Parson's Problems Application for Learning Management Systems
Design Document

Authors:
Devin Allen
Stephen Cox
Benjamin Maymir
Zhenyu Yang

Group: 17

Document Revision History

Date	Version	Description	Author
11/9/2022	1.0	Initial draft.	Devin Allen Stephen Cox Benjamin Maymir Zhenyu Yang
11/27/22	1.1	Revised to reflect new specifications: ● Replaced GUI specs with terminal command specs in section 3. ● Changed context and class diagrams. ● Added LineGrabber class to section 4. ● Updated other class information to reflect the new class diagram. ● Added a constraint in section 2.4.	Stephen Cox

Table of Contents

1 Introduction

1.1 Purpose

This document describes the design, specifications, uses, and constraints of the Parson's Problems Appliance for Learning Management Systems (PPALMS) application. The document begins with an overview of the PPALMS system – describing its intended purpose, base functionality, and intended audience – followed by the system's design and architecture (Section 2), subsystems and constraints, interface and data stores (Section 3), structure and classes (Section 4), use diagrams (Section 5), fulfillment of non-functional requirements (Section 6), and traceability of requirements (Section 7).

1.2 System Overview

PPALMS is a stand-alone application that allows users to import source code from a file and generate quiz or exam questions from lines of programming code. Question types include multiple-choice, line reordering, 2D problems (indentation), fill-in-the-blank, and bug selection. The system saves these questions to a file of type (QTI) which can be imported into Learning Management Systems (LMS): Canvas, Moodle, or Blackboard. PPALMS is intended for use by instructors, institutions, and researchers using Learning Management Systems (LMS) at all education levels. PPALMS shall operate on current versions of Windows, Linux, and Mac computers, in an environment in which the user can create, modify, and execute files.

1.3 Design Objectives

PPALMS allows for easy, simple generation of computer-science-related exam questions by automating the action of modifying or rearranging lines of source code. This relieves users (instructors and organizations) of the manual process and reduces the time and resources necessary to create quizzes for students.

1.4 References

This document may refer to the Software Requirements Specification for PPALMS (version 1.1) as the 'Requirements Document' or 'Requirements'.

1.5 Definitions, Acronyms, and Abbreviations

For simplicity, the Parson's Problems Appliance for Learning Management Systems will be denoted as **PPALMS** in most cases. Learning Management System (those such as Canvas, Moodle, or Blackboard) will be denoted as **LMS**. Any mention of an instructor is assumed to be a computer science educator using the system to generate questions. The words 'questions' or 'problems' carry identical meaning and are used interchangeably. A 'mutation' denotes a version of a generated problem for selected lines of code. Other definitions can be found in Appendix A: Glossary.

2 Design Overview

2.1 Introduction

PPALMS shall be built using an object-oriented approach to allow communication and data processing between system components. The system shall feature a type of Generic Layered Architecture containing interconnected parts: Interface, Controller, Question Generator, and Questions. Question Generators shall be built using the strategy design pattern so the Controller can use proper question-generating algorithms based on the user's chosen settings.

2.2 Environment Overview

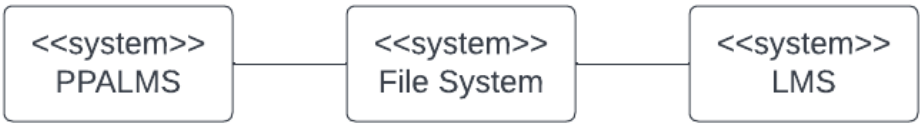

Figure 1: PPALMS Context Model

PPALMS is a stand-alone application that will be executable within a Linux environment. The system shall be able to output QTI files to the directory in which the system is saved. These files shall be importable to a LMS such as Canvas, Moodle, or Blackboard as determined by the user.

2.3 System Architecture

2.3.1 Top-level system structure of PPALMS

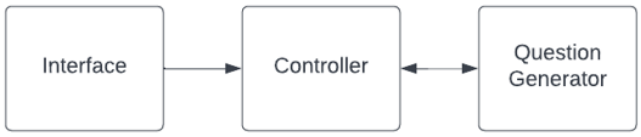

Figure 2: Top-level PPALMS Structure

The system consists of three major components: the terminal Interface, a system Controller, and a Question Generator. The user interacts with the Interface to send information to the Controller – the source code file path and the settings which affect question generation. The Controller also contains a type of Question Generator based on the chosen question type and saves all of the questions generated by the Generator.

2.3.2 Controller

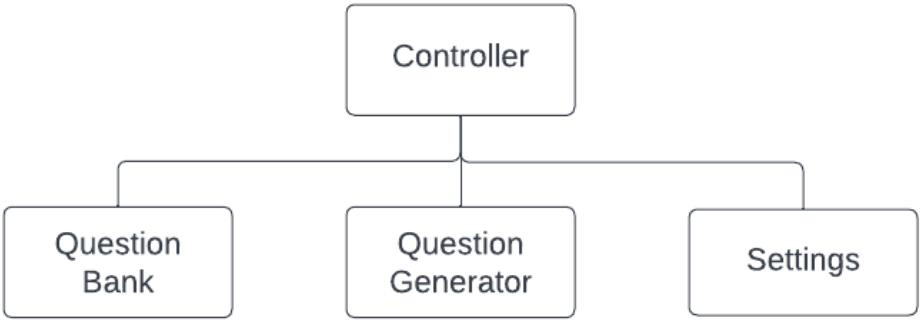

Figure 3: Controller Context Diagram

The Controller contains all system settings (mutation limit, intended LMS, question type, and source code file path) used in Question Generation. These settings are passed to the Generator whose type is decided by the selected question type by the user. Questions are generated and passed back to the Controller, then saved in a Question Bank for later export into a QTI file.

2.3.3 Generator

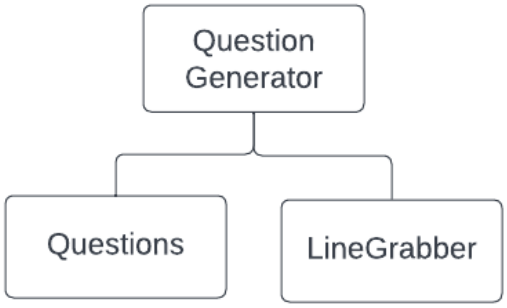

Figure 4: Generator Context Diagram

The Generator is responsible for generating questions based on the user's configured settings. The Generator uses a LineGrabber object which takes the settings and selected line numbers and returns the lines from a file back to the Generator. The Generator then uses different algorithms to create Parson's Problems (Questions) based on the given question type. These questions are returned back to the Controller.

2.4 Constraints and Assumptions

1. Question generation should not stagger system performance.

This is addressed by limiting the number of questions generated to a maximum of 25 with a default amount set to 5. The system creates questions based on selected lines of code and number of mutations.

2. PPALMS must allow users to select lines of code from an imported source file.

The system shall be developed using the object-oriented programming language, C++, which allows data reading and writing from files using file paths or file descriptors. The system shall allow users to enter line numbers (single lines or tuples) pertaining to a given file path to generate questions from.

3. Large source code files and number of lines may inhibit problem generation or output.

PPALMS shall require the computer to contain enough memory and storage to allow for problem generation and creation of a QTIi file (the common file type for use with an LMS) containing the sets of problems.

4. The user has a file they wish to use to generate questions.

The user provides the file and configures settings to generate questions.The file is assumed to be computer-programming code that compiles and runs as the user intends and is free of error.

3 Interfaces and Data Stores

3.1 System Interfaces

3.1.1 Terminal Interface

All input from the user is handled through the terminal. The system can be run using two similar commands:

./PPALMS <file_path> <lms> <question_type> <mutations> <line_nums>

./PPALMS

The first option runs the system once and generates questions based on the user's arguments as settings. The second option runs the system continuously and lists commands the user can input to generate sets of questions:

path <file_path>	Changes the current input file path e.g., path: ./project.cpp
lms <lms>	Changes the intended LMS target (Canvas, Moodle, or Blackboard)
type <question_type>	Changes question type to generate (2D, bugs, FITB, MC, or rearrange)
mutations <mutations>	Changes number of question mutation (Integer between 1 and 25)
lines <line_tuples> ... end	Changes line tuples to generate from e.g., lines: 1-2 6 9 45-48 end
generate	Generate questions with current settings
erase	Erase all generated questions
export	Exports saved questions to qti.txt
exit	Exits PPALMS

If the second option is selected, the system will run until the user enters the exit command.

3.2 Data Stores

The PPALMS system shall create a data store of the instructor's configured settings upon initial configuration interaction with the system through the terminalI. Additionally the multitude of questions produced from question generation shall be stored in a QuestionBank object which can then be exported to the instructor's hard disk drive as a QTI file.

4 Structural Design

4.1 Class Diagrams

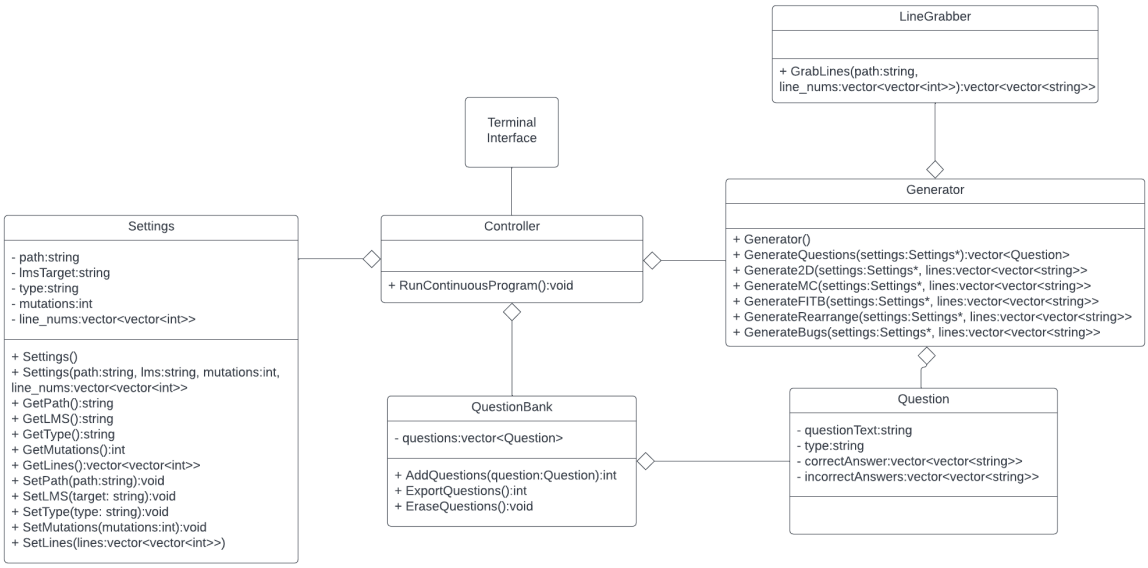

Figure 5: Class Diagram - PPALMS Overview

The user interacts with the system's Interface, which allows them to change the system's settings through the system's main Controller class. The Controller class contains Generator, QuestionBank, and Settings objects which it calls the methods on when the user enters the relevant command. The settings keeps all relevant information for question generation, such as the file path, question type, and line numbers. The Generator handles all question generation based on the configured settings and returns them to the Controller, which calls the AddQuestions() method on the QuestionBank to save them. The LineGrabber object takes the line numbers from the configured settings and returns the lines needed from the file back to the generator. Questions created include a question type, correct and incorrect answers, and a string of text to explain the question. The user can then call a command to export the generated questions from the QuestionBank.

4.2 Class Descriptions

4.2.1 Class: Controller

- Purpose: Control the flow of execution and link all subsystems together.
- Constraints: Terminal interface required to call methods of the Controller.
- Persistent: No (exists only when the system is running).

4.2.1.1 Attribute Descriptions

1. N/A

4.2.1.2 Method Descriptions

1. Method: RunContinuousProgram()
   Return Type: void
   Parameters: N/A
   Return value: N/A
   Pre-condition: Runs only when user starts the system with no initial settings or arguments.
   Post-condition: If all settings are entered and the system was prompted to generate questions, questions will be exported into a QTI file.
   Attributes read/used: User input and commands as described below.
   Methods called: N/A
   Processing logic: The program runs continuously until the user enters the exit command. Users can otherwise enter the following commands:

path <file_path>	Changes the current input file path e.g., path: ./project.cpp
lms <lms>	Changes the intended LMS target (Canvas, Moodle, or Blackboard)
type <question_type>	Changes question type to generate (2D, bugs, FITB, MC, or rearrange)
mutations <mutations>	Changes number of question mutation (Integer between 1 and 25)
lines <line_tuples> ... end	Changes line tuples to generate from e.g., lines: 1-2 6 9 45-48 end
generate	Generate questions with current settings
erase	Erase all generated questions
export	Exports saved questions to qti.txt
exit	Exits PPALMS

4.2.2 Class: Settings

- Purpose: Collects and holds settings related to question generation.
- Constraints: Limited to holding values specified by the user.
- Persistent: No (only exists while the system is running).

4.2.2.1 Attribute Descriptions

1. Attribute: mutations
   Type: int
   Description: The number of mutations that will be created per selected line
   Constraints: Must hold a value that is greater than or equal to 0.

2. Attribute: lms
   Type: string
   Description: The LMS target for which the QTI file will be generated
   Constraints: Must be one of “Canvas”, “Moodle” or “Blackboard”.

3. Attribute: type
   Type: string
   Description: The type of question that will be generated.
   Constraints: Must be one of “2D”, “Multiple Choice”, “Fill in the Blank”, “Bug Selection”, or “Reordering”.

4.2.2.2 Method Descriptions

1. Method: Settings()
   Return Type: none
   Parameters: none
   Return value: none
   Pre-condition: the RunContinuousProgram() function has been called in the controller.
   Post-condition: a base Settings object was created for configuration.
   Attributes read/used: none
   Methods called: N/A
   Processing logic: A base Settings object gets created for user configuration, which is then used for generating questions.

2. Method: Settings(path, lms, type, mutations, line_nums)
   Return Type: none
   Parameters: Intended settings: a file path, LMS target, question type, number of mutations, and selected line numbers.
   Return value: none
   Pre-condition: The user has run the system with settings arguments.
   Post-condition: a Settings object was created for question generation.
   Attributes read/used: path, lms, type, mutations, line_nums
   Methods called: N/A
   Processing logic: Given the arguments, a Settings object is created for generation.

3. Method: GetPath()
   Return Type: string
   Parameters: none
   Return value: the set file path for question generation.
   Pre-condition: there exists a file path.
   Post-condition: the file path remains unchanged
   Attributes read/used: path
   Methods called: N/A

Processing logic: The value of the file path is returned.

4. Method: GetLMS()
   Return Type: string
   Parameters: none
   Return value: the value of LMSTarget
   Pre-condition: LMSTarget holds a value
   Post-condition: LMSTarget remains unchanged
   Attributes read/used: LMSTarget
   Methods called: N/A
   Processing logic: The value of LMSTarget is returned.

5. Method: GetType()
   Return Type: string
   Parameters: none
   Return value: the value of questionType
   Pre-condition: questionType holds a value
   Post-condition: questionType remains unchanged
   Attributes read/used: questionType
   Methods called: N/A
   Processing logic: The value of questionType is returned.

6. Method: GetMutations()
   Return Type: int
   Parameters: none
   Return value: the value of mutations
   Pre-condition: mutations holds a value
   Post-condition: mutations remains unchanged
   Attributes read/used: mutations
   Methods called: N/A
   Processing logic: The value of mutations is returned.

7. Method: GetLines()
   Return Type: vector<vector<int>>
   Parameters: none
   Return value: a vector containing tuples of line numbers.
   Pre-condition: the line numbers have been set on the Settings object.
   Post-condition: line numbers remain unchanged in the object.
   Attributes read/used: line_nums
   Methods called: N/A
   Processing logic: A vector of line tuples (vectors) is returned.

8. Method: SetPath(path)
   Return Type: void
   Parameters: the desired file path
   Return value: N/A
   Pre-condition: N/A
   Post-condition: the Settings object now holds the intended file path.
   Attributes read/used: path
   Methods called: none
   Processing logic: The Settings object saves the given file path into its ‘path’ variable.

9. Method: SetLMS(lms)
   Return Type: void
   Parameters: the desired LMS target
   Return value: N/A
   Pre-condition: N/A
   Post-condition: the Settings object holds the intended LMS target value.
   Attributes read/used: lms
   Methods called: none
   Processing logic: The Settings object's lms is set to the value of the lms parameter.

10. Method: SetType(type)
    Return Type: void
    Parameters: the desired question type
    Return value: N/A
    Pre-condition: N/A
    Post-condition: the value of question type is equal to the questionType parameter
    Attributes read/used: questionTYpe
    Methods called: none
    Processing logic: questionType is set to the value of the questionType parameter.

11. Method: SetMutations(mutations)
    Return Type: void
    Parameters: the desired number of mutations
    Return value: N/A
    Pre-condition: N/A
    Post-condition: the value of mutations is equal to the the mutations parameter
    Attributes read/used: mutations
    Methods called: none
    Processing logic: mutations is set to the value of the mutations parameter.

12. Method: SetLines(lines)
    Return Type: void
    Parameters: vector<vector<int>>, a vector containing line tuples (as vectors).
    Return value: N/A
    Pre-condition: N/A
    Post-condition: the Settings object holds the line-number tuples.
    Attributes read/used: lines, line_nums
    Methods called: none
    Processing logic: The Settings object's line_nums variable is set to hold the
    given parameter (lines).

4.2.3 Class: Generator

- Purpose: Generates questions after the user has selected lines of code and settings.
- Constraints: question types are limited to 2D, multiple choice, fill-in-the-blank, rearranging lines, and bug selection.
- Persistent: No (only exists while the system is running).

4.2.3.1 Attribute Descriptions

1. N/A

4.2.3.2 Method Descriptions

1. Method: Generator()
   Return Type: none
   Parameters: none
   Return value: none
   Pre-condition: none
   Post-condition: a Generator object is created for use in question generation.
   Attributes read/used: none
   Methods called: none
   Processing logic: Creates a Generator object for use in question generation.

2. Method: GenerateQuesitons(settings)
   Return Type: vector<Question>
   Parameters: settings (a Settings pointer)
   Return value: a vector containing generated Question objects.
   Pre-condition: settings have been configured.
   Post-condition: questions generated based on the configured settings.
   Attributes read/used: settings
   Methods called: One of the following based on settings:
   Generate2D(settings, lines)
   GenerateMC(settings, lines)
   GenerateFITB(settings, lines)
   GenerateRearrange(settings, lines)
   GenerateBugs(settings, lines)

   Also calls the GrabLines(path, lines) function on the LineGrabber object. GetPath(), GetType(), GetLines() on the Settings object.

   Processing logic: The method calls the GrabLines() function on the LineGrabber object, then passes those and the settings to a relevant question generation function. The generated questions are passed back to the controller.

3. Method: Generate2D(settings, lines)
   Return Type: vector<Question>
   Parameters: settings (a Settings pointer), lines (lines grabbed from the given file path)
   Return value: a generated set of 2D questions.
   Pre-condition: settings have been configured and the user has selected lines of code.
   Post-condition: 2D questions generated based on the given settings and lines.
   Attributes read/used: settings, lines
   Methods called: N/A
   Processing logic: The method generates 2D questions based on the desired settings and grabbed lines.

4. Method: GenerateMC(settings, lines)
   Return Type: vector<Question>
   Parameters: settings (a Settings pointer), lines (lines grabbed from the given file path)
   Return value: a generated set of multiple choice questions.
   Pre-condition: settings have been configured and the user has selected lines of code.
   Post-condition: multiple choice questions generated based on the given settings and lines.
   Attributes read/used: settings, lines
   Methods called: N/A
   Processing logic: The method generates multiple choice questions based on the desired settings and grabbed lines.

5. Method: GenerateFITB(settings, lines)
   Return Type: vector<Question>
   Parameters: settings (a Settings pointer), lines (lines grabbed from the given file path)
   Return value: a generated set of fill-in-the-blank questions.
   Pre-condition: settings have been configured and the user has selected lines of code.
   Post-condition: fill-in-the-blank questions generated based on the given settings and lines.
   Attributes read/used: settings, lines
   Methods called: N/A
   Processing logic: The method generates fill-in-the-blank questions based on the desired settings and grabbed lines.

6. Method: GenerateRearrange(settings, lines)
   Return Type: vector<Question>
   Parameters: settings (a Settings pointer), lines (lines grabbed from the given file path)
   Return value: a generated set of line rearranging questions.
   Pre-condition: settings have been configured and the user has selected lines of code.
   Post-condition: line rearranging questions generated based on the given settings and lines.
   Attributes read/used: settings, lines
   Methods called: N/A
   Processing logic: The method generates line rearranging questions based on the desired settings and grabbed lines.

7. Method: GenerateBugs(settings, lines)
   Return Type: vector<Question>
   Parameters: settings (a Settings pointer), lines (lines grabbed from the given file path)
   Return value: a generated set of bug-selection questions.
   Pre-condition: settings have been configured and the user has selected lines of code.
   Post-condition: bug-selection questions generated based on the given settings and lines.
   Attributes read/used: settings, lines
   Methods called: N/A

Processing logic: The method generates bug-selection questions based on the desired settings and grabbed lines.

4.2.4 Class: LineGrabber

- Purpose: To read the file from the given file path and retrieve the selected lines for question generation.
- Constraints: A file path must be set.
- Persistent: No (object is initialized at runtime and deleted upon system termination).

4.2.4.1 Attribute Descriptions

1. N/A

4.2.4.2 Method Descriptions

1. Method: GrabLines(path, line_nums)
   Return Type: vector<vector<string>>
   Parameters: path (file path to read lines from), line_nums (tuples of line numbers)
   Return value: a set of lines from the file path corresponding to the given line tuples.
   Pre-condition: the file path and line numbers have been set in the Settings object.
   Post-condition: lines returned to the Generator object.
   Attributes read/used: path, line_nums from the Settings object.
   Methods called: GetPath(), GetLines() from the Settings object.
   Processing logic: Opens the file at the given file path and reads in all its lines. Grabs and returns all lines at the given line numbers to the Generator object.

4.2.5 Class: QuestionBank

- Purpose: To store the generated questions
- Constraints: None
- Persistent: No (object is initialized at runtime and deleted upon system termination)

4.2.5.1 Attribute Descriptions

1. Attribute: questions
   Type: vector<Question>
   Description: A vector of the questions generated (questions stored as question objects)
   Constraints: None

4.2.5.2 Method Descriptions

1. Method: AddQuestions(Question)
   Return Type: int
   Parameters: None
   Return value: on success returns 1, on failure returns 0.
   Pre-condition: None
   Post-condition: Expanded questions vector
   Attributes read/used: questions
   Methods called: None
   Processing logic: The method takes in a Question object as its sole parameter and attempts to append it to the questions attribute (vector<Question>). It returns 1 upon successfully appending the question, otherwise 0 is returned.

2. Method: ExportQuestions()
   Return Type: int
   Parameters: None
   Return value: on success returns 1, on failure returns 0.
   Pre-condition: None
   Post-condition: Question objects written to QTI file
   Attributes read/used: questions.
   Methods called: None
   Processing logic: The method loops through the questions vector and attempts to write each Question object to a given .qit file. If an error occurs during any of the write operations return 0, else return 1.

3. Method: EraseQuestions()
   Return Type: void
   Parameters: none
   Return value: none
   Pre-condition: none
   Post-condition: any saved questions are erased.
   Attributes read/used: questions.
   Methods called: None
   Processing logic: The method erases all the Question objects in the questions vector.

4.2.6 Class: Question

- Purpose: Contains a question to be saved to a QuestionBank.
- Constraints: Only generated by QuestionGenerator objects based on Settings.
- Persistent: No (only exists upon generation by a QuesitonGenerator object)

4.2.6.1 Attribute Descriptions

1. Attribute: questionText
   Type: string
   Description: text pertaining to the generated question.
   Constraints: determined by the type of question from Settings.

2. Attribute: type
   Type: string
   Description: the type of question generated (determined by the user).
   Constraints: determined by the type of question from Settings.

3. Attribute: correctAnswer
   Type: vector<string>
   Description: contains any correct answers for the generated question.
   Constraints: contains only one element if it is a Multiple Choice, Fill-in-the-Blank, or Bug Selection question type.

4. Attribute: incorrectAnswers
   Type: vector<string>
   Description: contains any incorrect answers for the question if they need to be recorded.
   Constraints: only contains incorrect answers for Multiple Choice questions.

5 Dynamic Model

5.1 Scenarios

5.1.1 Importing Source Code

Instructors and other users shall select the source code file they wish to import by using the system's interface. The system's controller receives the source code file and saves its path until it is ready to generate questions. The interface shall display a message to the user on success or an error if their selected file is invalid or does not exist.

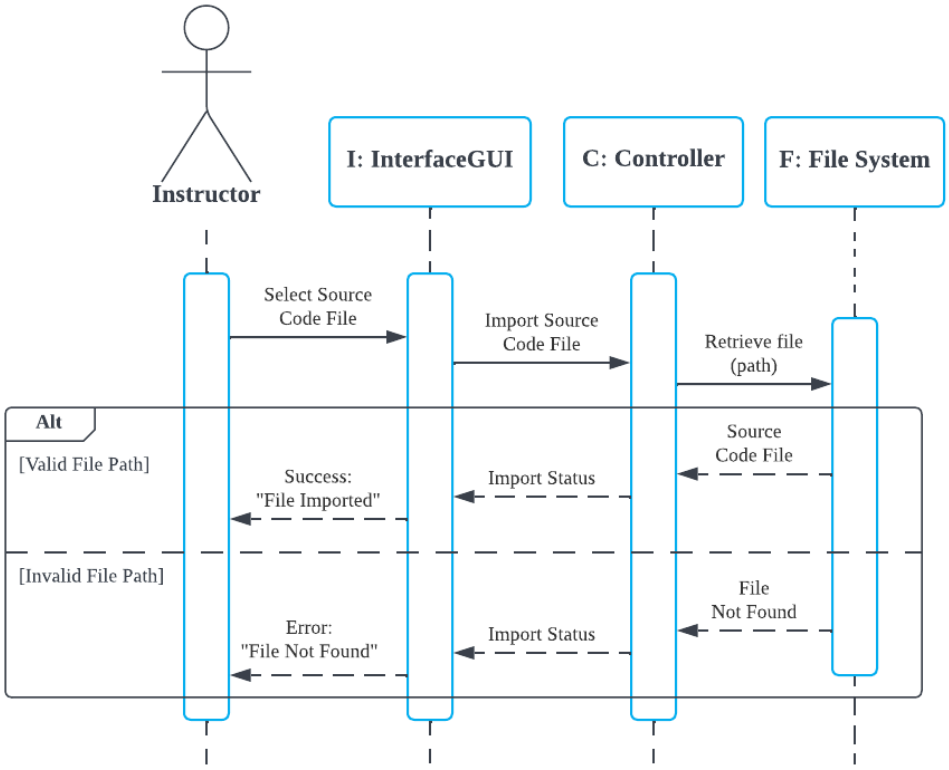

Figure 6: Sequence diagram detailing use case 1

5.1.2 Changing System Settings

Instructors and other users shall be able to change the system settings used in question generation: question mutation amount, question type, and intended LMS. The system's controller receives and saves these settings in a Settings object (Config). The interface shall display an error message to the user if the desired settings cannot be set.

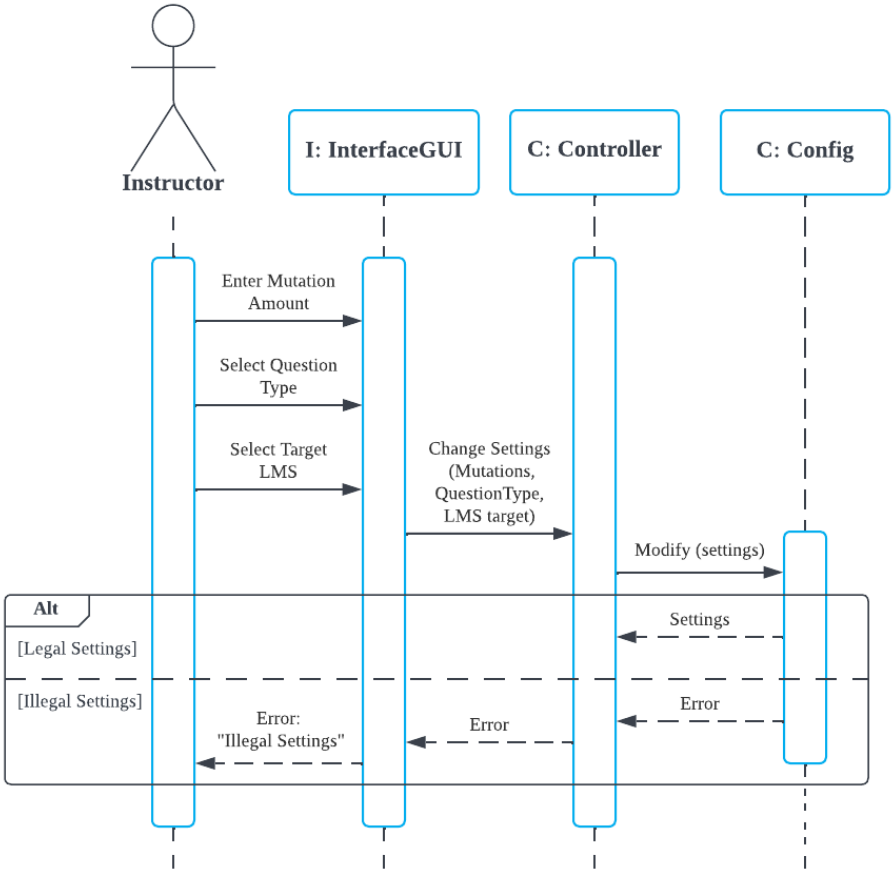

Figure 7: Sequence diagram detailing use case 2

5.1.3 Selecting Lines of Code

All users shall be able to select the lines from which questions will be generated. The system's controller shall inform the user of the current selected lines via the interface. If the user selects more than 200 lines, the system shall display a warning informing the user that they may experience a delay in completion due to the size of their selection. If the user selected no lines, the system shall display an error informing the user that they must make a selection.

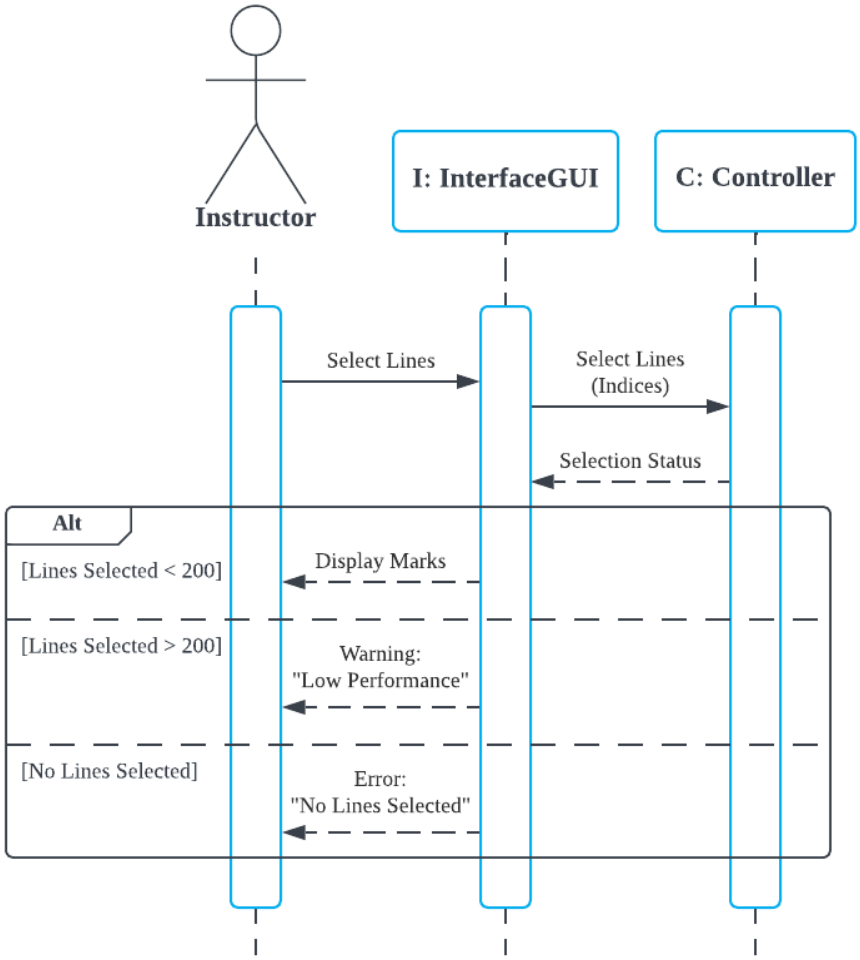

Figure 8: Sequence diagram detailing use case 3

5.1.4 Generate Questions

The system shall generate questions from the settings and lines selected by the user. The user shall request questions be generated by clicking on the Generate button. The controller shall check whether the question type is compatible with the lines selected, and display an error if it is not. Then the controller shall provide the question generator with the user's settings and selected lines, with which the generator shall create a bank of questions. Then the generator shall then verify that each question has a solution. The question generator shall finally return the questions generated to the controller. If no errors were encountered, the system shall display a message indicating that the questions were successfully created, and send the questions to the question bank. Otherwise the system shall display a message indicating that an error occurred.

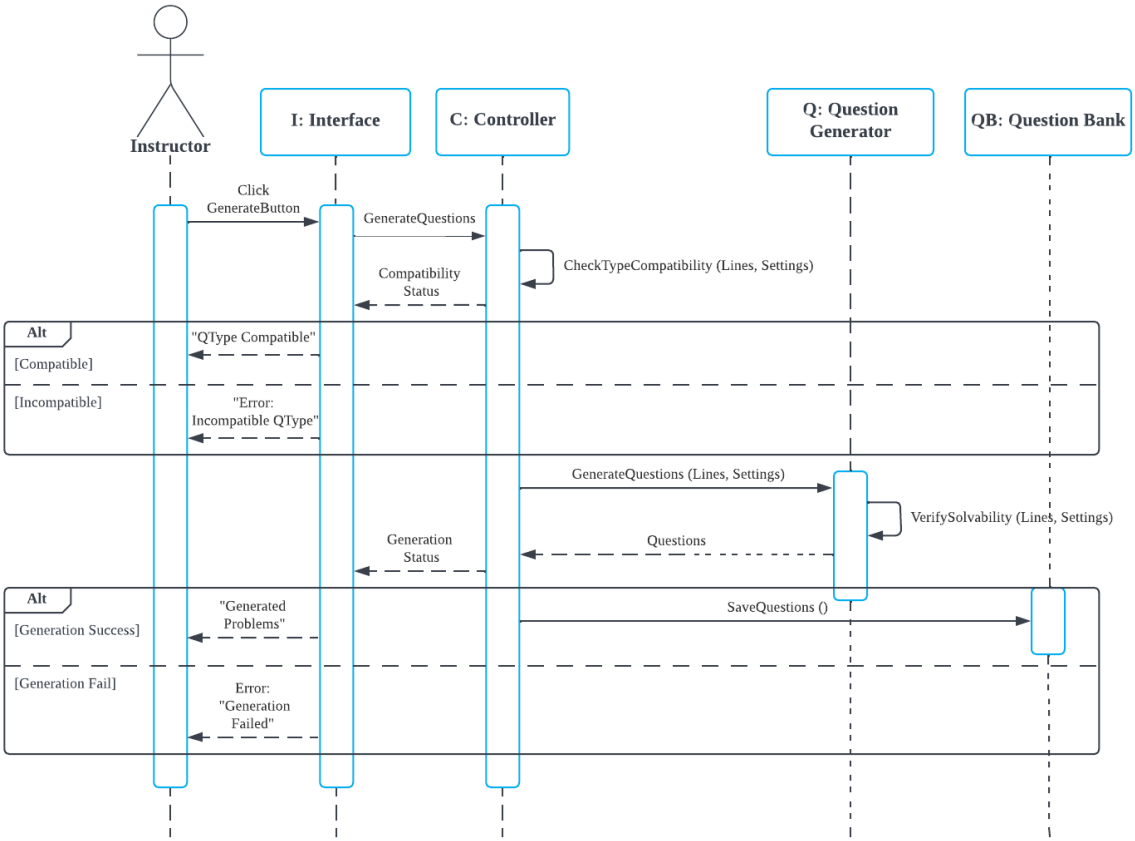

Figure 9: Sequence diagram detailing use case 4

5.1.5 Export Questions to File

The system shall export a question bank as a QTI file.. The user shall request questions be exported by clicking the Export button.. The controller shall call the question bank to create a QTI file from all questions previously generated.The question bank shall then save the file to the file system, and return a status to the controller. If the file was successfully created and saved, the system shall inform the user that the export was successful. Otherwise it will inform the user that an error occurred.

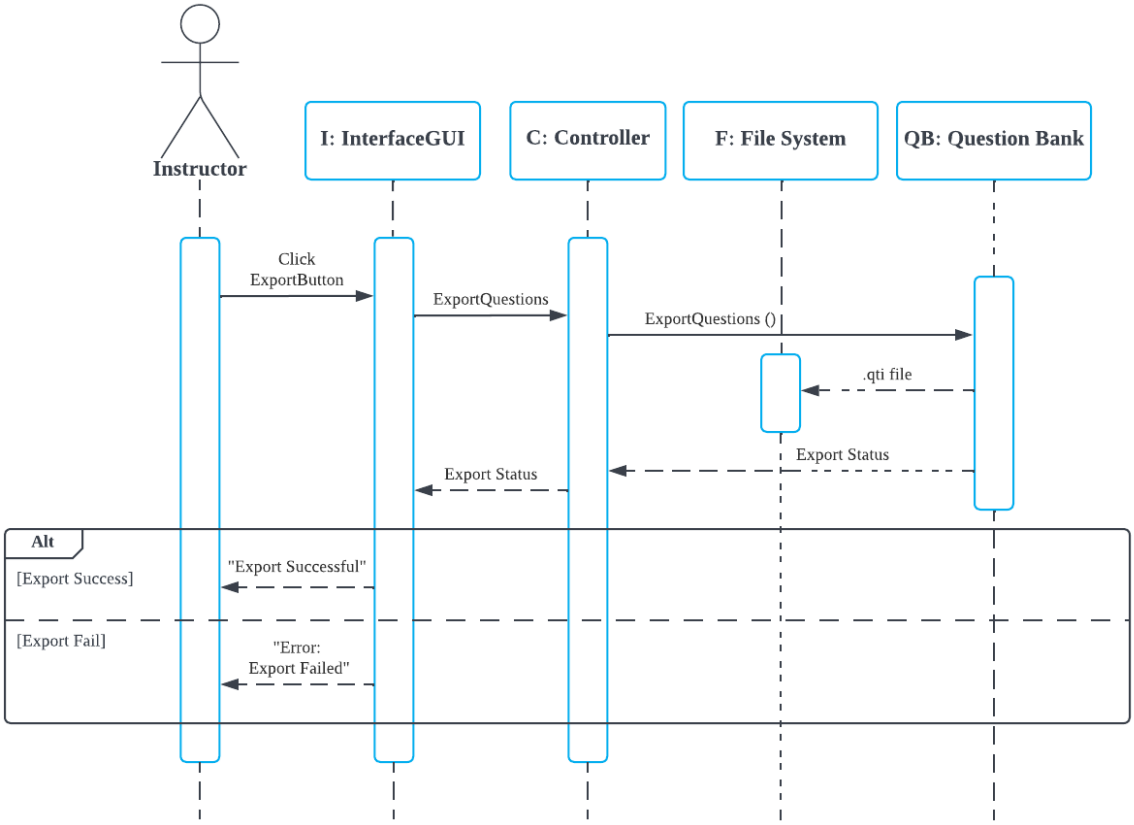

Figure 10: Sequence diagram detailing use case 5

6 Non-functional Requirements

6.1 Performance Requirements

PPALMS allows source code (as a text file) that contains correct, compilable programming language to be imported and used to generate problems. The number of mutations shall be set to 5 by default and be configurable by the user within a range from 1 to 25. The system shall strictly limit this number to maintain higher performance. If the user selects a large number of lines for problem generation, a warning shall be displayed indicating that system performance will be impacted and that the system will take more time to complete question generation.

6.2 Safety Requirements

The source code shall not be affected by the system; the system shall not add-to, delete, or modify imported source code in any way. The system shall not have capability to execute, correct, or compile the source code. Users shall only be able to generate problems from imported source code to be exported to an LMS – the system contains no other functionality. The source code will not be verified for compiling or runtime errors and will be assumed to be executable and functional to its full extent. Users should compile their code and make sure it is free from errors before importing to PPALMS.

6.3 Security Requirements

PPALMS will always be packaged along with a list of hashes for each file in the product. The user has the ability to verify the authenticity of the program by hashing the files they received and comparing the hashes to the hash list. The system should always be concerned with potential vulnerabilities like stack overflow. The system shall contain ASLR (Address Space Layout Randomization) to avoid vulnerabilities.

6.4 Software Quality Attributes

The PPALMS system shall be expandable to support source code containing other programming languages which are not currently supported. The system shall implement an I/O stream to import the source code and let the user select lines to generate corresponding problems. Generated problems are tested against the source code for equivalency – to prevent problems from having no correct or incorrect answers.

6.5 Business Rules

The permissions of the source code should be set to read-only, the system will not modify or execute the source code. The source code will not be verified for compiling or runtime errors and will be assumed to be executable and functional to its full extent. Users should compile their code and make sure it is free from errors before importing to PPALMS.

7 Requirements Traceability Matrix

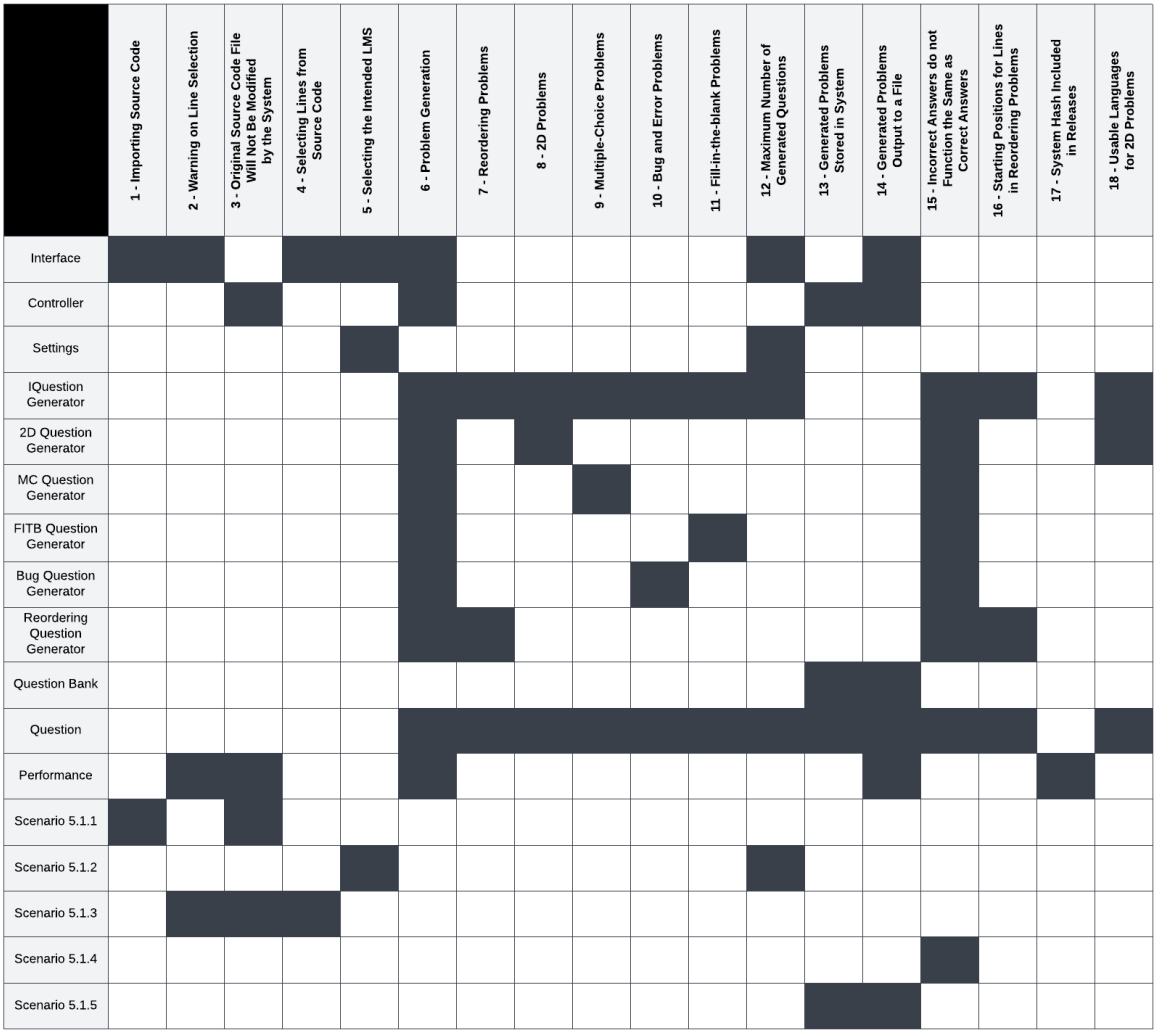

Figure 14: Traceability Matrix Diagram

Appendix: Glossary

LMS: Abbreviation for Learning Management System; a software system that assists educators, for use by both instructors, students, and institutions.

Mutation: Code that has been modified by the system for generated problems.

Parson's Problems: Computer programming problems based on modifying, reordering, or selecting lines of code.

PPALMS: Abbreviation for the name of the system (Parson's Problems Appliance for Learning Management Systems).

Regex: (1) Regular expression. (2) The regular expression libraries used to check expressions for equivalency and correctness.

QTI: (Question and Test Interoperability) A file type containing generated problems, able to be imported into an LMS.

Variation: (see Mutation).