platform for creating interactive ebooks primarily focused on computer science and programming and it incorporates Parsons problems as a module within its courses, available exclusively on the platform. This solution does not execute the code blocks, making it compatible with various programming languages. While the faded variation is unavailable, it supports distractors, optional and adaptive variations.

The main difference between these two implementations is the grading method. JSParsons provides execution-based feedback with two possibilities. Solution correctness is checked either by executing unit tests or by conducting a variable check after the execution of the student's code. The approach of Runestone is to offer line-based feedback, where the sequence of blocks, arranged according to the order the student placed them, is compared to the sequence of the actual solution. One study compared both grading methods and found that each has its own issues and benefits, with neither being considered superior to the other [4, 15].

	JSParsons	Runestone
Indentation	yes	yes
Distractor variation	yes	yes
Optional variation	no	yes
Faded variation	yes	no
Adaptive variation	no	yes
Supported languages	Python, Java, Pseudocode	any
Grading system	execution-based	line-based
Availability	JavaScript library	Module on the Runestone platform

2.2.4 Identified gaps in research

While the promising benefits of assessing cognitive abilities and problem-solving skills through Parsons problems are evident, significant research gaps persist. The gaps are signals of the necessity for further investigation. Initial studies suggest the efficiency of Parsons problems but a more robust set of evidence is required to confirm the findings. Additionally, certain domain of Parsons problem remain unexplored, presenting opportunities for future studies [4]. Here is a comprehensive compilation based on the gaps identified by a Parsons literature review [4] conducted in 2022:

1. Research on Parsons problems focuses on introductory courses, leaving a gap in understanding their effectiveness in more advanced algorithm-focused classes.

2. There is a lack of investigation into how class size impacts the benefits of Parsons problems, with smaller classes potentially offering more direct instructor support, it could outcomes differently than larger classes.

3. The influence of learners' prior programming backgrounds on the utility of Parsons problems remains underexplored.

4. Different learning management systems may impact the delivery and effectiveness of Parsons problems, necessitating a deeper understanding of these tools affordances.

5. Despite the known challenges of programming error messages for novices, little research examines how Parsons problems could address this issue through improved error message design and usability.

6. Some thematic research areas related to Parsons problems occur infrequently or have not been studied in conjunction, suggesting gaps in understanding their interplay and implications.

7. Identity issues for underrepresented learner groups are largely unexplored in the context of Parsons problems, highlighting a gap in understanding how diverse student backgrounds intersect with learning outcomes.

8. Replication and multi-institutional studies are needed to validate the effectiveness of Parsons problems across various educational contexts and populations, ensuring robustness and generalizability of findings.

9. There is a lack of studies examining the impact of different variations of Parsons problems on learning outcomes, insufficient comparative studies impede understanding whether specific variations of Parsons problems offer benefits or present challenges.

Chapter 3

Problem and objectives

3.1 Problem statement

The existing challenges within introductory computer science (CS1) courses underscore the need of solutions to enhance student learning and reduce the perception of programming as a difficult domain. Although the Inginious autograding platform has been crucial in addressing a more student-centered approach to learning programming by automating the evaluation of code submissions, it does not include support for Parsons problems yet. Based on the advantages mentioned earlier (cf. section 2.2.1), the addition of such problem type could further enrich the learning experience for CS1 students and provide instructors with additional resources to enhance course delivery and assessment. The integration of Parsons problems have to be done into the Inginious platform due to the widespread use of this platform among computer science students at UCLouvain. Inginious is the primary autograding platform used for the practice part of CS courses. The introduction of Parsons problems into the platform could improve the teaching process for both students and instructors by maintaining a uniform platform for all types of problems. Furthermore, the integration of Parsons problems does not only benefits immediate educational needs (cf. section 2.2.1) but could also present opportunities for advancing research in the field of parsons problems. By providing such problems to an accessible platform used by the UCLouvain, it could facilitate future research needed to fulfil research gaps identified [4] for Parsons problems.

3.2 Objectives

1. Develop a plugin for the Inginious platform that incorporates Parsons problems as a new type of problem available for students. The primary objective is to enrich the learning experience by providing constructive feedback to students. Simultaneously, we prioritize simplicity in task creating by ensuring an intuitive interface for the instructors.

2. Create tasks examples using Parsons problems based on a curated inventory of programming language misconceptions [10]. This will highlight the capability of Parsons problems in addressing misconceptions students may have.

3. Evaluate the usability, potential teaching enhancements, and perceived learning benefits from the perspective of instructors of the CS1 programming course at UCLouvain by providing scenarios and conducting a survey.

4. Examine the implications of integrating this plugin as a means to fill the identified research gaps surrounding Parsons problems.

Chapter 4

Methods

This chapter details the design, implementation, and application of our plugin. It highlights its features, interfaces and the steps taken to build a demo course.

4.1 Design

4.1.1 Features

One of the main goal in developing this plugin is to make it as complete as possible, enabling instructors to tailor problems effectively to meet students' needs. But also offer the opportunity to fulfil research gaps (cf. section 2.2.4). To achieve this, we have implemented a wide range of options. Here is a description of the features currently supported by the plugin.

Problems variations As mentioned earlier (cf. section 2.2), there exist multiple variations of Parsons problems. The plugin actually support these variations:

- **Distractor**: It is possible to create distractors (cf. section 2.2) on the problem configuration page from any existing choice. Initially, the distractor will have the same content as the choice it was derived from. The person configuring the problem can then modify the content as desired. Copying the content from a genuine block is a time-saving measure, as distractors often resemble the correct options. Additionally, it is possible to create multiple distractors from the same choice.
- **Optional / Paired distractors**: After creating a distractor, it can be paired with the genuine block it was derived from. This way, students will have the option to choose between the distractor(s) and the real choice but will only be able to use one. Once a choice is selected and placed in the solution, the other options can no longer be moved. Multiple distractors can be paired with a single genuine choice.
- **Adaptive**: It is possible to activate problem adaptivity by configuring three parameters. When adaptivity is enabled, the first hints will sequentially disable any distractors, if present. Following this, choice blocks will merge to make problem resolution easier. The first parameter determines the number of attempts allowed before the first hint appears, the second sets the number of attempts required between two hints, and the third establishes the minimum block limit to halt fusion.

Feedback One of the purpose of using Parsons problem is the ability to give instant feedback to students. Efforts have been made to provide extensive configurability for

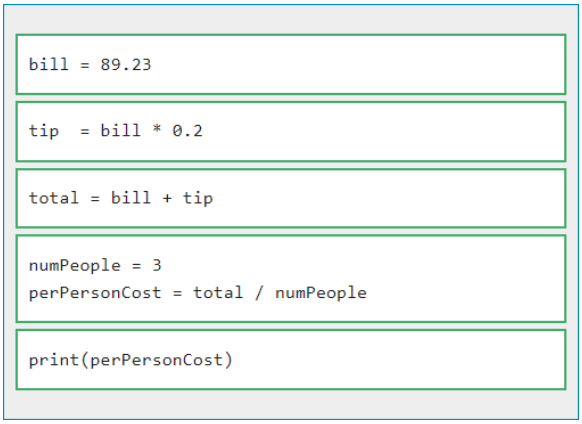
feedback, aiming for as comprehensive feedback options as possible, all of which are configurable.

- **Custom global message**: It allows you to compose a global message for each problem, whether the resolution is successful or a failure.
- **Custom message per block**: You have the option to include a failure or success message for each individual block. When a block is correctly placed, it will display the success message, regardless of whether the problem has failed overall. Otherwise, the failure message will be shown.
- **Ranged grading feedback**: It provides a grading percentage for the problem, indicating the proportion of correct blocks in the student's solution relative to the total number of blocks in the correct solution.
- **Length feedback**: Feedback is provided when the student's solution is either too long or too short compared to the correct solution length.
- **Colored borders**: There are three distinct settings, each providing varying levels of feedback to help students understand their mistakes. Figure 4.1 shows an example of these settings.
  - Disabled: This setting provides no colored feedback. Essentially, no visual clues are given to indicate correctness or incorrectness of choices in the solution sequence.
  - Partial: incorrectly positioned choices are highlighted with a red border, while correctly placed choices are bordered in green. A choice is considered correctly placed if it is a part of the longest increasing subsequence (LIS) within the solution sequence with a correct indentation.
  - Complete: Choices that are in the exact position as in the solution are highlighted in green. Choices that are nearly correct, meaning they are part of the longest increasing subsequence (LIS) within the solution sequence, are marked with an orange border. Incorrect choices are highlighted in red. Additionally, choices with incorrect indentation are marked with a red badge.

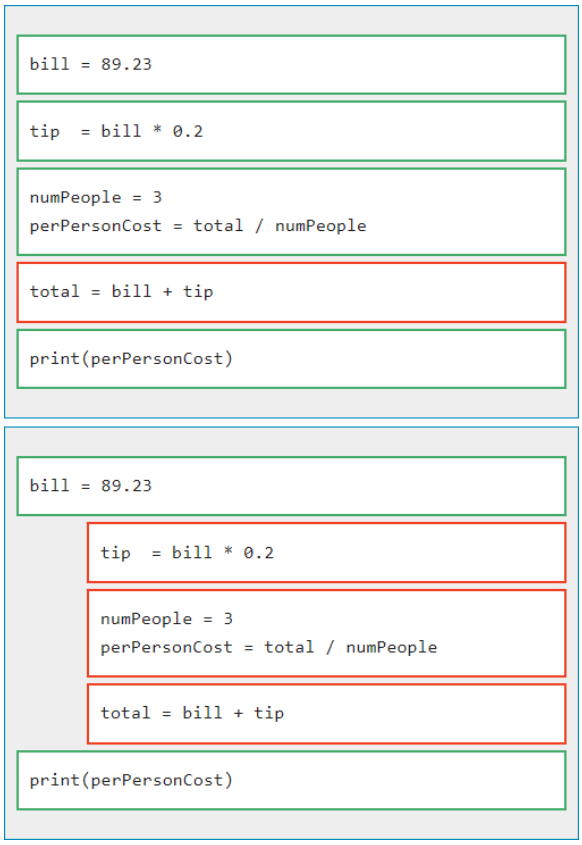
Easy back-up and importation of problems The plugin supports both export and import features to accelerate problem configuration and simplify the process of creating backups. To export a configured problem, a simple "Export to File" button is available on the configuration page. Clicking this button initiates the downloading of a JSON file containing all the previously configured data for the problem. For importing problems, users can use the "Generate from File" button. Clicking this button prompts a file picker dialog to appear. Upon selecting a JSON file, all inputs are automatically filled based on its data. For other file types, each line is parsed, and a choice is created with the content of each line.

4.1.2 Interface

Special attention has been devoted to the interface on both the task and configuration pages. The aim is to minimize confusion and streamline both the configuration and problem-solving processes as much as possible.



(a) Correct solution



(b) Partial setting

(c) Complete setting

Figure 4.1: Examples for colored feedback

Task interface The task interface has been deliberately kept simple. At the top, users can find the name and context of the task. Below, two lists are displayed side by side (or stacked if the screen is too narrow). The first list presents all available choices that can potentially lead to a solution. The second list starts empty and must be filled with the choices in the correct order, accomplished via drag and drop from the first list. Once completed, users can submit their answers using the button to receive the feedback. Figure 4.2 shows a wireframe of the task interface.

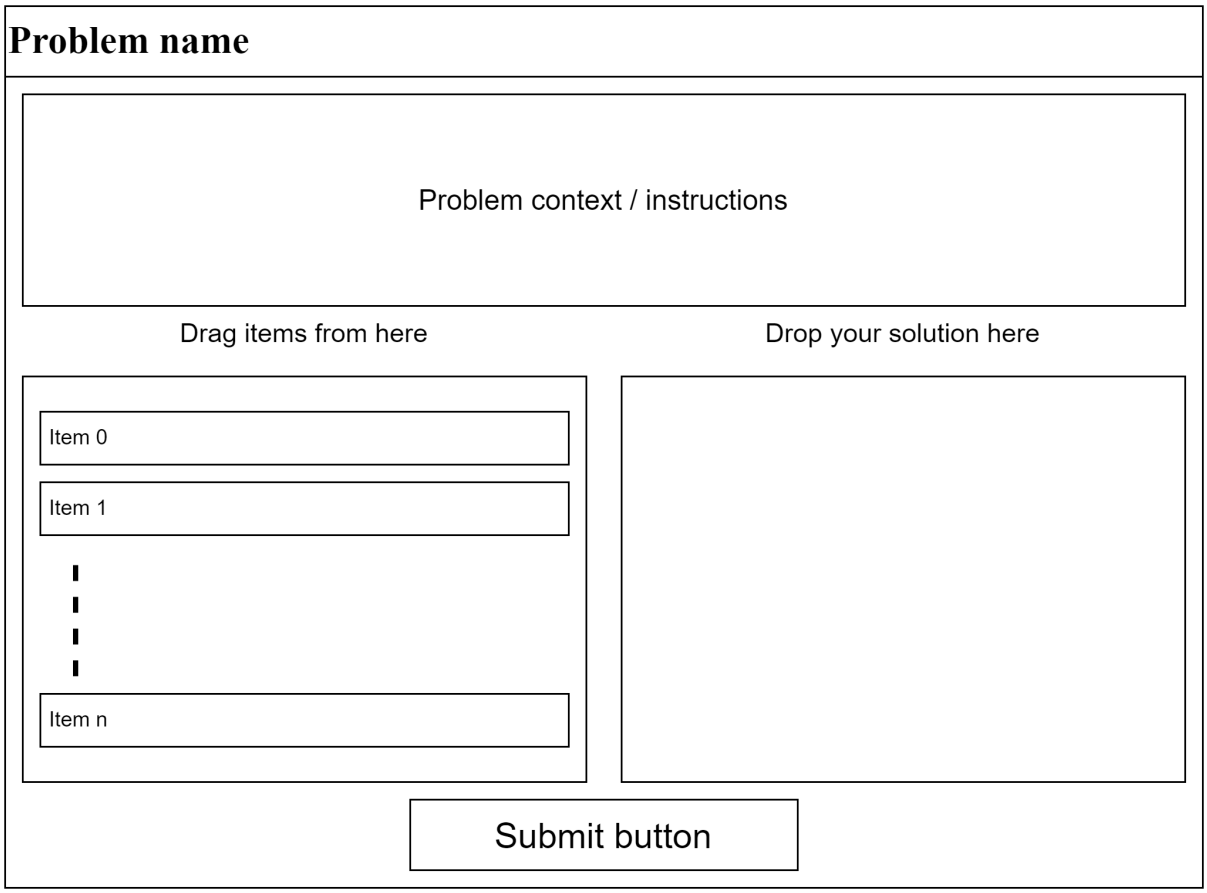


Figure 4.2: Task interface wireframe

Problem configuration interface The interface begins with input fields for the name and context. Below this, there are options that globally affect feedback, such as success message, failure message, indication level, ranged grading, length feedback, and adaptive problem settings. Following this, there is a checkbox to enable indentation of the blocks. Below the checkboxes is the main area, with a large space dedicated to configuring the correct solution. Within this space, multiple choices can be created. These choices are draggable, allowing users to rearrange and indent them as needed to configure the correct solution for students. Adjacent to this, there is another area designated for distractors. Distractors are not part of the solution and are kept separate from the correct solution during configuration. Unlike choices, distractors are not draggable within this area. Additionally, next to any choice (including distractors), there is a "+" button. Clicking this button opens a window that allow to configure individual block feedback, create distractors (from any choice), or pair distractors (from any distractor). Figure 4.3 shows a wireframe of the config interface.

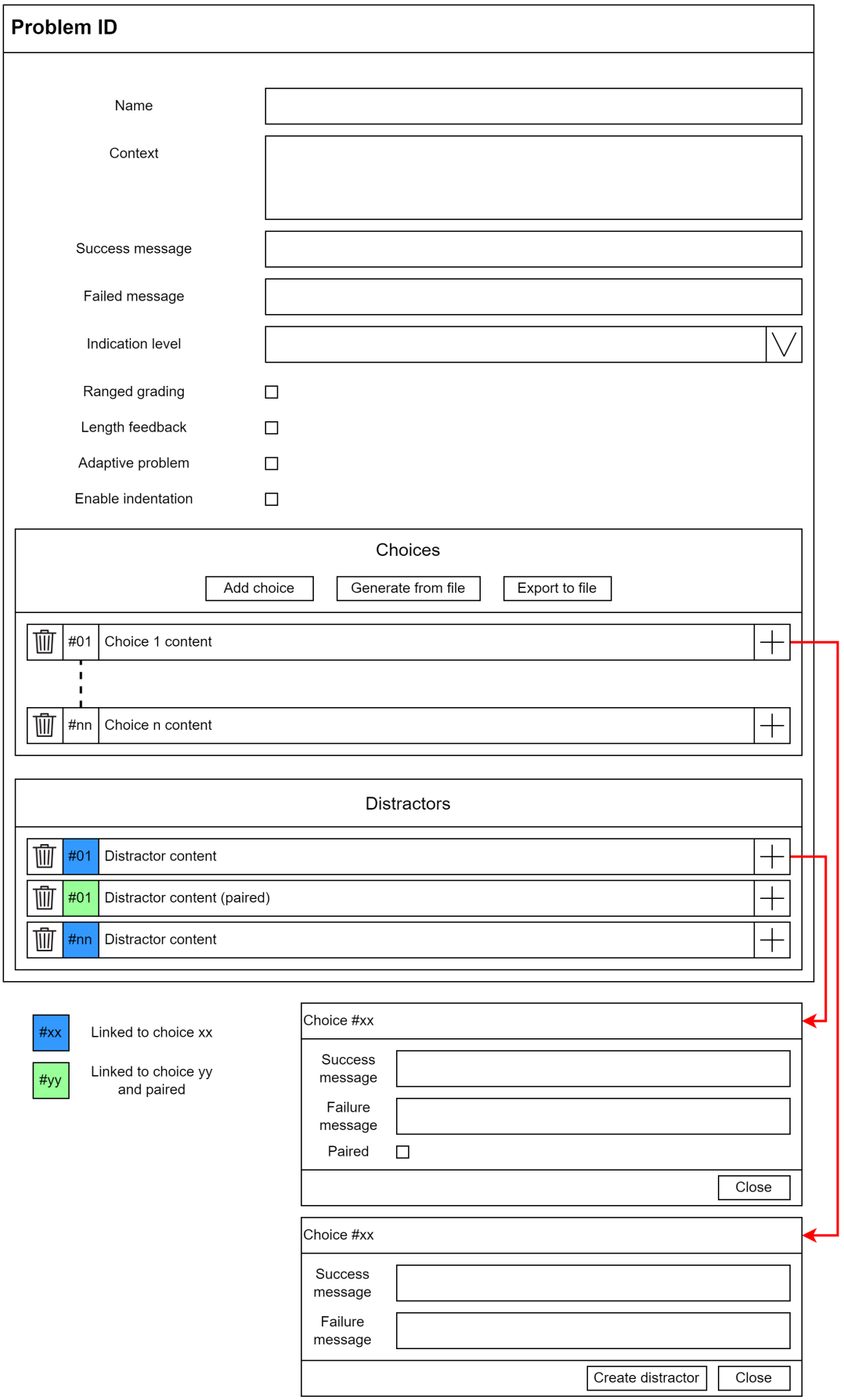


Figure 4.3: Configuration interface wireframe

4.2 Implementation

Inginious, being an adaptable platform, provides the flexibility to integrate new problem types through the development of custom plugins. To incorporate Parsons problems in the platform, a new plugin needs to be developed. This section will provide an overview of the process involved in creating such a plugin on the platform. Additionally, it will outline two primary implementation approaches, offering insights into how Parsons problems will function once integrated into the platform.

The plugin is available on GitHub[1] for those who want to use it or explore the code to see how it works.

4.2.1 Developing new problem types on Inginious

Developing new problem types for Inginious is a relatively easy process, as it requires no modifications to the platform's base code. Specifically, the process involves designing the interface for the problem, defining how the problem's data is stored and presented to students, and establishing the grading mechanism. This entire process can be completed without the need to handle database operations or any other webserver-related tasks.

Language and framework used by the platform Below are the technologies used by Inginious for its operations.

- **Flask**: A Python web framework, handling HTTP requests, routing, and dynamic content rendering. Its simplicity and flexibility make it suitable for developing web applications.
- **Bootstrap**: A front-end framework, ensuring consistency and responsiveness in the user interface. It provides CSS and JavaScript components for creating sleek and user-friendly interfaces.
- **MongoDB**: A database system, offering a document-oriented structure for flexible and scalable data management.
- **Docker**: Docker is used to encapsulate student-written code during automatic grading. It provides secure and isolated execution environments. (cf. section 2.1)
- **HTML templates and JavaScript scripts**: HTML templates and JavaScript scripts drive the user interface.

Plugin structure The plugin structure is straightforward, consisting of four essential file types that collectively define the functionality and presentation of problem types on the platform. Figure 4.4 shows the interactions between all the files that are part of the plugin.

- **Python files**: These files define how the configured problem data is saved, how it is displayed on both the task and configuration pages, and how it is graded automatically.
  Two abstract classes have to be implemented:
  - The Problem class encapsulates the problem description, typically represented by a dictionary, and defines essential back-end methods. For instance the problem description might be structured as follows:

[1] `https://github.com/WhippedCoconut/INGInious-problems-parsons`

{"header": "What is the answer to... ?", "answer": 42}.
The most important methods that need to be implemented are the following:

* ***parse_problem()***: Called when the problem is saved from the configuration page, it parses the raw data of the problem gathered from the configuration page.
* ***___init___()***: Called right after the problem parsing, it initializes the problem using the parsed data.
* ***check_answer()***: Called when a student submits a solution, it checks the answer and generates the feedback data that needs to be displayed to the student.

– The DisplayableProblem class contains all the templating methods for the interface. The methods that need to be implemented determine which HTML template should be displayed for the task or the configuration page. They also specify which data from the problem needs to be displayed on these pages. For Parsons problem, the task page displays only the content of the blocks, while the configuration page shows all previously completed and saved input.

- **HTML templates**: Two templates are required, one for the task interface, which is the one seen by the students, and another for the problem configuration interface. An additional optional template may be required if a template needs to be injected multiple times into the configuration interface. For example, a choice template might be necessary if the configuration for a new problem type requires the addition of multiple choices. This is particularly useful for Parsons problems, where the configuration might need an undefined number of blocks for each problem. The "add choice" button (cf. section 4.1.2) injects an HTML template for every choice added.
- **JavaScript files**: These files manage the functionalities of the problem. For example, Parsons problems necessitate an interactive drag-and-drop interface. These scripts manage these interactions, along with displaying feedback and other functionalities.
- **Optional CSS files**: These files are used to create new CSS classes or override predefined ones by Bootstrap.

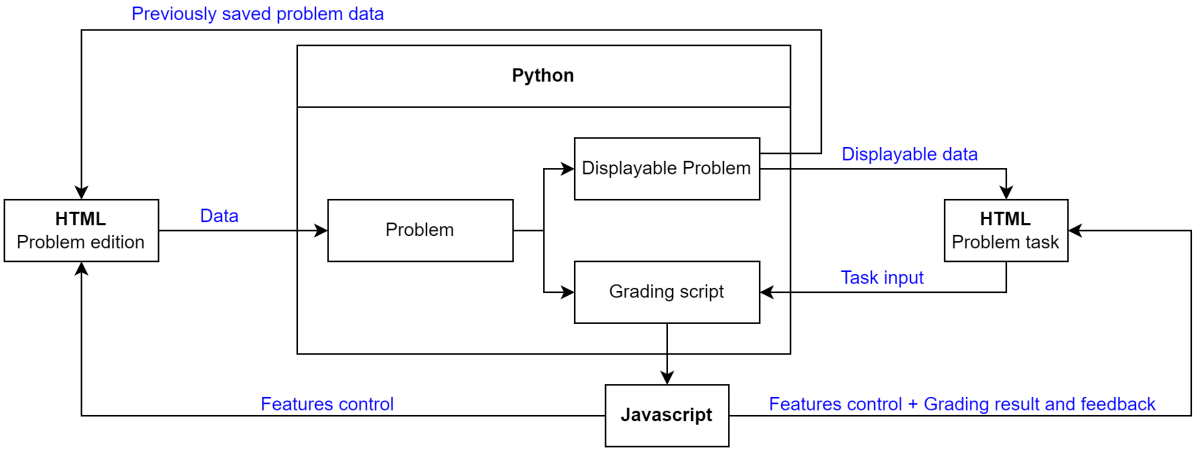


Figure 4.4: Interaction between plugin files

4.2.2 Grading system

In our integration, we opted for line-based grading over execution-based grading for two primary reasons. Firstly, execution-based grading would necessitate instructors to

write additional grading scripts, increasing the time required for problem configuration. Secondly, line-based grading facilitates easier detection of misplaced blocks, simplifying the implementation of colored border feedback, a feature we aim to support. However, line-based grading does come with its drawbacks. One of the most challenging aspects is accommodating multiple correct solutions during grading. Additionally, detecting compilation or execution errors can be difficult. Despite these challenges, we determined that line-based grading would be the better fit for this integration.

The grading is based on a longest increasing subsequence algorithm (LIS[2]). It seeks to find a subsequence within a given sequence where the elements are sorted in ascending order and the subsequence is as long as possible. This subsequence does not have to be unique. The problem can be solved in O(n log n) time complexity. Figures 4.5 shows an example of how the algorithm works.

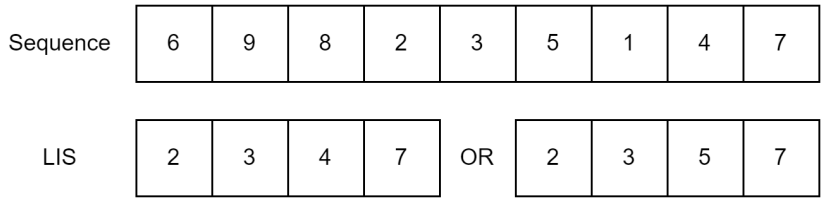


Figure 4.5: Longest increasing subsequence

Problem input The input is structured as a JSON object with two attributes: 'lines' an array that defines the order of blocks placed by the students, and 'indent,' which specifies the indentation level for each block. In this structure, 'lines[block_id]' denotes the position, and 'indent[block_id]' denotes the amount of indentation for the block identified by 'block_id' in the student solution. If a block is not a part of the student's solution, the position is set to -1.

Grading execution A sequence is generated from the arrangement of blocks and analyzed using the LIS algorithm to determine the longest sequence in the student's solution relative to the correct sequence. If the algorithm finds multiple LIS, it selects the first one found. Indentation is evaluated independently, with each block directly compared to the correct indentation. Subsequently, a global grade is computed, and each block is assigned a value to facilitate individual feedback. The potential values are:

- **0**: The block is precisely in the correct position.
- **1**: The block is precisely in the correct position but wrongly indented.
- **2**: The block is a part of the LIS.
- **3**: The block is a part of the LIS but wrongly indented.
- **4**: The block is wrongly placed.
- **5**: The block is wrongly placed and wrongly indented.

Having six distinct values for each block simplifies the process of offering detailed feedback. It also allows for classifying blocks into different categories, providing a wider range of options for varying levels of feedback. For example, it is possible to categorize blocks into two classes. The first class would include only the blocks with a value of 0, while the second class would include all other blocks. In this case, blocks will be categorized

[2] `https://en.wikipedia.org/wiki/Longest_increasing_subsequence`

as correct if they have the exact placement with the correct indentation, or incorrect otherwise. Alternatively, categorizing into four classes is also an option: three for blocks that are correctly indented based on their positioning (good, near-good, wrong) and one for incorrectly indented blocks (bad-indent).

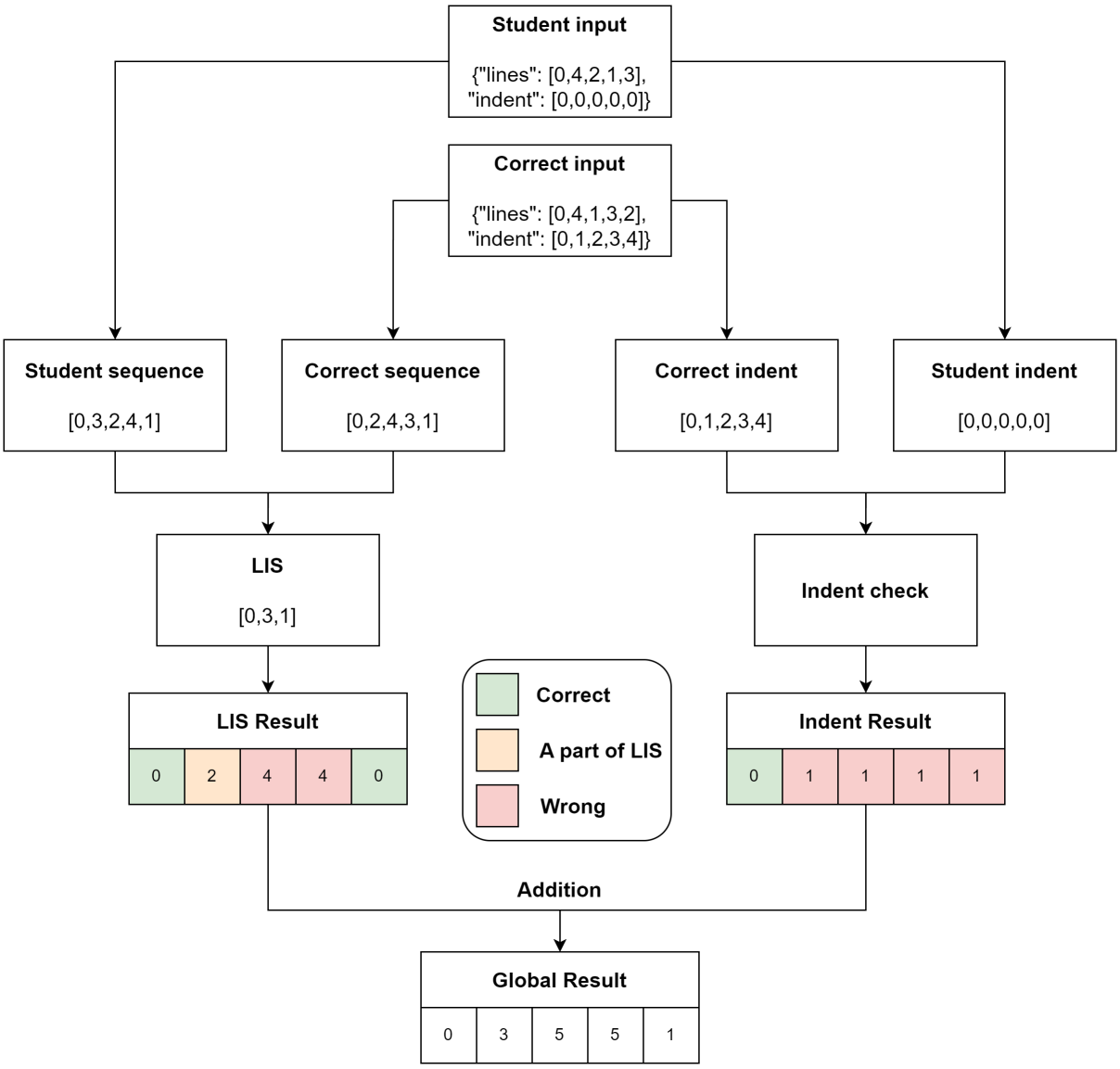


Figure 4.6: Grading execution

4.2.3 Drag and drop

Parsons problems involve an interactive drag-and-drop functionality, allowing users to sort and indent a list of blocks. Since we couldn't find any JavaScript libraries capable of handling sortable lists with indentation, we opted to implement the drag-and-drop functionality ourselves. This approach not only enabled us to implement the desired features such as distractors, paired distractors, or adaptive problems more easily but also gave us greater flexibility and control over the implementation process.

The implementation of this functionality is quite simple, it is built upon jQuery and the HTML5 Drag and Drop API. Each block has dragstart and dragend events to determine which block is currently being dragged. Each list has a dragover event to determine where the block should be placed when it is dropped. Additionally, it automatically handles the problem input update that changes when a block is moved.

4.3 Setting up a demo course

When integrating a new problem type into an existing platform, such as Parsons problems, it's crucial to provide comprehensive documentation for instructors interested in using it for their course material. In our case, this entails setting up a course[3] on Inginious, divided into two parts. The first part serves as a tutorial on creating tasks incorporating Parsons problems within Inginious. The second part consists of a compilation of examples designed to address common student programming misconceptions in Python, providing educators with practical insights and resources to enhance their teaching experience. These misconceptions are drawn from an available and curated inventory[4] [10].
The course is available directly on Inginious at `https://inginious.info.ucl.ac.be/course/parsons-demo`

4.3.1 Tutorial

The tutorial is structured to guide instructors in creating Inginious tasks incorporating Parsons problems, organized into four sections, each addressing different aspects of Parsons problem configuration.

1. **Introduction to Parsons problems**: This section clarifies Parsons problems for instructors who may not be familiar with them, explaining their concept and how students should approach solving them.

2. **Create a simple problem**: Instructors are introduced to the fundamental steps required to create a basic Parsons problem, focusing on essential elements without delving into advanced features such as customized feedback or additional problem variations.

3. **Feedback options**: Instructors explore various settings related to feedback, empowering them to design more complex problems with feedback features.

4. **Distractors**: The final section covers the inclusion of distractors and paired distractors.

4.3.2 Examples based on misconceptions

In the second part of the course, an Inginious task is designed for each Python misconception listed in the available inventory[5], with the exception of two. These two exceptions, namely "DefferedReturn" and "NoReservedWord," are not supported by the plugin. This limitation arises from the plugin's decision to utilize a line-based grading system (cf. section 4.2.2). The content of the blocks is neither read or execute, it is then impossible to find reserved words or return statements.

[3] `https://inginious.info.ucl.ac.be/course/parsons-demo`
[4] `https://progmiscon.org/misconceptions/Python/`
[5] `https://progmiscon.org/misconceptions/Python/`

Chapter 5

Validation

This chapter describes the validation process for integrating Parsons problems into Inginious, emphasizing the evaluation of usability and identification of areas for improvement. This is achieved through instructional scenarios and feedback gathered via a survey.

5.1 Validation methodology

The approach of this validation is mainly to gather feedback from key individuals who are likely to use the plugin. This includes teachers, course assistants, and tutors involved in the first-year programming course. The primary goal is to assess if the integration is at a stage where instructors can seamlessly incorporate Parsons problems into their CS1 courses. Additionally, the feedback will help identify any necessary modifications or additions to meet the instructors' needs. Furthermore, this process may uncover any unnoticed bugs that require fixing. To collect this feedback, we've prepared instructions detailing validation scenarios along with a survey that participants must complete. This chapter details only the design of the validation. The results and analyses will be discussed in a future chapter. (cf. chapter 6)

5.2 Validation Scenarios

The instructions given to the participants consist of scenarios and additional practical guidance needed to complete them. There are four different scenarios in total. The first scenario requires participants to act as students, while the remaining three are from the perspective of an instructor. Participants are also asked to follow the tutorial provided in the demo course to learn how to incorporate Parsons problems into Inginious before completing the instructor scenarios. The goal of the instructions is to explore all the features and get an overall understanding of the plugin.

- **Student scenario**: Two Inginious tasks must be completed. The participant is instructed to deliberately fail the first attempt by arranging the blocks in a specified incorrect manner. Next, the participant must solve the problem using the feedback obtained from the initial attempt so that he can experience the feedback given to students. This scenario serves as an introduction to Parsons problems and also gathers feedback on the task interface and the feedback mechanism.
- **Instructor scenario 1**: Participants are asked to create a task based on a given Python code, add some distractors, and configure the feedback. This scenario is useful for guiding participants through the entire configuration interface.

- **Instructor scenario 2**: This scenario requires some creativity. Participants are asked to configure a task that addresses a specific programming misconception, ensuring that students having that misconception are identified. This scenario demonstrates to instructors how integrating Parsons problems can help address programming misconceptions.
- **Instructor scenario 3**: In this scenario, participants are asked to create a single task containing two Parsons problems. The first problem will be configured by importing a given JSON file, while the second will be configured by importing a given Python file, which requires additional configuration compared to JSON. This scenario demonstrates to participants that a single task can contain multiple problems and showcases the import feature working with different file types.

The complete instructions given to the participants are available in the appendix A.

5.3 Survey Design

Participants receive both instructions and a survey to gather their opinions about the plugin. The survey is designed so that participants complete specific tasks from the instructions and then immediately respond to the corresponding section of the survey. This approach ensures that answers are not influenced by future sections. The survey consists of six sections, each focusing on a specific topic. Each section includes statements rated on a scale from 1 (strongly disagree) to 5 (strongly agree), along with one open-response question for additional feedback.

1. **Interface clarity**: This section evaluates the clarity of the interface provided to the students.
2. **Tutorial clarity**: This section assesses the clarity of the tutorial for learning how to create Parsons problem tasks on Inginious.
3. **Usability**: This section evaluates the usability of the plugin. Feedback, gathered through the System Usability Scale [16], will help determine how easy or difficult it is for participants to use the plugin interface and features.
4. **Teaching enhancement**: This section evaluates how the plugin of Parsons problems enhances teaching methods, aiming to determine how well it assists instructors in delivering course content.
5. **Potential learning impact on students**: This section explores the potential of Parsons problems to facilitate programming learning for students based on participants' experiences, seeking insights on whether such problems effectively aid in learning programming.
6. **Missing features and modifications**: This section gathers insights on any features participants believe are missing from the integration and those that require modifications to better meet their teaching needs.

The full questionnaire is available in the appendix B.

5.4 Threats to validity

As several factors may limit the validity of the validation process, it is important to list them to provide a clear understanding of the potential constraints and considerations influencing the interpretation of the results.

Timing constraints Due to timing issues, first-year students had already completed their programming courses for the year. Consequently, it was impossible to gather data with real students. As a result, the section originally intended to assess the "Learning Impact on Students" was renamed "Potential Learning Impact on Students." This change reflects the speculative nature of the feedback gathered, as it is based on participants' opinions rather than real student situations.

Choice of participants As students were not involved in the validation process, the participants were limited to the teaching staff of the first-year programming course. While these individuals offer valuable insights as potential users of the plugin, their perspectives may not fully represent the diverse range of experiences and needs that students might have. Also, this part of participants is more limited in size (5 assistants/professors and 3 tutors, for a total of 8 participants) than our pool of students should have been.

Exclusion of adaptive Parsons problems The validation process did not include the assessment of the adaptive variation of Parsons problems. This exclusion comes from the fact that adaptive problem feature was not yet implemented during the validation period. Consequently, the feedback gathered may not fully account for the potential benefits or challenges associated with adaptive Parsons problems.

Despite these limitations, the validation process serves as a structured approach to gather feedback on the integration. Future iterations of the validation should include a broader range of participants, such as students, and assess additional features like the adaptive variation

Chapter 6

Results of the survey

This chapter presents the feedback gathered from the survey conducted during the validation process. Each section corresponds to a specific area of the survey: interface clarity, tutorial clarity, usability, teaching enhancement, potential learning impact on students, and missing features and modifications. The survey included 8 participants from the first-year programming course, consisting of 5 assistants/professors and 3 tutors.

6.1 Task interface clarity

In the interface clarity section, participants assessed the ease of understanding the interface and feedback mechanisms. Figure 6.1 shows that all participants found the interface clear enough to easily solve Parsons problems, indicating strong overall satisfaction with the interface design. The written feedback received from the plugin had mixed reviews: a significant portion disagreed, suggesting room for improvement in clarity. The colored borders feedback was generally well-received, with most participants finding it comprehensible. However, some participants suggested from the open question that the colors need clearer explanations regarding their meanings, and considerations should be made for colorblind users, highlighting a potential area for refinement.

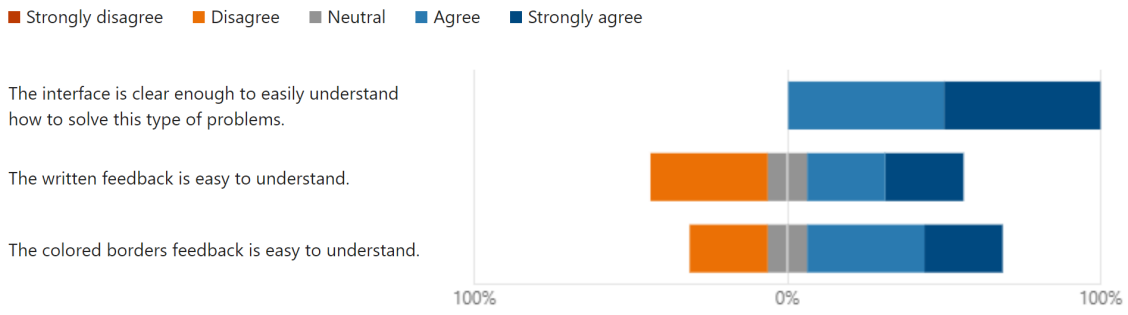


Figure 6.1: Task interface clarity results

6.2 Tutorial clarity

Figure 6.2 shows that participants expressed agreement that the tutorial was clear in explaining how to use the plugin and that they understood the purpose of each available option. This indicates a successful delivery of instructional content. However, participants' experience varied regarding encountering points of confusion during the tutorial. This highlights the importance of refining tutorial content to address potential areas of confusion. Despite this, the majority of participants felt confident in their ability to use the new

problem type after completing the tutorial, suggesting that, overall, it contributed positively to their learning and proficiency.

Feedback from the open question also mentioned the need to reorder some sections of the tutorial, which could be a contributing factor to the points of confusion encountered.

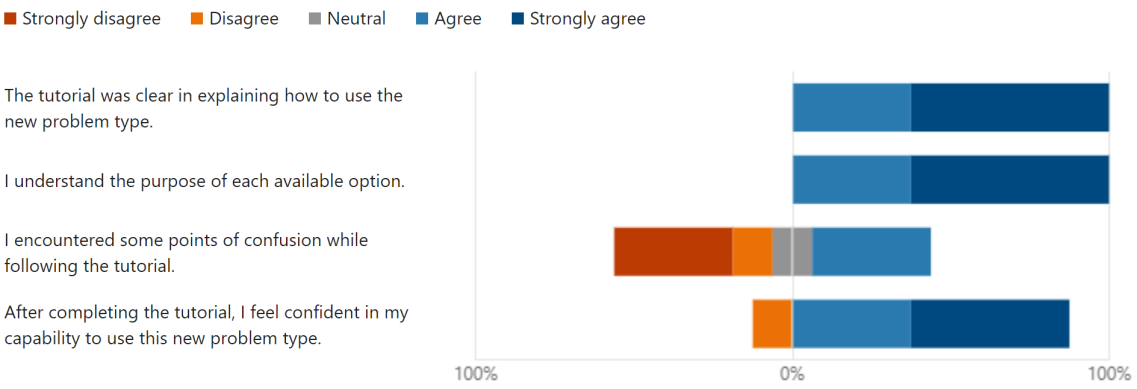


Figure 6.2: Tutorial clarity results

6.3 Usability

The System Usability Scale (SUS) [16], is a recognized method for evaluating usability. SUS provides a standardized approach to measure the usability of a variety of systems, typically focused on software applications, by generating scores ranging from 0 to 100. Scores over 68 are considered better than the industry average [17]. There are a total of 10 questions, interleaving positive and negative senses. Ideally, Figure 6.3 should show odd questions to the right and even questions to the left. The plugin obtained a score of 80.625, denoting excellent usability as it surpasses the industry average. It reflects a highly user-friendly system [17].

Min score: 67.5 **Max score**: 92.5 **Average score**: 80.625

6.4 Teaching enhancement

In the teaching enhancement section of the survey, opinions differed on the ability for the plugin to enhance the design of tasks and to facilitate providing feedback to students. One reason for this is that Inginious allows providing feedback for any type of problem, meaning that the plugin does not necessarily streamline the provision of feedback since Inginious already does that correctly for other types of problems. But there was a consensus that the plugin effectively facilitates the customization of feedback and the adaptation to students' needs. Additionally, participants agreed on its time-saving benefits as well as the incorporation with their course materials. Overall, Figure 6.4 illustrates a positive reception of the plugin.

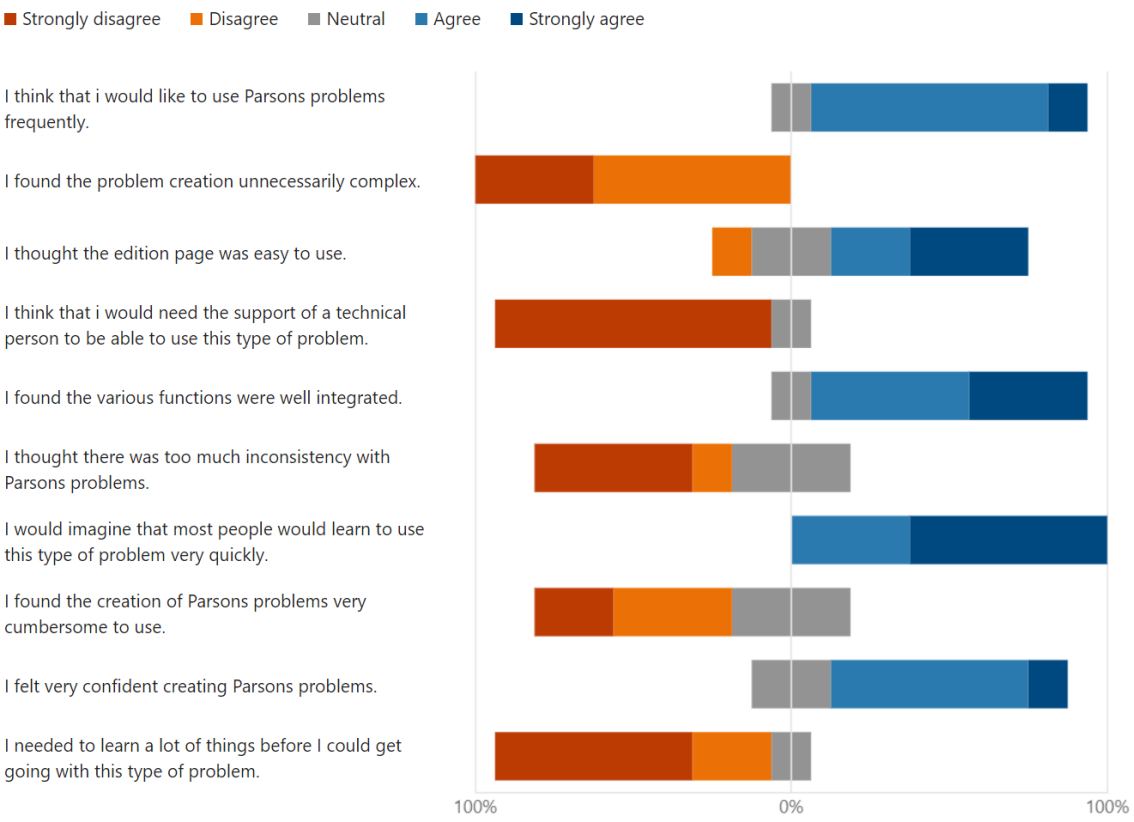


Figure 6.3: Usability results

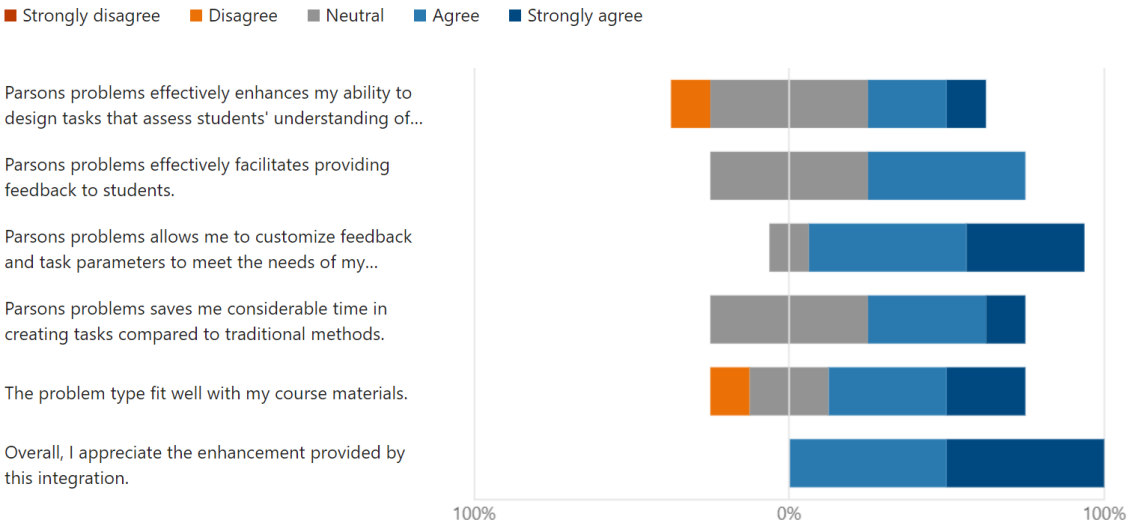


Figure 6.4: Teaching enhancement results

6.5 Potential learning impact on students

The results for this section of the survey show a relatively strong agreement, as shown in Figure 6.5, with the majority of participants acknowledging the benefits and no one expressing disagreement. The positive perception of Parsons problems suggests a potential to enhance various aspects of student learning. It is important to remember that these responses are speculative, based on participants' opinions rather than real student data.

Open-response feedback highlighted that while Parsons problems are not necessarily superior to traditional programming exercises, they can serve as a valuable complement.

Participants also emphasized that the feedback provided by Parsons problems is particularly impactful
responsibility of the problem designer to ensure effective feedback.

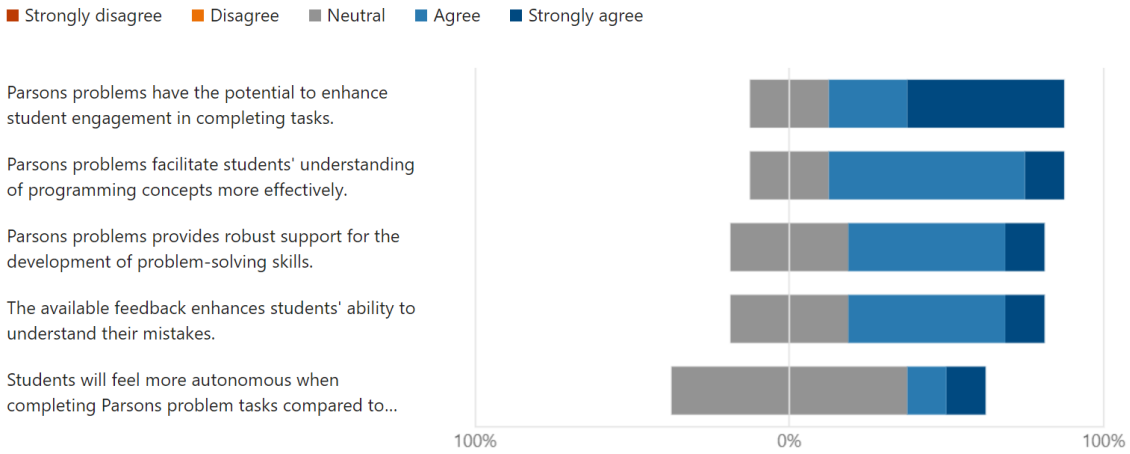


Figure 6.5: Speculative results of the impact of learning on students

6.6 Missing features and modifications

In the section where participants provided open-response feedback on desired changes or additions to the plugin, several valuable suggestions were retained for further development to enhance the plugin.

- The ability to create distractors without necessarily linking them to a correct choice.
- The option to configure problems with multiple possible solutions.
- A feature to test the code (at least in Python) to verify if the code written by a teacher when configuring the problem is correct.
- The option to generate automatic feedback based on a list of common programming misconceptions.
- An indicator on the task page to show if indentation is enabled, helping students understand that blocks can be indented.
- Modify the import feature to accommodate a textarea or a similar interface that enables users to paste entire code for import, alongside code and JSON files.

6.7 Overall conclusion of the results

The integration of Parsons problems on Inginious seems welcome from the teaching staff of the introductory programming course at UCLouvain. The feedback gathered reflects a generally positive reception of the plugin, with participants acknowledging its potential to enhance learning experiences and streamline certain aspects of teaching. While there are areas for improvement highlighted, the overall sentiment suggests that the integration of Parsons problems holds promise for improving programming education within the context of the course. Moving forward, addressing the identified areas of improvement and iteratively refining the plugin based on user feedback will be essential to maximize its utility and impact in the educational setting.

Chapter 7

Future work

With the successful integration of Parsons problems into the Inginious platform, attention turns to further enhancing the plugin and exploring the potential impact of this integration. While the plugin is currently fully functional and capable of incorporating Parsons problems into Inginious, there is always room for improvement. This chapter details additional features that could enhance the plugin's usability and completeness in configuring Parsons problems, suggests a better validation process to assess the integration's impact on CS1 programming courses, and discusses how the integration helps in closing research gaps in the field.

7.1 Further plugin additions

7.1.1 Interface

Translation Inginious offers methods to facilitate relatively straightforward translations for plugins. Considering that a majority of platform users, including both students and instructors, are native French speakers, implementing a French translation could be highly beneficial.

Drag and drop for mobile devices Currently, the drag-and-drop functionality does not work on devices that use only touch screens. To increase the accessibility of this integration, it should be made compatible with touch screen devices.

Indentation indication An indicator on the task page improves clarity by making students aware that indentation is active. This will enhance usability and increase student autonomy when completing Parsons problems.

7.1.2 Problem configuration

Creation of unlinked distractors The creation of distractors could be simplified by allowing the addition of distractors that are not directly linked to a genuine choice. This would make it easier to create distractors when a non-paired or non-resembling distractor is needed.

Pre-configured block placement Providing an option to place specific blocks directly into the solution when the task is loaded could be useful for some cases. For example, in a problem with multiple distractors, this feature could indicate that the pre-placed blocks are genuine. Additionally, function signatures could be pre-placed in the solution

to ensure the correct order for grading without requiring students to arrange them in a specific sequence.

Enhanced Import Functionality Introducing a textarea or similar interface for the import feature could significantly improve the importation. This addition would provide a convenient method to directly paste entire code snippets for import, eliminating the need to first save them to files.

Multiple solutions support The ability to save multiple solutions for a single problem would be very useful. It would simplify the design process by removing the requirement for problems to have only one possible solution. Additionally, by allowing multiple solutions to be saved and tagged as either optimal or working but not optimal, it could provide students with more feedback on program optimization.

Code verification Integrating a "check solution" feature could be highly beneficial for verifying the accuracy of configured solutions, minimizing student confusion resulting from unnoticed configuration errors. It is mandatory that this feature is language-specific, with essential support for common languages like Python, Java, JavaScript, and SQL, which are frequently taught in CS1 courses.

7.1.3 Code execution

Execution based grading Incorporating the option to choose between execution-based or line-based grading would significantly enhance the versatility of the integration. Instructors would gain the ability to design more complex problems that might necessitate unit testing.

Automated feedback generation By having the ability to execute the code, it will also be possible to perform pattern matching on the code submitted by the student, making it possible to detect common misconceptions. Automatic feedback could then be generated based on the misconceptions the student is still facing.

7.2 Completing the demo course

Once some of these additions have been implemented and deployed, the tutorial will need to be updated to keep the documentation on the plugin up to date and ensure that instructors understand how the new features work. For example, there is no tutorial part for the adaptive variation of Parsons problem since it was still under development during the tutorial's setup. This will also need to be added to the tutorial when the adaptive problems will be deployed.

More task examples should also be designed to help demonstrate the application of the new features in different scenarios.

7.3 Suggested validation process

Due to the timing constraints mentioned earlier (cf. section 5.4), the validation process requires a more comprehensive iteration to fully assess the beneficial impact of the integration on CS1 programming courses. Here is a suggestion for that future iteration, a brief description of the validation process that could have been conducted if there were no timing constraints.

The approach involves collecting data from students enrolled in the introductory programming course at UCLouvain, in collaboration with the teaching staff. The students will be divided into two groups, each with an equal number of participants and a balanced gender distribution. The first group, following the current programming course, will serve as the control group. This control group will provide a basis for comparison with the second group, which will follow a modified course incorporating Parsons problems. These problems will be used to introduce each programming concept and address common misconceptions. The course duration spans one semester, and the final exam will remain the same for both student groups. With the progress of students gathered during the semester and the exam results, a comparison is possible across multiple metrics such as student dropout rate, exam success rate, or average number of submissions needed for students to complete their tasks/exercises. Additionally, it may be possible to quantify and compare the average number of misconceptions that persist over time. This comparison should provide sufficient insights to discern any potential learning impact on students. A questionnaire, similar to the one conducted in the actual validation of this work (cf. section 5.3), will also be completed by the teaching staff. Having used the integration for a complete semester, the staff would possess a broader perspective on whether it enhances their teaching methods or not.

7.4 Closing the research gaps

Integrating Parsons problems into the Inginious platform opens up significant opportunities for future research, particularly in addressing previously identified gaps in the field (cf. section 2.2.4). This integration provides a foundation for conducting studies that can close these gaps.

Support of multiple problem variations Our integration supports four distinct variations of Parsons problems: normal, distractor, optional and adaptive. This could address the gap concerning the lack of studies examining the impact of different variations of Parsons problems on learning outcomes (gap 9). By providing these variations, our integration enables instructors to explore and compare the effectiveness of different types of Parsons problems.

Available all around the world Inginious is freely available and accessible on GitHub. This open access ensures that multiple institutions worldwide can adopt and use Inginious, and more specifically, our integration of Parsons problems. This widespread availability could address multiple gaps. By facilitating adoption across various educational contexts and institutions, it allows for replication and multi-institutional studies (gap 8), thereby validating the effectiveness of Parsons problems across diverse settings, including different class sizes (gap 2). Furthermore, this wide accessibility can contribute to a better understanding of underrepresented learner groups (gap 7).

Support for more advanced algorithms While the integration was primarily developed for introductory computer science courses, it is also designed to support any code, including more complex algorithms. This possibility addresses the gap concerning the limited understanding of the effectiveness of Parsons problems in more advanced (gap 1), algorithm-focused classes. By enabling the creation and use of Parsons problems for complex algorithms, the integration facilitates research into how these problems can aid in the learning and understanding of advanced topics.

A new tool among the others By introducing a new tool for creating Parsons problems, added to the existing array of systems and tools available for this purpose. This could address the gap concerning the impact of different learning management systems on the delivery and effectiveness of Parsons problems (gap 4). By providing an additional tool, our integration contributes to a deeper understanding of how various platforms and systems can enhance or hinder the use of Parsons problems.

Chapter 8

Conclusion

The integration of Parsons problems into the Inginious platform for the introductory programming course at UCLouvain has been positively received by the teaching staff, who acknowledge its potential to enhance learning experiences and teaching efficiency. While certain areas for improvement have been identified, the overall impression indicates promise for enhancing programming education within the course. Moreover, this integration introduces a valuable tool that could contribute to addressing gaps identified in existing literature, providing a platform for future research on Parsons problems.

However, it is important to remind certain limitations in the validation process. The absence of real student involvement restricted feedback to teaching staff perspectives, potentially overlooking student experiences. Additionally, the exclusion of adaptive Parsons problems from the validation process limits the comprehensive evaluation of the plugin's potential benefits.

Looking ahead, efforts will focus on refining the plugin based on the received feedback and addressing the identified areas of improvement. Future validation should prioritize involving students to ensure a better evaluation and consider the development of adaptive Parsons problems.

Bibliography

[1] A. Luxton-Reilly, "Learning to program is easy," in *Proceedings of the 2016 ACM Conference on Innovation and Technology in Computer Science Education*, ser. ITiCSE '16. New York, NY, USA: Association for Computing Machinery, 2016, p. 284–289. [Online]. Available: https://doi.org/10.1145/2899415.2899432

[2] K. Overby, "Student-centered learning," *ESSAI*, vol. 9, p. Article 32, 2011. [Online]. Available: https://dc.cod.edu/essai/vol9/iss1/32

[3] P. Ihantola and V. Karavirta, "Two-dimensional parson's puzzles: The concept, tools, and first observations," *Journal of Information Technology Education. Innovations in Practice*, vol. 10, p. 119, 2011.

[4] B. J. Ericson, P. Denny, J. Prather, R. Duran, A. Hellas, J. Leinonen, C. S. Miller, B. B. Morrison, J. L. Pearce, and S. H. Rodger, "Parsons problems and beyond: Systematic literature review and empirical study designs," in *Proceedings of the 2022 Working Group Reports on Innovation and Technology in Computer Science Education*, ser. ITiCSE-WGR '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 191–234. [Online]. Available: https://doi.org/10.1145/3571785.3574127

[5] S. Mishra, S. Balan, S. Iyer, and S. Murthy, "Effect of a 2-week scratch intervention in cs1 on learners with varying prior knowledge," in *Proceedings of the 2014 Conference on Innovation & Technology in Computer Science Education*, ser. ITiCSE '14. New York, NY, USA: Association for Computing Machinery, 2014, p. 45–50. [Online]. Available: https://doi.org/10.1145/2591708.2591733

[6] G. Derval, A. Gego, P. Reinbold, B. Frantzen, and P. Van Roy, "Automatic grading of programming exercises in a mooc using the inginious platform," *European Stakeholder Summit on experiences and best practices in and around MOOCs (EMOOCS'15)*, pp. 86–91, 2015.

[7] G. Derval and A. Gego, "Inginious grading platform for students," Available from https://github.com/UCL-INGI/INGInious, 2014.

[8] B. J. Ericson, L. E. Margulieux, and J. Rick, "Solving parsons problems versus fixing and writing code," in *Proceedings of the 17th Koli Calling International Conference on Computing Education Research*, ser. Koli Calling '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 20–29. [Online]. Available: https://doi.org/10.1145/3141880.3141895

[9] J. Sweller, "Cognitive load during problem solving: Effects on learning," *Cognitive Science*, vol. 12, no. 2, pp. 257–285, 1988. [Online]. Available: https://www.sciencedirect.com/science/article/pii/0364021388900237

[10] L. Chiodini, I. Moreno Santos, A. Gallidabino, A. Tafliovich, A. L. Santos, and M. Hauswirth, "A curated inventory of programming language misconceptions,"

in *Proceedings of the 26th ACM Conference on Innovation and Technology in Computer Science Education V. 1*, ser. ITiCSE '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 380–386. [Online]. Available: https://doi.org/10.1145/3430665.3456343

[11] A. Swidan, F. Hermans, and M. Smit, "Programming misconceptions for school students," in *Proceedings of the 2018 ACM Conference on International Computing Education Research*, ser. ICER '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 151–159. [Online]. Available: https://doi.org/10.1145/3230977.3230995

[12] D. Parsons and P. Haden, "Parson's programming puzzles: a fun and effective learning tool for first programming courses," in *Proceedings of the 8th Australasian Conference on Computing Education-Volume 52*, 2006, pp. 157–163.

[13] O. Goletti, K. Mens, and F. Hermans, "An analysis of tutors' adoption of explicit instructional strategies in an introductory programming course," in *Proceedings of the 22nd Koli Calling International Conference on Computing Education Research*, 2022, pp. 1–12.

[14] B. J. Ericson and B. N. Miller, "Free and interactive ebooks for computing courses with new types of parsons problems and support for peer instruction," in *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 2*, ser. SIGCSE 2023. New York, NY, USA: Association for Computing Machinery, 2023, p. 1180. [Online]. Available: https://doi.org/10.1145/3545947.3569631

[15] J. Helminen, P. Ihantola, V. Karavirta, and S. Alaoutinen, "How do students solve parsons programming problems? – execution-based vs. line-based feedback," in *2013 Learning and Teaching in Computing and Engineering*, 2013, pp. 55–61.

[16] J. Brooke, "Sus: A quick and dirty usability scale," *Usability Eval. Ind.*, vol. 189, 11 1995.

[17] J. Sauro, *A Practical Guide to the System Usability Scale: Background, Benchmarks & Best Practices*. Measuring Usability LLC, 2011. [Online]. Available: https://books.google.be/books?id=BL0kKQEACAAJ

Appendix A

Validation instructions

This appendix refers to the validation instructions discussed in section 5.2

Thank you for taking the time to participate in validating my master thesis, your involvement is truly appreciated and will greatly contribute to shaping the outcome of my thesis.

As part of this process, here are the materials provided for your review:

- A PDF document (the one you are currently reading) containing the instructions to follow in order to complete the review.
- A course in Inginious, a clone of the original, where you are the administrator.
  - You will solve exercises in the first part of the instructions.
  - You will next follow a tutorial on how to create Parsons problem tasks on Inginious.
  - After completing the tutorial, you will also create a few new tasks.
- A questionnaire: click here.
- Two files used in the scenarios: "partition.json" and "quicksort.py".

A.1 Parsons problem from the student POV

Step 1 - Proceed with the following scenario.

Student scenario - As a student enrolled in the first-year programming course, you are assigned various tasks on Inginious related to the material covered each week. You still have to complete two tasks this week.

1. Open the inginious course.
2. Complete the first task.
   (a) Open the task called “Student scenario 01”.
   (b) Try to solve the exercise by submitting the following code using the available blocks:

```
bill = 89.23
numPeople = 3
perPersonCost = total / numPeople
tip = bill * 0.2
total = bill + tip
print(perPersonCost)
```

(c) Use the given feedback to understand your mistake and solve the exercise.

3. Complete the second task.

(a) Open the task called “Student scenario 02”.

(b) Try to solve the exercise by submitting the following code using the available blocks:

```
def is_even(node):
    if node is not None:
        if node.value % 2 == 0:
            return True
        else:
            return False
```

(c) Use the given feedback to understand your mistake and solve the exercise.

Step 2 - Fill out the initial section of the questionnaire regarding the students interface clarity.

A.2 Parsons problem from the instructor POV

Step 1 - Open the Inginious course.
Step 2 - Finish all four tasks in the tutorial section
Step 3 - Fill out the second section of the questionnaire regarding the tutorial.
Step 4 - Proceed with the following three scenarios.

Instructor scenario 1 - As an instructor of the first-year programming course, you’re tasked with creating exercises for the second part of the course, which focuses on if-else statements. Your goal is to craft engaging and challenging tasks that reinforce the students’ understanding of this fundamental programming concept. You are about to create the first exercise on this topic through a Parsons problem.

1. Create an Inginious task under your name called “FirstnameLastname-1” and configure a simple Parsons problem.

(a) Consult the tutorial section labeled "Create a simple Parsons problem" if necessary.

(b) Configure a simple Parsons Problem containing each line of the following code:

```
def classify(number):
    if number % 2 == 0:
        print(f"{number} is even.")
    else:
        print(f"{number} is odd.")
```

2. Add complexity to the resolution of this exercise using distractors.

(a) Consult the tutorial section labeled "Distractors" if necessary.

(b) Create a distractor for the line containing the 'if' statement.

(c) Modify its content so students may eventually mistaken the distractor with the genuine block.

3. Customize the feedback.

(a) Change the indication level to "partial".

(b) Activate ranged grading.

(c) Add a failure message to the previously created distractor explaining why this choice is not correct in the solution.

(d) Test the problem to verify if the feedback works.

Scenario instructor 2 - As an instructor of the first-year programming course, you're tasked with creating exercises for the part of the course that focuses on loops. One common misconception among novice programmers is the belief that the body of an if-statement executes repeatedly as long as the condition holds. To address this misconception and ensure clarity among students, you aim to create an Inginious task using a python Parsons problem.

1. Take note of the misconception. Click here

2. Create a new Inginious task under your name called "FirstnameLastname-2".

3. Design a Parsons problem in a way that effectively confronts the discussed misconception, using all necessary features to highlight the misunderstanding to students.

4. Verify whether the problem you have designed is working as intended.

Scenario instructor 3 - As an instructor of the first-year programming course, you're tasked with creating exercises for the part of the course that focuses on sorting algorithms. You want to create an Inginious task employing Python Parsons problems that will ask students to complete a Quicksort algorithm. You also want to encourage students to divide the problem they have to solve into multiple sub-problems. That's why you will ask students to solve the partition part of Quicksort in the first Parsons problem and implement the Quicksort algorithm using a partition() function in a second problem. Since a Parsons problem on the partition function has been previously created, you will use the backed-up problem for this segment.

1. Create a new Inginious task under your name called "FirstnameLastname-3".

2. Create a first Parsons problem and import the partition problem.

(a) Add a parsons-type problem called "partition".

(b) Click on "generate from file" and select the file "partition.JSON".

3. Create another Parsons problem and import the code of Quicksort.

(a) Add a parsons-type problem called "quicksort".

(b) Enable the indentation of the problem.

(c) Click on "generate from file" and select the file "quicksort.py".

(d) Make sure the solution is correctly drawn.

(e) Change the feedback options according to your preferences.

4. Save the modifications and verify whether the problem is correctly configured on the student's page.

Step 5 - Complete the questionnaire to its conclusion.

Appendix B

Validation survey

This appendix refers to the survey discussed in section 5.3

B.1 Survey Preface

Thank you for agreeing to participate in our survey! Your insights are crucial for validating the integration of Parsons problems on the Inginious learning platform. This survey is divided into six sections to gather valuable feedback regarding the proposed integration of Parsons problems into the Inginious platform.

1. Students interface clarity (To be completed at Part 1 - Step 2)
2. Parsons problems tutorial clarity (To be completed at Part 2 - Step 3)
3. Usability Evaluation
4. Teaching Enhancement Evaluation
5. Potential learning impact on students
6. Missing features and modifications

Your honest feedback in each section is greatly appreciated for validating the integration and understanding the potential challenges and improvements needed. We truly appreciate you taking the time to participate and share your thoughts with us.

What is your role in the programming course?
Tutor/Course assistant/professor: _____

B.2 Student interface clarity evaluation

This section is designed for you to evaluate the clarity of the interface provided to the students. **This is the only section to complete for Part 1 - Step 2 of the instructions.**

Q1. The interface is clear enough to easily understand how to solve this type of problems.

- 1.Strongly Disagree — 5.Strongly Agree

Q2. The written feedback is easy to understand.

- 1.Strongly Disagree — 5.Strongly Agree

Q3. The colored borders feedback is easy to understand.

- 1.Strongly Disagree — 5.Strongly Agree

Q4. Please share any additional comments, suggestions, or feedback you have regarding the student interface.

- _____

B.3 Parsons problems tutorial clarity evaluation

This section evaluates the clarity of the tutorial provided for learning how to create Parsons problem tasks on Inginious. **This is the only section to complete for Part 2 - Step 3 of the instructions.**

Q1. The tutorial was clear in explaining how to use the new problem type.

- 1.Strongly Disagree — 5.Strongly Agree

Q2. I understand the purpose of each available option.

- 1.Strongly Disagree — 5.Strongly Agree

Q3. I encountered some points of confusion while following the tutorial.

- 1.Strongly Disagree — 5.Strongly Agree

Q4. After completing the tutorial, I feel confident in my capability to use this new problem type.

- 1.Strongly Disagree — 5.Strongly Agree

Q5. Please share any additional comments, suggestions, or feedback you have regarding your experience with the tutorial.

- _____

B.4 Usability Evaluation

This section is designed for you to evaluate the usability of the integration. Your feedback, gathered through the System Usability Scale (SUS), will help us understand how easy or difficult it is for you to use the interface and features.

Q1. I think that I would like to use Parsons problems frequently.

- 1.Strongly Disagree — 5.Strongly Agree

Q2. I found the problem creation unnecessarily complex.

- 1.Strongly Disagree — 5.Strongly Agree

Q3. I thought the edition page was easy to use.

- 1.Strongly Disagree — 5.Strongly Agree

Q4. I think that I would need the support of a technical person to be able to use this type of problem.

– 1.Strongly Disagree — 5.Strongly Agree

Q5. I found the various functions were well integrated.

– 1.Strongly Disagree — 5.Strongly Agree

Q6. I thought there was too much inconsistency with Parsons problems.

– 1.Strongly Disagree — 5.Strongly Agree

Q7. I would imagine that most people would learn to use this type of problem very quickly.

– 1.Strongly Disagree — 5.Strongly Agree

Q8. I found the creation of Parsons problems very cumbersome to use.

– 1.Strongly Disagree — 5.Strongly Agree

Q9. I felt very confident creating Parsons problems.

– 1.Strongly Disagree — 5.Strongly Agree

Q10. I needed to learn a lot of things before I could get going with this type of problem.

– 1.Strongly Disagree — 5.Strongly Agree

Q11. Please share any additional comments, suggestions, or feedback you have regarding your experience with Parsons problems.

– _____

B.5 Teaching Enhancement Evaluation

In this section, we'll evaluate how the integration of Parsons problems enhances teaching methods with the purpose of understanding if it will assist instructors in delivering course content.

Q1. Parsons problems effectively enhances my ability to design tasks that assess students' understanding of programming concepts.

– 1.Strongly Disagree — 5.Strongly Agree

Q2. Parsons problems effectively facilitates providing feedback to students.

– 1.Strongly Disagree — 5.Strongly Agree

Q3. Parsons problems allows me to customize feedback and task parameters to meet the needs of my students.

– 1.Strongly Disagree — 5.Strongly Agree

Q4. Parsons problems saves me considerable time in creating tasks compared to traditional methods.

– 1.Strongly Disagree — 5.Strongly Agree

Q5. The problem type fit well with my course materials.

– 1.Strongly Disagree — 5.Strongly Agree

Q6. Overall, I appreciate the enhancement provided by this integration.

– 1.Strongly Disagree — 5.Strongly Agree

Q7. Please share any additional comments, suggestions, or feedback you have regarding how Parsons problems could further enhance teaching methods.

– _____

B.6 Potential learning impact on students

In this section, we'll discuss the potential of Parsons problems to facilitate programming learning for students based on your experience. We're interested in your insights on whether such problems have the potential to aid students in learning programming effectively.

Q1. Parsons problems have the potential to enhance student engagement in completing tasks.

– 1.Strongly Disagree — 5.Strongly Agree

Q2. Parsons problems facilitate students' understanding of programming concepts more effectively.

– 1.Strongly Disagree — 5.Strongly Agree

Q3. Parsons problems provides robust support for the development of problem-solving skills.

– 1.Strongly Disagree — 5.Strongly Agree

Q4. The available feedback enhances students' ability to understand their mistakes.

– 1.Strongly Disagree — 5.Strongly Agree

Q5. Students will feel more autonomous when completing Parsons problem tasks compared to traditional coding tasks.

– 1.Strongly Disagree — 5.Strongly Agree

Q6. Please share any additional comments, suggestions, or feedback you have regarding how Parsons problems could further enhance student learning.

– _____

B.7 Missing features and modifications

This section is dedicated to gathering your insights on any features you believe are missing from the integration and those that require modifications to better meet your teaching needs.

Features changes - Do you identify any particular features that you believe require changes for an enhanced teaching experience?

_____

Features Suggestions - Please provide any suggestions for additional features you believe would enhance the functionality and effectiveness of Parsons problems.

UNIVERSITÉ CATHOLIQUE DE LOUVAIN
École polytechnique de Louvain
Rue Archimède, 1 bte L6.11.01, 1348 Louvain-la-Neuve, Belgique | **www.uclouvain.be/epl**