Erasmus+ Project No: 2018-1-PL01-KA203-050803

IO2: Data Exchange Format for Gamified Programming Exercises

Version 1.0

Document Manager	
Jakub Swacha	US

Document Version:	1.0	**Date:**	2020-02-03

Published Filename:	FGPE_IO2_Data_Exchange_Format_for_Gamified_Programming_Exercises.pdf
Intellectual Output	IO2
Storage location / link	

License

Disclaimer

The FGPE Consortium consists of the following partners

Participant no.	Participant organization name	Short name	Country
1	University of Szczecin	US	Poland
2	CRACS University of Porto	CRACS	Portugal
3	Aalborg University Copenhagen	AAU	Denmark
4	University of Napoli Parthenope	UNP	Italy

Revision history

Version	Author	Notes	Date
0.1	José C. Paiva (CRACS) Ricardo Queirós (CRACS) José Paulo Leal (CRACS)	Initial version of IO2: Data Exchange Format for Gamified Programming Exercises.	October 09, 2019
0.2	Sokol Kosta (AAU)	Revision of the document.	January 31, 2020
1.0	Jakub Swacha (US)	Technical corrections.	February 3, 2020

Table of Contents

1. Executive Summary

This document provides a description of the data formats defined for describing gamified programming exercises. Two distinct definitions are elaborated. One is a JSON format based on an existing XML dialect [1] for describing programming exercises. The other is a JSON format to design the gamification layer put on top of the programming exercises, creating a gamified programming curriculum.

These formats are a direct result of the work done during the second phase of the Framework for Gamified Programming Education project, consistent with the gamification scheme identified in the first phase. They constitute a necessary base for the development of further project outputs, primarily, for building the tools that will allow to create gamified programming courses adhering to them.

Section 2 provides more details on this document's background, purpose and scope. Section 3 presents the properties supported by the extended data format for describing programming exercises without gamification-related data, and how it differs from the base format [1]. Section 4 describes the data format for building a gamification layer for gamified programming courses.

2. Introduction

2.1 Background

The combined use of automated assessment, which provides fast feedback to the students experimenting with their code, and gamification, which provides additional motivation for the students to intensify their learning effort, can help pass the barrier of difficulty in learning programming. In such an environment, students keep receiving the relevant feedback no matter how many times they try (thanks to automated assessment) and their engagement is retained (thanks to gamification).

Learning programming relies on practicing. While there are a number of open software and programming exercise collections supporting automated assessment, up to this date, there are no available open collections of gamified programming exercises, no open interactive programming learning environments to support such exercises, and even no open standards for the representation of such exercises so that they could be developed by different educational institutions and shared among them.

Therefore, the primary objective of the Framework for Gamified Programming Education (FGPE) project is to provide a framework for application of gamification to programming education, including the necessary specifications (of the gamification scheme and the exercise definition format), collection of gamified exercises (for popular programming languages), and software (a toolset for editing exercises and an interactive learning environment for the students to solve them).

2.2 Purpose

To share gamified programming exercises among different universities and courses requires a common data format. Inclusion of elements such as the challenge definition, including the optional modifiers to adapt its difficulty, the interactive story layer, including the links to other exercises; and the award definition in terms of points, badges, and virtual items in the Data Exchange Format for Gamified Programming Exercises will allow the exchange of ready-to-use programming exercises along with the gamification-related data among different universities and courses, which makes it an important innovation in programming education with a high practical impact, as it will help save a lot of instructors' time that they would otherwise had to spend on defining the gamification rules themselves.

The purpose of this document is to present the developed data format. According to the best of our knowledge, there are several formats for defining programming exercises but none of them supports gamification-related data. Furthermore, the existing formats do not support all the different types of programming exercises identified in the first phase of this project. Hence, the authors decided to split the formatization of gamified programming exercises in two layers: a programming exercise layer and a gamification layer. The programming exercise layer extends an existing format [1] to support each of the previously identified types of programming exercises, without gamification-related data. The gamification layer is a completely new data format that couples with those programming exercises to create a gamified course curriculum.

2.3 Scope and related documents

This document covers the definition of the data interchange format for gamified programming exercises and courses. Section 3 presents the properties supported by the extended data format for describing programming exercises without gamification-related data, and how it differs from PExIL [1]. Section 4 describes the data format for building gamification layers to gamify programming courses.

As gamification is a rapidly developing field, the contents of this document are subject to change in future. Please consult the FGPE project website (http://fgpe.usz.edu.pl) for its up-to-date version.

This document is based on the Gamification Scheme for Programming Exercises document [6], which covers the relevant gamification concepts for programming education (published in May 2019).

This document does not cover the documentation of the tools supporting editing and conversion of gamified programming exercises, which is to be covered in the Tools Supporting Editing and Conversion of Programming Exercises document (scheduled for February 2020).

This document does not cover the documentation of the online platform providing gamified programming courses, which is to be covered in the Programming Learning Environment featuring Gamified Exercises document (scheduled for July 2020).

This document does not cover the presentation of the gamified programming exercises developed within the FGPE project, which is to be covered in the Programming Courses featuring Gamified Exercises document (scheduled for February 2021).

2.4 Acknowledgements

This document is a direct result of the work done within the Framework for Gamified Programming Education project supported by the European Union's Erasmus Plus programme (agreement no. 2018-1-PL01-KA203-050803).

Although the editors of this document are listed on page 3, its content reflects the results of the intellectual work of all the involved project team members:

- on behalf of University of Szczecin: Jakub Swacha, and Karolina Muszyńska,
- on behalf of CRACS University of Porto: Ricardo Queirós, José C. Paiva, and José Paulo Leal,
- on behalf of Aalborg University Copenhagen: Sokol Kosta and Reza Tadayoni,
- on behalf of University of Napoli Parthenope: Raffaele Montella.

3. Programming Exercises Format - YAPExIL

This section details the format developed for describing programming exercises - the **Y**et **A**nother **P**rogramming **Ex**ercises **I**nteroperability **L**anguage (YAPExIL) [3]. This format is partially based in the XML dialect PExIL [1], but (1) it is a JSON format instead of XML, (2) removes support for automatic test generation, and (3) supports different types of programming exercises (e.g., solution improvement, bug fix, gap filling, block sorting, and spot the bug).

YAPExIL can be broken down into four distinct facets: **metadata**, which contains simple properties providing information about the exercise; **presentation**, which relates to what is presented to the student; **assessment**, which encompass what is used in the evaluation phase; and **tools**, which includes any additional tools that the author may use in the exercise.

3.1 Metadata Facet

Table 3.1 Metadata Facet

Property	Description
ID	Universally Unique Identifier (UUID) of the exercise.
Title	Title of the exercise.
Module	Module in which the exercise is in (description of its main topic).
Author	Author of the exercise.
Keywords	Set of keywords of the exercise.
Type	The type of programming exercise to be solved. Possible values: BLANK_SHEET, EXTENSION, IMPROVEMENT, BUG_FIX, FILL_IN_GAPS, SORT_BLOCKS, SPOT_BUG.
Event	Event at which the exercise was created (if any).
Platform	Platform requirements (if any).
Difficulty	Difficulty of the exercise. Possible values: BEGINNER, EASY, AVERAGE, HARD, MASTER.
Status	Status of the exercise. Possible values: DRAFT, PUBLISHED, UNPUBLISHED, TRASH.
Creation Date	Date of creation of the exercise.
Last Update Date	Date of the last update to the exercise.

3.2 Presentation Facet

Table 3.2 Presentation Facet

Property	Description
Instructions	Instructions to teachers about the exercise.
Statements	The statement of the exercise to present to the student in the various languages.
Embeddables	Images, videos, and other files that can be embedded in the statement.
Skeletons	Part of a solution that is provided to the students.

3.3 Assessment Facet

Table 3.3 Assessment Facet

Property	Description
Templates	Part of a solution that wraps students' code without their awareness.
Libraries	Code libraries that can be used by solutions, either in compilation or execution phase.
Static Correctors	External programs that are invoked before dynamic correction to classify/process the program's source code.
Dynamic Correctors	External programs that are invoked after the main correction to classify each run.
Solutions	Solutions provided by the author(s) of the exercise.
Tests	Set(s) of test cases to validate if attempts are correct.

3.4 Tools Facet

Table 3.4 Tools Facet

Property	Description
Feedback Generators	External programs that generate the feedback to give to the student about his/her attempt to achieve a solution.
Test Generators	External programs that generate the test cases to validate a solution.

4. Gamified Programming Education Format - GEdIL

This section presents the data format for describing gamified programming course materials - the **G**amified **Ed**ucation **I**nteroperability **L**anguage (GEdIL) [4]. This data format aims to accomplish the requirements identified in a research work [2] conducted for this specific purpose. Those requirements are identified at two levels, exercise-level (those whose scope can be narrowed down to a single exercise) and course-level (those with a broader context).

However, the strategy followed was not a direct fulfilment, but rather a separation of what pertains to a programming exercise and what only makes sense in a gamified curriculum (i.e., the gamification layer). This means that programming exercises will still adhere to the YAPExIL definition, as exercise-level gamification requirements are met in a separate layer that links to non-gamified exercises. Hence, this modular approach allows the programming exercises to be reused without gamification (i.e., used in non-gamified contexts) or reused with different gamification layers.

4.1 Structure of the Format

GEdIL can be divided into five main definitions: **gamification layer**, which is the root of the definition and contains objects with a global context; **challenge**, which the basic unit of a gamified programming course (it can be a leaf challenge if it references exercise(s) or a branch challenge if it references other challenge(s)); **reward**, which is something given to users who achieve a certain goal; **rule**, which allows the definition of actions to be executed if a certain criteria is met; and **leaderboard**, which is an ordered set of metrics according to which players will be sorted.

4.1.1 Gamification Layer Definition

Table 4.1.1 Gamification Layer Definition

Property	Description
ID	UUID of the gamification layer.
Name	Name of the gamification layer.
Description	Description of the gamification layer.
Keywords	Keywords of the gamification layer.
Status	Status of the gamification layer. Possible values: DRAFT, PUBLISHED, UNPUBLISHED, TRASH.
Rules	Set of rules on the top-level.
Leaderboards	Set of leaderboards on the top-level.
Rewards	Set of rewards given on the top-level.
Challenges	Set of top-level challenges.

4.1.2 Challenge Definition

Table 4.1.2 Challenge Definition

Property	Description
ID	UUID of the challenge.
Name	Name of the challenge.
Description	Description of the challenge.
References	Reference to non-gamified exercises of this challenge.
Mode	The mode in which the programming exercise should be solved. Possible values: NORMAL, SHAPESHIFTER, SHORTENING, SPEEDUP, HACK_IT, TIME_BOMB, DUEL.
Mode Parameters	Parameters to adjust the challenge according to the mode (e.g., threshold number of lines, threshold execution time, or time of the bomb).
Locked	Is the challenge initially locked?
Hidden	Is the challenge initially hidden?
Difficulty	Difficulty of the challenge. Possible values: BEGINNER, EASY, AVERAGE, HARD, MASTER.
Feedback Generators	External programs that generate the feedback to give to the student when the challenge is complete.
Rules	Set of rules on the challenge-level.
Rewards	Set of rewards given on the challenge-level.
Leaderboards	Set of leaderboards in the challenge.
Children	Sub-challenges of this challenge.

4.1.3 Reward Definition

Table 4.1.3 Reward Definition

Property	Description

ID	UUID of the reward.
Name	Name of the reward.
Description	Description of the reward.
Kind	Type of reward. Possible values: POINT, BADGE, VIRTUAL_ITEM, COUPON, REVEAL, UNLOCK, HINT, MESSAGE.
Amount	Quantity of the reward (if applicable).
Unlockables	List of resources that get unlocked (if applicable).
Revealables	List of resources that get revealed (if applicable).
Hints	Messages that help solving the challenge (if applicable).
Congratulations	Messages to congratulate the user (if applicable).
Criteria	Conditions to obtain this reward.

4.1.4 Rule Definition

Table 4.1.4 Rule Definition

Property	Description
ID	UUID of the rule.
Name	Name of the rule.
Criteria	Conditions to activate this rule.
Actions	List of actions to execute (GIVE, TAKE, UPDATE, ...) if rule is activated (actions may have parameters).

4.1.5 Leaderboard Definition

Table 4.1.5 Leaderboard Definition

Property	Description

ID	UUID of the leaderboard.
Name	Name of the leaderboard.
Metrics	Set of metrics considered in this leaderboard, sorted by importance.
Sorting Orders	List with |metrics| elements, consisting of the sort direction (ASC or DESC) for that metric.

4.2 Requirements Fulfillment

The next sub-subsections describe how requirements of each specific concept are fulfilled.

4.2.1 Gamification Concepts Related to Course Organization

Table 4.2.1 Fulfillment of requirements from gamification concepts related to course organization

Concept	Fulfillment
Course Module	YAPExIL has a "module" field (see 3.1).
Exercise Type	YAPExIL has a "type" of exercise field (see 3.1).
Exercise Mode	GEdIL supports "mode" and "mode parameters" in Challenge (see 4.1.2). Furthermore, a Challenge may be just an envelope for a single exercise by referencing it.
Locked Content	GEdIL has a "locked" property in Challenge (see 4.1.2).
Secret	GEdIL has a "hidden" property in Challenge (see 4.1.2).
Difficulty Level	GEdIL has a "difficulty" property in Challenge (see 4.1.2). Challenge is a tree, hence every branch may have a different difficulty level.

4.2.2 Gamification Concepts Related to Goals Definition

Table 4.2.2 Fulfillment of requirements from gamification concepts related to goals definition

Concept	Fulfillment
Challenge	GEdIL has Challenge (see 4.1.2).
Requirements	Overall requirement is to solve the challenge, which includes solving the referenced exercises in a specific mode (see 4.1.2). Each exercise may define certain requirements through the Assessment Facet (see 3.3).

Quest	A Challenge in GEdIL may reference multiple exercises and define its rules and rewards (see 4.1.2).
Streak	Rules in GEdIL may be attached to the Course (see 4.1.1 and 4.1.4).
Record	Multiple Leaderboards may be created according to certain metrics and attached to the course (see 4.1.1 and 4.1.5).

4.2.3 Gamification Concepts Related to Definition of Rewards

Table 4.2.3 Fulfillment of requirements from gamification concepts related to definition of rewards

Concept	Fulfillment
Point	GEdIL defines a Reward with "type" (which can be POINT) and "amount" properties (see 4.1.3).
Level	A Gamification Layer in GEdIL can have attached Rules to deal with level progression (see 4.1.1 and 4.1.4).
Held Record	GEdIL defines a Leaderboard and means to attach them to Gamification Layer and Challenge (see 4.1.1, 4.1.2, and 4.1.5).
Current Rank	GEdIL defines a Leaderboard and means to attach them to Gamification Layer and Challenge (see 4.1.1, 4.1.2, and 4.1.5).
Badge	GEdIL defines a Reward with "type" (which can be BADGE) and "name" (to specify the badge) properties (see 4.1.3).
Virtual Item	GEdIL defines a Reward with "type" (which can be VIRTUAL_ITEM), "name" (to specify the virtual item), and "amount" properties (see 4.1.3).
Coupon	GEdIL defines a Reward with "type" (which can be COUPON), "name" (to specify the coupon), and "amount" properties (see 4.1.3).
Content Discovery	GEdIL defines a Reward with "type" (which can be REVEAL) and "revealables" properties (see 4.1.3).
Content Unlock	GEdIL defines a Reward with "type" (which can be UNLOCK) and "unlockables" properties (see 4.1.3).
Hint	GEdIL defines a Reward with "type" (which can be HINT) and "hints" properties (see 4.1.3).
Congratulations	GEdIL defines a Reward with "type" (which can be MESSAGE) and "congratulations" properties (see 4.1.3).

4.2.4 Types of Conditions Upon Which Rewards Are Granted

Table 4.2.4 Fulfillment of requirements of the types of conditions upon which rewards are granted

Concept	Fulfillment
Attempt	Rules in GEdIL may be attached to the Challenge and give a Reward (see 4.1.2 and 4.1.4).
Achievement	Rewards in GEdIL may be attached to the Challenge (see 4.1.2).
Failure	Rules in GEdIL may be attached to the Challenge and give a Reward on failure (see 4.1.2 and 4.1.4).
Progress threshold	Rules in GEdIL may be attached to the Gamification Layer and give a Reward (see 4.1.1 and 4.1.4).
Progress in competition	Leaderboards in GEdIL may be attached to the Gamification Layer (see 4.1.1 and 4.1.5).

4.2.5 Gamified Programming Exercise Types

Table 4.2.5 Fulfillment of requirements of gamified programming exercises types

Concept	Fulfillment
Blank sheet	YAPExIL has a “type” of exercise field, which can be BLANK_SHEET (see 3.1) and the Assessment Facet specifies the requirements (see 3.3).
Code extension	YAPExIL has a “type” of exercise field, which can be EXTENSION (see 3.1), a “skeleton” field to set the initial code (see 3.2), and the Assessment Facet specifies the requirements (see 3.3).
Code improvement	YAPExIL has a “type” of exercise field, which can be IMPROVEMENT (see 3.1), a “skeleton” field to set the initial code (see 3.2), and the Assessment Facet specifies the requirements (see 3.3).
Buggy code	YAPExIL has a “type” of exercise field, which can be BUG_FIX (see 3.1), a “skeleton” field to set the initial code with bugs (see 3.2), and the Assessment Facet specifies the requirements (see 3.3).
Fill-in the gap	YAPExIL has a “type” of exercise field, which can be FILL_IN_GAPS (see 3.1), a “skeleton” field to set the initial code with gaps, identified as `{{gap}}` (see 3.2), and the Assessment Facet specifies the requirements (see 3.3). See Appendix 1.
Mixed code	YAPExIL has a “type” of exercise field, which can be SORT_BLOCKS (see 3.1), a “skeleton” field which accepts multiple files - the blocks (see

	3.2), and the Assessment Facet specifies the requirements (see 3.3). See Appendix 2.
Show me	YAPExIL has a "libraries" field, which can have the API for the limited instruction set and allows to specify the requirements (see 3.3). Visual feedback can be generated by a tool as of YAPExIL definition (see 3.4). See Appendix 3.
Spot the bug	YAPExIL has a "type" of exercise field, which can be SPOT_BUG (see 3.1), a "skeleton" field to set the initial code with bugs (see 3.2), and the Assessment Facet specifies the requirements (see 3.3). For instance, a static corrector that accepts a line number. See Appendix 4.

4.2.6 Gamified Programming Exercise Modes

Table 4.2.6 Fulfillment of requirements of gamified programming exercise modes

Concept	Fulfillment
Shapeshifter	A Challenge in GEdIL may reference multiple exercises and have a "mode" (which can be SHAPESHIFTER) and a "mode parameters" (which can be the shift time) fields (see 4.1.2).
Shortening challenge	A Challenge in GEdIL may reference a single exercise and have a "mode" (which can be SHORTENING) and a "mode parameters" (which can be the number of lines/characters/etc.) fields (see 4.1.2).
Speedup challenge	A Challenge in GEdIL may reference a single exercise and have a "mode" (which can be SPEEDUP) and a "mode parameters" (which can be the execution time) fields (see 4.1.2).
Hack the problem	A Challenge in GEdIL may reference multiple exercises, have a "mode" field (which can be HACK_IT), and a Rule to trigger based on an evaluation metric (see 4.1.2). Additional metrics can be evaluated with a static corrector (e.g., to detect tricks) (see 3.3).
Time bomb	A Challenge in GEdIL may reference a single exercise and have a "mode" (which can be TIME_BOMB) and a "mode parameters" (which can be the bomb time) fields (see 4.1.2).

4.2.7 Badges Granted on Exercise-level

Table 4.2.7 Fulfillment of requirements for badges granted on exercise-level

Concept	Fulfillment
Hardworker	A Challenge in GEdIL may have a Rule to give a Reward of type BADGE

	and name “Hardworker”, if certain criteria are met (see 4.1.2, 4.1.3, and 4.1.4).
Scientist	A Challenge in GEdIL may have a Rule to give a Reward of type BADGE and name “Scientist”, if certain criteria are met (see 4.1.2, 4.1.3, and 4.1.4).
Keyword	A Challenge in GEdIL may have a Rule to give a Reward of type BADGE and name “Keyword”, if certain criteria are met (see 4.1.2, 4.1.3, and 4.1.4). Additional metrics can be evaluated with a static corrector (e.g., to detect certain keywords) (see 3.3).
Straight	A Challenge in GEdIL may have a Rule to give a Reward of type BADGE and name “Straight”, if certain criteria are met (see 4.1.2, 4.1.3, and 4.1.4). Additional metrics can be evaluated with a static corrector (e.g., to count loops) (see 3.3).

4.2.8 Types of Challenges Spanning Beyond a Single Programming Exercise

Table 4.2.8 Fulfillment of requirements of types of challenges spanning beyond a single programming exercise

Concept	Fulfillment
Duel	A Challenge in GEdIL may reference multiple exercises or challenges, define its rules and rewards, and have mode DUEL (see 4.1.2). A random challenge with mode DUEL may be picked when requested.
Quest	A Challenge in GEdIL may reference multiple exercises and define its rules and rewards (see 4.1.2). A Gamification Layer can have multiple Challenges (see 4.1.1).
Streak	Rules and Rewards in GEdIL may be attached to the Gamification Layer (see 4.1.1, 4.1.3, and 4.1.4).
Story	A Challenge in GEdIL may reference multiple exercises or challenges, define its rules and rewards to reveal content, and generate feedback with “feedback generators” (see 4.1.2).
Tournament	A Gamification Layer in GEdIL may represent a tournament as Leaderboards, Challenges, and Rewards can be attached (see 4.1.1). For the same reasons, a Challenge in GEdIL may represent a tournament.
Mystery Track	A Challenge in GEdIL may reference multiple exercises or challenges, define its rules and rewards to reveal content (see 4.1.2).

4.2.9 Badges Available on Course-level

Table 4.2.9 Fulfillment of requirements of badges available on course-level

Concept	Fulfillment
Solver	A Gamification Layer in GEdIL may have a Rule to give a Reward of type BADGE and name "Solver", if certain criteria are met - refer to the number of exercises solved without wrong submissions in a row (see 4.1.2, 4.1.3, and 4.1.4).
Man of Duty	A Gamification Layer in GEdIL may have a Rule to give a Reward of type BADGE and name "Man of Duty", if certain criteria are met - refer to the number of time units the player's streak lasts (see 4.1.2, 4.1.3, and 4.1.4).
Runner	A Gamification Layer in GEdIL may have a Rule to give a Reward of type BADGE and name "Runner", if certain criteria are met - check if it is the first (second, third) player who met it (see 4.1.2, 4.1.3, and 4.1.4).
Explorer	A Gamification Layer in GEdIL may have a Rule to give a Reward of type BADGE and name "Explorer", if certain criteria are met - refer to the number of hidden content elements uncovered in a course (see 4.1.2, 4.1.3, and 4.1.4).
Pathfinder	A Gamification Layer in GEdIL may have a Rule to give a Reward of type BADGE and name "Pathfinder", if certain criteria are met - refer to the number of exercises that the player solved the first (see 4.1.2, 4.1.3, and 4.1.4).

5. Appendices

5.1 Partial Implementation for Fill-in the Gap Exercise

This partial implementation aims to highlight the implementation details of a Fill-in the Gap exercise that differ from standard exercise types.

Consider an introductory exercise of Java,

Complete the following program so that it writes "Hello, World!".

The author of the exercise needs to select **FILL_IN_GAPS** in the Type field of the exercise, and provide a Skeleton with the gaps identified with `{{gap}}`. Example,

```
class HelloWorld
{
    public static void main(String args[])
    {
```

```
        System.out.{{gap}}("Hello, World");
    }
}
```

5.2 Partial Implementation for Mixed Code Exercise

This partial implementation aims to highlight the implementation details of a Mixed Code exercise that differ from standard exercise types.

Consider the following Python exercise,

> *Complete the following program to implement heap sort by sorting the provided blocks of code*

```
def heapify(nums, heap_size, root_index):
    # insert blocks here

def heap_sort(nums):
    n = len(nums)
    for i in range(n, -1, -1):
        heapify(nums, n, i)
    for i in range(n - 1, 0, -1):
        nums[i], nums[0] = nums[0], nums[i]
        heapify(nums, i, 0)
```

The author of the exercise needs to select **SORT_BLOCKS** in the Type field of the exercise, and provide multiple Skeletons corresponding to code blocks. Example,

SKELETON 1

```
largest = root_index
left_child = (2 * root_index) + 1
right_child = (2 * root_index) + 2
```

SKELETON 2

```
if left_child < heap_size and nums[left_child] > nums[largest]:
    largest = left_child
```

SKELETON 3

```
if right_child < heap_size and nums[right_child] > nums[largest]:
    largest = right_child
```

SKELETON 4

```
if largest != root_index:
    nums[root_index], nums[largest] = nums[largest], nums[root_index]
    heapify(nums, heap_size, largest)
```

Note: the code provided in the statement should be inserted in a Template with `{{code}}` in the place to insert blocks. However, this is common to other exercises using a template.

5.3 Partial Implementation for Show Me Exercise

This partial implementation aims to highlight the implementation details of a Show Me exercise that differ from standard exercise types.

Consider an introductory exercise of Java,

Print the points that form a straight line at an angle of 90° to the given line, with exactly the same length and center (the first point to print is the one with lower x or, if x is equal, lower y). The input/output should follow this format

start_x start_y end_x end_y

Example

***Input**: 2 4 6 4*

***Output**: 4 2 4 6*

The author of the exercise needs to provide a Feedback Generator that prints instructions to play the animation in VIMo (Vectors In Motion) [5], a format defined for this purpose. Example of output would be,

```
{
  "title": "Perpendicular Line",
  "width": 10,
  "height": 10,
  "fps": 1,
  "frames": [
    {
      "items": [
        {
          "type": "L",
          "content": {
            "x1": "2", "y1": "4", "x2": "6", "y2": "4",
            "attrs": { "stroke": "#000000" }
          }
        }
      ]
    },
```

```
    {
      "items": [
        {
          "type": "L",
          "content": {
            "x1": "2", "y1": "4", "x2": "6", "y2": "4",
            "attrs": { "stroke": "#000000" }
          }
        },
        {
          "type": "L",
          "content": {
            "x1": "4", "y1": "2", "x2": "4", "y2": "6",
            "attrs": { "stroke": "#ff0000" }
          }
        }
      ]
    }
  ]
}
```

5.4 Partial Implementation for Spot the Bug Exercise

This partial implementation aims to highlight the implementation details of a Spot the Bug exercise that differ from standard exercise types.

Consider an exercise of Python,

Identify the bugged lines in the given Quicksort implementation.

The author of the exercise needs to select **SPOT_THE_BUG** in the Type field of the exercise, and provide a Skeleton with the wrong code. Example,

```
def partition(nums, low, high):
    pivot = nums[(low + high) // 2]
    i = high + 1
    j = low - 1
    while True:
        i += 1
        while nums[i] < pivot:
            i += 1
        j -= 1
        while nums[j] > pivot:
```

```
            j -= 1
        if i >= j:
            return j
        nums[i], nums[j] = nums[j], nums[i]

def quick_sort(nums):
    def _quick_sort(items, low, high):
        if low < high:
            split_index = partition(items, low, high)
            _quick_sort(items, low, split_index)
            _quick_sort(items, split_index + 1, high)

    _quick_sort(nums, 0, len(nums) - 1)
```

The Solution would be a plain text file with the bugged line numbers. Example,

```
3
4
```

A Static Corrector could be a shell script as follows,

```
if ! diff -q $1 $2 &>/dev/null; then
  exit 2 # code for Wrong Answer
fi
```

executed with the following Command Line

```
./static_corrector.sh $attempt $solution
```

6. References

[1] Queirós, R., & Leal, J. P. (2011). Pexil: Programming exercises interoperability language. In *Conferência Nacional XATA: XML, aplicações e tecnologias associadas, 9. ª* (pp. 37-48). ESEIG.

[2] Swacha, J., Queirós, R., Paiva, J. C., & Leal, J. P. (2019). Defining Requirements for a Gamified Programming Exercises Format. In *23rd International Conference on Knowledge-Based and Intelligent Information & Engineering Systems*, Budapest, Hungary. (to appear).

[3] FGPE Consortium (2020). Yet Another Programming Exercise Interoperability Language. https://github.com/FGPE-Erasmus/format-specifications/blob/master/schemas/yapexil.schema.json Last updated at 2020-01-30.

[4] FGPE Consortium (2020). Gamified Education Interoperability Language. https://github.com/FGPE-Erasmus/format-specifications/blob/master/schemas/gedil.schema.json Last updated at 2020-01-30.

[5] FGPE Consortium (2020). Vectors In Motion. https://github.com/FGPE-Erasmus/format-specifications/blob/master/schemas/vimo.schema.json Last updated at 2020-01-30.

[6] FGPE Consortium (2019). IO1: Gamification Scheme for Programming Exercises. http://fgpe.usz.edu.pl/wp-content/uploads/FGPE_IO1_Gamification_Scheme_for_Programming_Exercises.pdf Last updated at 2019-05-25.