

Using Parson's Problems in a Mobile App with Student Modelling

By

Aaryan Jay Ragoo

Project unit: PJE40

Supervisor: Dr Matthew Poole

April 2020

Abstract

This project presents the development of a mobile system using Parson's problems code rearrangement exercises with student modelling to contribute to solving the ever-present problem of educating beginner programmers. Through employing a software engineering methodology, relevant literature is studied, system requirements are produced, designs are formed and the system implementation is coded and tested. Student modelling, the process by which the educational system adapts to the user's ability, is chosen as a means for furthering the application of Parson's problems to a mobile system. This is because of the identified gap in research and extant software that effectively combines the three concepts of informal mobile learning, Parson's problems, and student modelling.

Word count from Introduction to Conclusion (inclusive): 15,502.

Total document word count: 20,780.

Acknowledgements

The author would like to express their complete gratitude to the project supervisor, Dr Matthew Poole, for their constant guidance throughout the duration of the project and to the project moderator, Nadim Bakhshov, for agreeing to attempt to arrange an observational study.

Contents

1. Introduction	4
1.1 Background	4
1.2 Project Aims	6
1.3 Document Outline	7
2. Literature Review	8
2.1 Parson's Problems	8
2.2 Why Parson's Problems Are Used	13
2.3 The Role of Student Modelling	15
3. Methodology & Management	17
3.1 Project Management for Risk Mitigation	17
3.2 Software Process Model	18
4. Requirements Engineering	20
4.1 Requirements Elicitation	20
4.2 Requirements Specification	21
5. Design	25
5.1 System Architecture	25
5.2 User Interface	27
5.3 Student Modelling Algorithm Design	33
6. Implementation	35
6.1 Issues and Solutions	35
6.2 Approach to Testing	45
7. Evaluation against Requirements	47
8. Conclusion	53
9. References	55
10. Appendices	57
A - Project Initiation Document	57
B - Ethics Certificate	65
C - Observational Study Information Sheet	66
D - Brief Extant Software Survey	69
E - Testing Table	77

1. Introduction

This project is concerned with the development of a mobile app that uses a type of code rearrangement exercise, named Parson's problems, as a tool to aid a beginner programmer's learning. This chapter will cover the context in which the project takes place in, to introduce the reader to the paradigm of informal learning through the use of Parson's problems. Section 1.1 contextualizes the project by looking at the background to the problem of learning computer programming. Section 1.2 discusses the objectives of the project from a high level of scope and Section 1.3 lays out the structure of the content of this document and briefly discusses the intention of each following chapter.

1.1 Background

As with learning any new skill, learning to use a programming language to solve computational problems can be difficult for a beginner programmer due to the large amount of semantic, syntactic and lexical knowledge required to develop complex computer programs. One solution used to tackle this difficulty is to distribute the learning process across informal mobile platforms which aim to be as engaging as they are pedagogically stimulating; under the accepted assumption that a student is more willing to spend time learning a subject if the process is both immediately gratifying and accessible. Pre-existing informal mobile programming education tools such as 'Mimo' (Fig. 1A) and 'Codemurai' (Fig. 1B) make use of various question types, including Parsons's Problems, in order to offer a kind of informal Education-as-a-Service system that a beginner programmer may use to develop or further their understanding of a programming language.

Parson's problems are a code rearrangement exercise where the participant orders a given set of lines of code to provide a solution to a given problem or prompt. An example of a problem or prompt could be to "Arrange a series of unordered lines of code to output the message 'Hello, World!'." Distractor lines, which are lines of code not required to complete the solution to the problem, can be included in the given set of lines from which the user selects, but should not be used to correctly answer the problem. The problem type exists out of the necessity to create a programming exercise that requires more creative involvement than reading sections of code and identifying the code's output, but that is also not as mentally arduous and time consuming as writing new code to solve problems from scratch.

This necessity is substantiated by the wealth of academic literature published on the topic of using Parson's problems within an educational context (reviewed in Chapter 2), that provide research on the nuances of implementing a Parson's problems as well as how they can outperform more traditional exercise types such as code reading and writing in their efficacy for learning.



Fig. 1A. Mimo implementation of Parson's problems.

Can you drag the following blocks of code and place them in the **correct order** to do the following.

1. Declare a variable called canJump.
2. Give it a value of true.
3. Give it a value of false.



Fig. 1B Codemuray implementation of Parsons Problems.

There does not yet exist a tool which exclusively makes use of Parson's problems as a means for offering formative challenges to a beginner programmer. Instead, many mobile programming education tools, such as SoloLearn (Figs. 1C, 1D, and 1E), attempt to provide a complete programming course including information pages with various types of questions, such as filling in blank spaces in segments of code, multiple choice questions and guessing the output of sections of code.

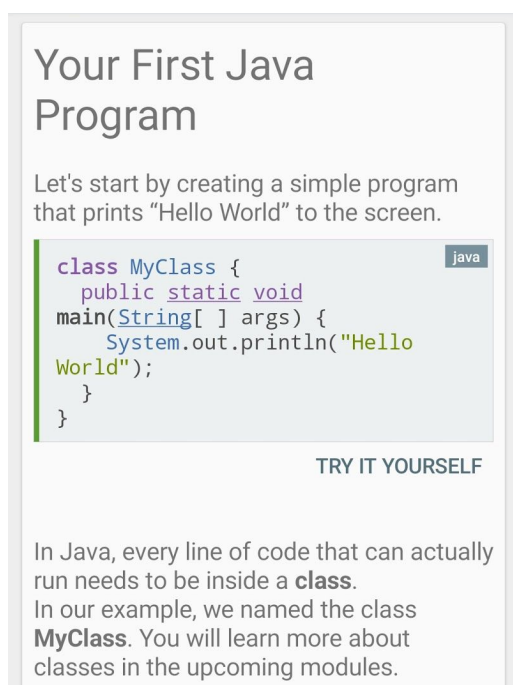


Fig. 1C. SoloLearn instructional page.

What is the result of the following code?

```
int x = 15; int y = 4;
int result = x / y;
System.out.println(result);
```

Fig. 1D. SoloLearn code output question.

Fill in the blanks to create a valid Java program.

```
____ Apples {
    public static void ____ (String[] args)
    {
        System.out.println("Hello, World!");
    }
}
```

Fig. 1E SoloLearn 'fill in the blank' question.

Historically, a beginner programmer could utilise a textbook in order to learn an unfamiliar programming language. In the development of a new electronic tool that caters to beginner programmers, it may be useful to identify the progression of topics a programming textbook author chooses to teach a learner. In looking at the contents pages from three different textbooks for the programming languages C (Kernighan & Ritchie, 1988), Python (Matthes, 2016), and Java (Schildt, 2019), the consistently occurring beginner topics include: using variables, using control structures, data structures, function definition, input/output and object oriented programming principles. It can be seen how a system aiming to supplement teaching introductory programming principles with Java as an example language should use these topics to guide the user's learning.

Student modelling is a technique whereby a data representation of a student is developed as they use an educational tool. The purpose of such a technique is to develop an understanding of the student's capabilities in order to present educational media that is as close as possible to matching, and is able to adapt to the student's current understanding. By doing so, exercises can be presented that are better able to stimulate the user's learning in such a way that is neither excessively trivial or challenging.

This project is motivated by the recognition that while mobile tools exist to teach beginner programmers, none fully exploit the potential, demonstrated by academic literature, that Parson's problems hold, in tandem with student modelling techniques. By developing a new system that builds on both academic consensus on the use of Parson's problems, as well as the usability successes of existing mobile educational tools, knowledge on the role of informal learning can be furthered.

1.2 Project Aims

For the successful completion of this project, specific aims must be established which describe the events that should take place across the development of the project. Firstly, a review of academic literature concerning the use of Parson's problems as an educational tool should be conducted in order to develop a knowledge of what researchers have come to understand about ways in which Parson's problems can be most effectively used in a pedagogical context. With the literature review completed system requirements can be elicited and specified from knowledge gained in reading academic papers. Then a GUI wireframe to illustrate the interaction flow of the system should be designed that is informed by the requirements engineering process. Next, a code artifact that uses the GUI designs and satisfies the specified requirements should be developed. To further the development of the code artifact, several Parson's problems, which the system can make use of, should be defined. Then a

testing and evaluative process should be applied to the implemented system before any conclusions are drawn. The development process of the system is then documented in this written report.

1.3 Document Outline

This section provides a brief description of each chapter.

In Chapter 2, a review of academic literature is conducted on the topic of the use of Parson's problems as a pedagogical tool. The Chapter evaluates this literature in order to develop an appropriate list of requirements that a new tool using Parson's problems must, should, and could satisfy in order to be effective (described in Chapter 4).

In Chapter 3 the processes by which the development of the project is managed is discussed, by justifying the use of one software process model over others as well as evaluating the techniques used to mitigate risks.

Chapter 4 goes through the process by which the project's requirements are elicited, prioritized and then formally defined.

Chapter 5 details the design process in order to develop a GUI model and to develop graphical representations of the developed system's architecture.

Chapter 6 covers the process by which implementation issues are considered and then solved, as well as the employed approach to testing.

Chapter 7 looks at how effectively the developed system satisfies requirements as specified in Chapter 4, and evaluates and justifies any deviations made by the system away from the defined requirements.

Chapter 8 concludes the project by considering the contributions to the domain as well as suggesting any further work that could be completed to further the project.

Chapter 9 includes any references made throughout this document and Chapter 10 includes the Project's appendices which includes supporting material such as the Project Initiation Document, Ethics Certificate, Observational Study Information Sheet, Extant Software Survey, and Testing Table.

2. Literature Review

The aim of this chapter is to review the literature concerning the use of Parson's problems as a means for teaching students the skills and syntax used in writing computer programs. Since their inception by Parsons & Haden, 2006 (Fig. 2A), Parson's problems have inspired a wealth of academic literature within the field of computer science education. In reviewing this literature, we can recognise certain emergent themes and use these themes as a way to guide discussion on the minutiae of Parson's problems which can then elicit requirements for a new system to be developed. This chapter breaks up its review of literature into these themes by looking at: in Section 2.1 design considerations, in Section 2.2 pedagogical suitability, and in Section 2.3 the role of student modelling, each as pertaining to Parson's problems.

Lab 1
Syntax practise

Drag the statements on the right into the correct order. Do not use the incorrect statements.
Should produce four lines of text- read them to get them in the correct order.

1	WriteLn("This is the first line");
2	WriteLn("Don't use a line that's not correct as line 2");
3	WriteLn(5.67.2);
4	Begin InitOurCrt;
5	WriteLn("Don't use a line that's not correct as line 2");
6	WriteLn(5.67234.5.2);
7	Program Pick_the_correct_lines; (\$APPTYPE CONSOLE)
8	End;
	End;
	uses SysUtils, OurCrt;
	WriteLn("The line below should show the number 5.67");

Fig. 2A. An example unsolved Parson's problem as implemented by Parsons & Haden (2006)

2.1 Parson's Problems

Certain factors concerning the varying designs of Parson's problems are presented upon reviewing relevant literature. Denny et al. (2008) set out a method for creating variations on Parson's problems by modifying certain properties that the problem could exhibit. These properties include whether the

given lines are a superset of the required lines (i.e. whether distractor lines are used), whether the answer space is scaffolded (to indicate the structure of the solution) or plain, and whether or not distractor lines are paired with their correct counterpart (Fig. 2B) or randomly ordered with correct lines in the ‘bank’ of given lines. In their exam scenario, the most difficult option (jumbled line options while using distractors) were not used as they were regarded as too confusing and tedious, but it can be seen how that in spite of this, these properties can be used to control the difficulty of Parson's problems to allow for the incremental progression of a large set of Parson's problems.

```

return result;
return word;

String result = "";
String result;

if (word.charAt(i) == 'a')
if (word.charAt(i) != 'a')

for (int i = 0; i < word.length(); i++)
for (int i = 0; i < word.length; i++)

result = result + word.charAt(i);
result = word.charAt(i);

private String removeAllAs(String word)
private String removeAllAs(word)

```

Fig. 2B. An example Parson's problem exam question used by
Dennis et al (2008)

Ihantola and Karavirta (2011) introduce another variation on Parson's problems (Fig. 2C) by adding another dimension of movement for the lines of code. As well as arranging lines vertically, the lines can be moved horizontally in order to represent the necessary indentation for forming a correctly functioning Python program. While such a concept would be non-essential for a language where indentation makes no difference to the function of the program (such as Java), it provides an interesting design consideration that could be exploited to reinforce ideas around good general indentation practices. Within the same paper, Ihantola and Karavirta offer another design consideration in observing how experienced programmers, when solving the Parson's problems, would solve the problem non-linearly and would rarely use the option to request automated feedback on their solution before submitting their response. This could suggest how that it would be more useful for the development of the skills of an experienced programmer for a system to provide feedback to the user after they have submitted their answer, rather than preempting the user. However they do not provide any reason whether or not this may hold true for less experienced programmers.

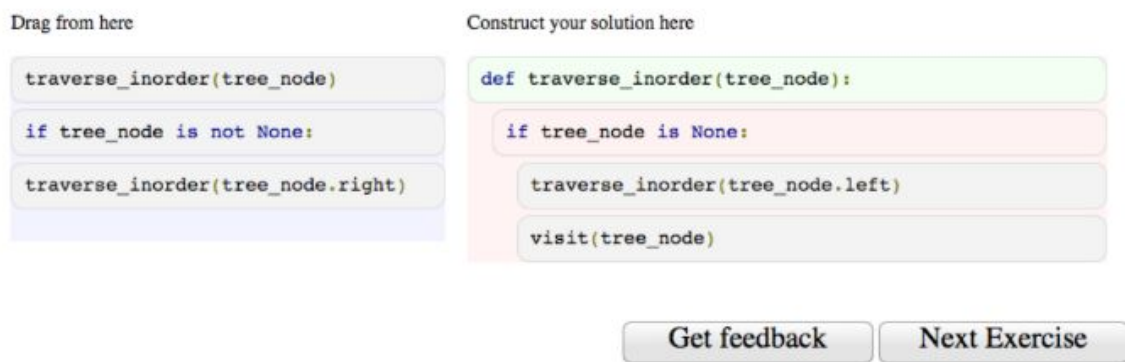


Fig. 2C. A 2-dimensional Parson's problem using distractor lines developed by Ihantola and Karavirta (2011)

A variation presented by Fabic et al. (2019) (Fig. 2D) involves both rearranging lines of code as well as filling in some missing content from some of the lines. Their justification for this is that “Solving Parsons problems with incomplete LOCs [lines of code] may be closer to enhancing code writing skills than Parsons problems with/without distractors.” They further this reasoning in referencing the study by Harms et al. (2016) which indicates how distractors may decrease learning efficiency for students aged 10-15.



Fig. 2D. GUI for the Parson's problem system used by Fabic et al. (2019) showing lines to be sorted and gaps in code to be filled.

When drawing conclusions on the development of their mobile Parson's problems system (Fig. 2E) Karavirta et al. (2012) state how that a future system could be developed to be more “engaging and addictive” by “assigning scores to solutions, rewarding learners for achievements such as completing

many assignments or using little time to complete assignments.” Intuitively, it can be seen that there could be great potential in a system that actively employs means to engage the user.

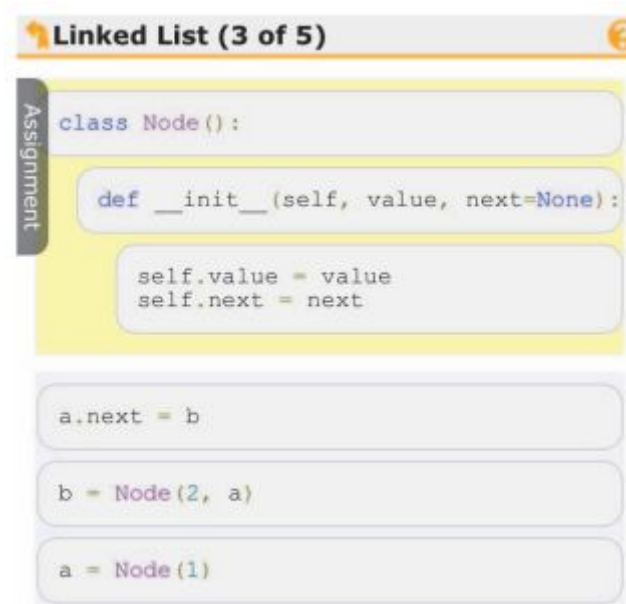


Fig. 2E. Mobile Parson's problem implemented by
Karavirta et al. (2012)

An important design consideration mentioned by Helminen et al. (2013) is: how “there may exist multiple possible arrangements [of lines of code] that meet the requirements [of the Parson's problem]. Avoiding such situations is one of the challenges in designing non-trivial Parsons problems”. What this means, is that for any given problem description with a given set of lines to form the solution, there may exist multiple permutations in which the lines can be ordered to solve the problem. For example, if a problem consists of three lines which define two variables and then defines a third variable as the sum of the first two variables, there would be two ways to correctly solve this, however if the two lines which define the first two variables are in the same code segment, then there would only be one solution. Perhaps this could be managed through the use of immovable code already placed in the answer space so that the user can fill in the gaps with given lines. Helminen et al. mention how this can be used to tie “problems to a larger context and allows for more complex programs without making the task too difficult”, but also by restricting the structure of a solution one can also reduce the number of permutations of correct solutions. Parsons and Haden (2006) corroborate this practice when they discuss how “draggable fragments may contain single or multiple lines of code.”

It has been discussed by Parsons and Haden (2006) how certain aesthetic factors can affect the engagement of students, which can include the use of colours, sound and rewards for completing problems. They aimed to design Parson's problems under the principles of: maximal engagement, constraining logic, considering common mistakes, modelling good code, and to provide immediate feedback. These principles can be seen to hold a significant influence over the development of Parson's problem systems when looking at the development of system GUIs since the problem type's inception in 2006 to some of the latest developed Parson's problems systems.

2.1.1 Feedback Design

A prominent consideration in the design of Parson's problems is the way in which a sub-system for providing feedback to the user about their solution attempt functions. Helminen et al. (2013) point to the academic impetus for an effective feedback system when using Parson's problems, as they can be used for the automatic assessment and education of large groups of learners. Parsons and Haden (2006) identify the difficulty of providing precise feedback for each permutation of incorrect responses to a Parson's problem, which is further recognised when, of the 17 students surveyed in their study, 14 students expressed a desire for “more detailed feedback about errors made”. A suggestion that Parsons and Haden present is to “provide a general explanation of the issue addressed by each puzzle”, however it can be seen how this still does not address, explicitly, the reasons an attempted solution may be incorrect.

Furthering discussion on feedback, Karavirta et al. (2012) identify two major potential problems, when implementing an automatic feedback system: (1) the student using trial-and-error by excessively requesting feedback, (2) feedback not enabling the user to avoid looping solutions (returning to solution states that they have already visited). With such problems in mind, they present a preemptive feedback system, whereby the system may give “subtle feedback when students return to a state in the solution they have visited before” as well as providing feedback based on the order of the lines in the presented solution which, when used, incurs a time penalty to the user's score. It can be seen how this provides more specific feedback based on the users' attempt, rather than providing a general explanation of the topic the problem regards. A gap in present understanding is found in that it is not clear whether one method is necessarily more effective than another.

Helminen et al. (2013) discuss how the results from their study suggest that feedback based on execution encourages students to ensure code is correct and runnable and to think about the running of the program as a whole rather than when using line-based feedback which puts more focus on the arrangement of individual lines. While the authors are not convinced either method is necessarily

better than the other, this could indicate how that Parson's problems could be a more effective tool if their purpose is to provide a practical and realistic simulation of programming situations in order to deepen a novice programmer's knowledge, rather than just indicating which lines are in the wrong positions.

Perhaps this could suggest that a middle way solution could be to associate an 'error message' with each distractor line that could be presented to the user after submitting a solution that uses a distractor line. For example if the distractor line is missing a semicolon, the error response given on submission could be "line is missing a semicolon".

2.1.2 Distractors

Harms et al. (2016) find that the use of distractors, and the increased mental load that comes with using distractors, provides a decrease in the learning efficiency of novice programmers (within the researched age range of 10 - 15 years old). As such, Harms et al. encourage educators to be cautious in their use of distractors and encourage further research on circumstances where distractors offer a clear benefit. This could spur further discussion on the purpose of electronic pedagogy within this context: is efficiency of learning necessarily as or more important to the development of a programming student as depth of learning? It may be expected that in using distractor lines, the Parson's problem more realistically recreates a practical code editing situation which requires the identifying of erroneous lines of code. This is corroborated by Helminen et al. (2013) who highlight the use of distractors as a means for controlling difficulty when discussing how "they can also be used to drill students' syntactical knowledge by providing alternative versions of a fragment with some minor syntactical errors."

2.2 Why Parson's Problems Are Used

It can be seen that Parson's problems have been used for a wide variety of purposes with the intention of achieving different goals. The ways in which Parson's problems are used should be evaluated in order to develop an understanding of their effectiveness in different contexts.

2.2.1 Why Parson's Problems Over Other Question Types?

Denny et al. (2008) discuss using a pencil and paper variant of Parson's problems as an exam question through the impetus of them being easier to grade and as they capture the code creation process. In

their study they aimed to compare Parson's problems effectiveness with more traditional programming assessment questions such as code tracing (reading) and code writing. They concluded that there is a closer correlation between the code writing process and the process of solving Parson's problems than between code tracing and Parson's problems or between code tracing and code writing, and found that, generally, Parson's problems marks were higher than code writing marks, which were higher than code tracing marks. However, in reviewing student responses to the exam, they found that students found it difficult to provide answers to questions where they had to use the given lines from a Parson's problem and would sometimes rather write their own solutions, with students saying they prefer “to solve things my own way.” As well as this Denny et al. recognises how Parson's problems can be solved by students who don't fully understand the concepts the question covers but can guess at how the lines can be arranged, based on some small knowledge of syntax. They suggest that this could be mitigated by having multiple distractors for each valid line.

Perhaps one of the most salient conclusions drawn from Denny et al.'s study is described when they say that “When faced with a challenging [code] writing problem, some students do not know where to start and may leave the page completely blank – six out of our 74 students did. These students often do know something about code, but are not able to show what they know in a writing task. Parsons problems allow students the opportunity to show what they do know.” This is reinforced by the fact that Ericson et al. (2017) find in their study that solving 2-dimensional Parson's problems with distractors look less time with no impact on “learning performance, or in student retention of the knowledge one week later”, compared to exercises where the student has to fix distractor lines or has to write the equivalent code.

Parsons and Haden (2006) compare learning a programming language to learning a spoken language, in that both require having to acquire syntax and semantic knowledge. In discussing this they mention the limits of drill exercises, in that they are boring, and that it is hard to separate a “single syntactic unit from the logical context in which it occurs”. Parsons and Haden also further this discussion when they describe how using Parson's problems “exposes students to good programming practice”.

2.2.2 The Value of Parson's Problems to a Novice Programmer's Learning

The idea that a system using Parson's problems should be geared towards a beginner or novice programming students at all is reinforced by what is revealed by the analysis of a study conducted by Fabic et al. (2019) when they discuss how their use of Parson's problems were “especially effective for students with low prior knowledge”.

Garcia et al. (2018) also implicate Parson's problems as means for teaching the process of problem design when they discuss their study's initial findings which revealed how that students who didn't complete the Parson's problems used as a scaffolding tool to provide students with design guidance attained lower assessment scores. The authors admit a certain lacking in analysis in the impact of Parson's problems in this context, but this could suggest that Parson's problems could effectively be employed to develop a student's understanding of algorithm design, and as such it can be seen how, 'Algorithms' could be a useful topic to cover by a programming education system.

The use of Parson's problems as means for teaching higher level programming theory such as algorithm design can be further supported by Morrison et al. (2016) when they discuss the use of subgoal labels as a means for decomposing a task given in the form of a Parson's problem in order to reduce the cognitive load of the student solving the problem. The effectiveness of subgoal labelling provide an insight into how problem prompts could be structured to develop more cogently described problems; to the extent that for easier problems, the description may describe some of the problem's subgoals ("initialise variables," "define a function," "iterate through a list," "then return a value").

2.3 The Role of Student Modelling

Perhaps the most prevalent gap in most implementations of Parson's problems that are considered is the use of a student modelling system which can inform the difficulty of a problem served to a user, as most of the systems described by the literature work independently of any knowledge that can be gained about how the student answered previous problems.

Parsons and Haden (2006) discuss how they were aiming to develop 50 - 100 questions for each topic. A system which could implement a volume of even 50 questions per topic would allow for the development of a modelling system which could gain information about the user as they complete each question, so that more pedagogically appropriate questions are served to the user.

The EvoParsons system described by Bari et al. (2019) proposes a way by which "coevolutionary optimization" is used to evolve a set of preexisting Parson's problems in order to "generate pedagogically-sound Parsons puzzles for a population of novice programmers." This can be seen to be a form of student modelling as it uses the answers provided by the students in order to more appropriately cater to the student's abilities such that changes in the student's responses should impact the Parson's problems given by the system and changes in the system should be reflected in the student's subsequent responses. Such a technique is employed by Bari et al. as an attempt to develop

assignments that fall within the “zone of proximal development” which should provide students with problems that are neither too difficult or trivial for them to solve.

3. Methodology & Management

In conducting this project, risks that can threaten progress should be considered. This chapter considers the project management techniques employed to mitigate risk within the project and maintain project progression in Section 3.1; and in Section 3.2, the software process model by which this project follows is described and justified.

3.1 Project Management for Risk Mitigation

Two primary risks pose the biggest threat to preventing successful project development. The project must be completed within a time constraint and to tackle this different techniques have been employed. The first was to schedule seven dedicated two hour work periods per week on a personal calendar in order to maintain constant work. In practice not every scheduled session was completed, but they served as a constant reminder that near-daily progress on the project should be made. A second time management technique used was to schedule fortnightly meetings with the project supervisor. These meetings became weekly when approaching the project deadline, and were useful as they allowed for dedicating extended periods of time to developing specific parts of the project which could gain specific feedback from the supervisor.

Secondly, the limits of the author's expertise, and the limit to the amount of new knowledge the author can acquire within a reasonable amount of time is a risk must be considered. The primary deficit in the author's knowledge before starting the project was in using the Android Studio IDE/Android development paradigm. As the author was already confident in using the Java programming language, learning to apply this skill to Android development became straightforward by accessing tutorials on building android apps. This risk is further mitigated by aiming to start the implementation of the system as early as was reasonably possible. This enabled the ability to produce a prototype-like system that could then be extended and improved as development progresses.

Both of these two risks, time and expertise management, are associated with a certain level of severity and consequence. Should time not be effectively managed then it is obvious to see how not all of the intended project work would be completed by the deadline. This risk was realized in the form of not being able to conduct the planned observational study with programming students due to the

unexpected closure of the university during the time the study was intended to take place. This could have been avoided by anticipating the utility of the observational study earlier in the lifecycle of the project so that the study could have taken place earlier. Failure to acquire the necessary skills for development could have resulted in similar consequences in that if too much time was spent on learning new skills then not enough time would have been available for development, and inversely if not enough time was spent on learning new skills the quality of the system would likely suffer. Fortunately the author was able to develop many of the necessary skills for Android development, parallel to the time in which earlier processes such as the project initiation, literature review and initial design work, so that these skills could be used once the implementation stage of development had been reached.

3.2 Software Process Model

While the original process model was initially intended to be purely Waterfall, as indicated by the Gantt charts developed for the PID, inevitably, some of the restrictions of the Waterfall Model were broken in order to more efficiently make smaller adjustments to previous tasks as the project evolved. An example of this is that the requirements for the system were set out early in the development cycle, but then were then refined as the implementation took place, after a better understanding of the context of the system had been developed.

The actual software process model employed could be more accurately described as a bespoke hybrid waterfall/concurrent process model using a partial cyclical/iterative technique (Fig. 3A). The model employed cannot be accurately described only by either the Concurrent Process Model, the Waterfall model, or a Spiral/Agile Model on their own as it employs features of each of these. Elements of a 'Waterfall' process is used in that requirements definitions took place before the implementation, which took place before testing; elements of a 'Concurrent' process is used in that report writing took place simultaneously with implementation; and elements of an 'Agile' process was used in that cycles of review, design, and code implementation took place as part of the implementation process.

Use of this process model is justified by its flexibility as well as providing a structure that can help to identify the order in which certain tasks must be done. The consequence of this is that system requirements can be understood, which can inform the design process, which then informs the implementation and testing process, which informs the evaluation process, which allows for the writing of meaningful conclusions on the project. A further benefit to using this process model is that a basic code artifact was able to be developed earlier in the process than when using a totally

Waterfall based model. A totally ‘Agile’ methodology would not have been appropriate to use for this project as they are more conducive to teamwork environments where every aspect of development is revisited in cycles, whereas this project sought to complete objectives such as the literature review (which took place as part of the ‘requirements definition’ task) and the evaluation against requirements (which took place as part of the report writing task), and only revisit them as necessary.

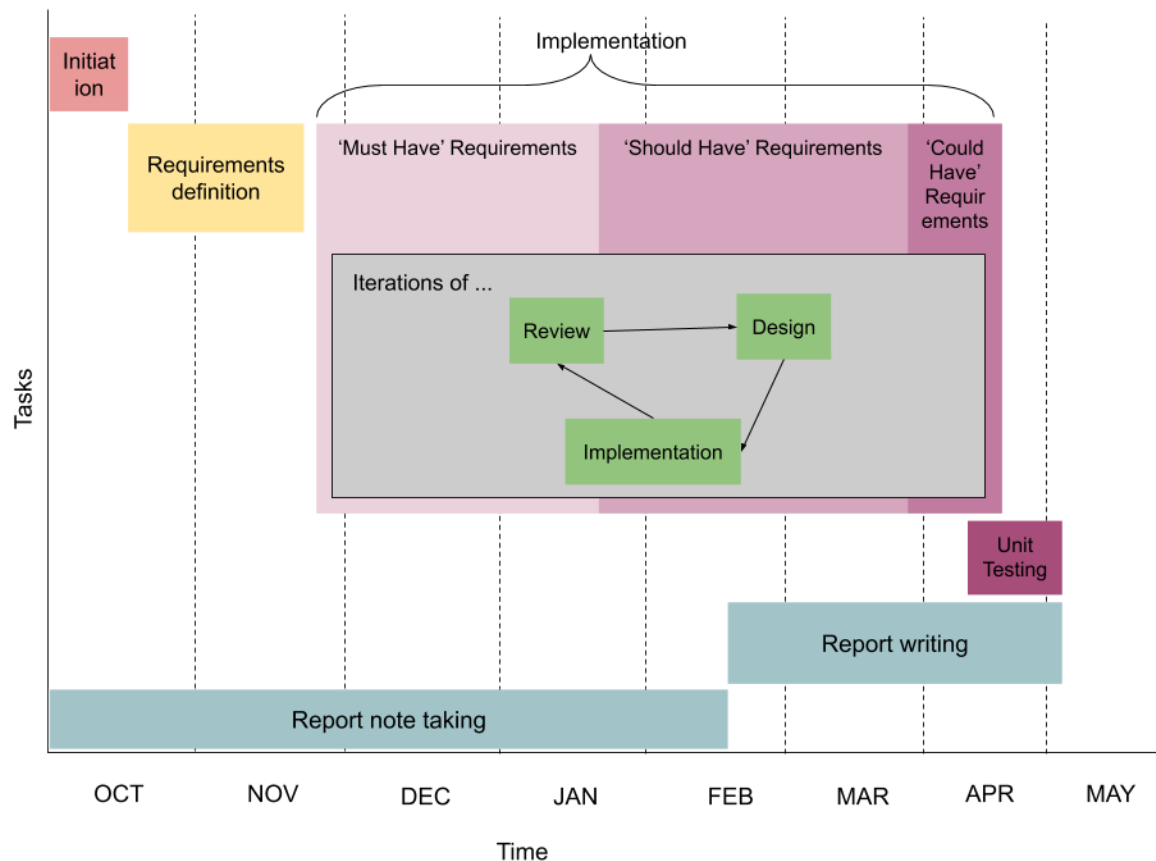


Fig. 3A Process chart that more accurately describes the completion of development processes throughout the project compared to the initial Gantt chart as described in the PID (Appendix A Section 7.2.1).

4. Requirements Engineering

This chapter aims to detail the process by which the requirements for the developed app are engineered. In Section 4.1 the process by which user requirements are gained is described and in Section 4.2 the elicited requirements are described, specified and prioritised.

4.1 Requirements Elicitation

At the proposal stage of the project a series of ‘general requirements’ were established by developing a general understanding of what Parson’s problems are by conducting a brief survey of existing mobile programming education tools (Appendix A Section 5.2, Appendix D). By doing this it was able to be understood - what a novel mobile informal programming learning app that uses Parson’s problems is required to do to (in a general sense) be effective. This however is not sufficient in order to be able to develop a system that is testable and verifiable against a precise set of conditions. Instead, specific requirements that originate from some sort of evidence must be found and then specified at a technical level when appropriate.

The formal elicitation process for this project uses the knowledge gained through conducting a literature review (Chapter 2) to establish a set of requirements that are then prioritised using the MoSCoW scheme. Doing this enables the identification of requirements which ‘Must’, ‘Should’, and ‘Could’ be satisfied to ensure project success. By using this prioritisation scheme the project can be more effectively managed as more essential requirements can be targeted to be completed earlier in development.

When describing the requirements, it can be identified whether a requirement belongs to the class of functional requirements (which describe the system’s behavior) or non-functional requirements (which describe the criteria under which the system operates). Doing so enables a more refined analysis of the extent to which the final developed system satisfies the requirements which can provide a more accurate evaluation of whether or not the system would be an effective programming education tool.

4.2 Requirements Specification

Section 4.2 is divided into three subsections which consider each of the three MoSCoW categories considered within the project. Each of the requirements for each of the sections contain a requirement code (e.g. MH1) for further reference, a title (in bold), a description, a source (generally stemming from the literature review), and a category (either functional or non-functional).

4.2.1 Must Have Requirements

MH1 - The system must present the user with a Parson's problem (functional). By using the graphical user interface, the system must enable the user to see the problem prompt, the set of valid lines, the set of distractor lines, an answer space, and buttons to aid interaction with the problem, such as a 'check answer' button and a 'next problem' button. The valid and distractor lines which are given should be in a randomised order (Source: Fig. 2A).

MH2 - The system must allow the user to drag lines into the answer space (functional). The user must be able to use the touch screen to interact with the GUI to touch and drag lines from a 'given lines' set space into an answer space, such that the order of the lines placed in the 'answer space' form the user's attempt at the problem (Source: Section 2.1).

MH3 - The system must verify the correctness of the user's attempt at a Parson's problem (functional). The user must be able to click a button as part of the GUI in order to verify whether or not their attempt at the problem is correct. The GUI must make use of some form of visual signifier to establish whether or not the attempted solution to the problem is correct (Source: Section 2.1).

MH4 - The system must make use of a touch screen interface on an Android based device (non-functional). The system must be able to conduct all user-based interactions through the use of the touch screen on a mobile device using the Android operating system (Source: Section 1.1).

MH5 - The system must make use of the Java programming language as the example problem language (non-functional). The programming language that the user is required to use to arrange the lines to solve the Parson's problem should be valid and correct Java code (Source: independent justification; Java is a widely used programming language).

4.2.2 Should Have Requirements

SH1 - The system should load the Parson's Problems from a statically defined file (functional).

When programmatically displaying a Parson's problem, the content of the problem should be retrieved from a text file that is structured in such a way that enables the development of a parser to read the problem information that can be used to present the problem. This information must be statically defined in a file in order to enable scalability in the development of multiple problems (Source: Section 4.2.2 SH2).

SH2 - The system should store and use at least 30 problems for each problem topic (functional).

This enables the development of a means for modelling the student's capabilities in completing problems concerning different topics within programming. The number: 30 is chosen as a scaling down from the number of 50 - 100 proposed by Parsons & Haden (2006) in order to account for the limitations and scope of the project (Source: Section 2.3).

SH3 - The system should organise the problems comprehensively by topic and difficulty (non-functional). This is required in order to programmatically retrieve a single specific problem from a set of many, based on the topic the user has selected to study and the skill level of the user for that topic (Source: Section 4.2.2 SH2, Section 4.2.2 SH1).

SH4 - The system should present the user with problems reflective of their skill, and change in skill over time, in the concerning topic (functional). When presenting the user with a new Parson's problem, it should suitably match the user's skill within the topic of that problem. This describes the way by which the user is modelled as part of the system. As the user successfully completes a problem, their skill level in that topic is incremented. As the user incorrectly answers a problem, their skill level in that topic is decremented. As a result the students skill in a topic does not change if they successfully answer the problem on their second attempt - which provides a means for effectively modelling the user's skill (Source: Section 2.3).

SH5 - The system should save skill level to a local file (functional). The system should record to a local file the progress the user has made within the completion of different levels of difficulty for each of the problem topics, such that when the user closes the app, they can return to the app and be presented with problems at the same level of difficulty as they achieved in their previous use of the app (Source: Section 4.2.2 SH4).

SH6 - The difficulty of the problems should be designed to be appropriate for beginner programmers (non-functional). The problems should be completable by users who are aware of common programming concepts, such as variables and control structures, but who are perhaps novice Java programmers. While higher difficulty-level problems may not immediately be solvable by beginner programmers, they should be solvable by users who have completed lower difficulty-level problems (Source: Section 2.2.2).

SH7 - The GUI should consider common usability concerns (non-functional). Care should be taken such that the GUI for the app considers the effects that colour-blindness, minor physical impairments, and minor vision impairments may have on user experience. As such: the colours red and green should not be used in close proximity to each other to differentiate screen items; touch interactive screen items should be appropriately sized; and text should not be needlessly small (Source: Section 4.2.1 MH1, Section 4.2.1 MH2, Section 2.1).

SH8 - The system should provide a feedback message after attempting to complete the given problem (functional). In the most basic case, the feedback subsystem should indicate to the user that successfully completed the problem and can progress to the next problem. When the user incorrectly attempts to answer the problem, different feedback messages should be presented to the user in order to indicate a way by which the user could improve their answer in order to correctly complete it (Source: Section 2.1.1).

SH 9 - The system should be designed with an awareness of the implications of using distractors (non-functional). Each problem should use no more distractor lines than the number of valid lines required to solve the problem. Distractor lines should be used in order to explicitly ‘test out’ an area of programming understanding, so that the user is actively mentally engaged when answering the problem (Source: Section 2.1.2).

4.2.3 Could Have Requirements

CH1 - The system could introduce the necessity for the user to arrange lines horizontally as well as vertically (functional). This would present problems in a ‘2-dimensional’ form which would add an additional component of difficulty for the user as not only would they need to arrange the lines of code in a computationally valid order, but also to arrange the answer in a way that uses ‘best practice’ indentation (Source: Section 2.1).

CH2 - The system could make use of online user score storage, that can be accessed using Google Sign-In service (functional). A user could use their Google account as credentials for accessing a remotely stored score file. Doing so would allow a user to continue completing problems at the level they last attained, across multiple devices and across multiple instances of the app's installation (Source: Section 1.1).

CH3 - The system could further the Student Modelling procedure to consider multiplicity of skills which a problem from any one topic may develop (functional). Doing so would enable the development of a more accurate model for the student's capabilities, and consequently enable the system to serve more pedagogically appropriate problems. To do so, likely more than 30 questions for each topic would need to be developed as well as a scheme for identifying the multiple programming skills which a single problem would test for (Source: Section 2.3).

CH4- The system could allow the user to load custom Parson's problems from a text file (functional). The user would be able to select a directory on their mobile device that contains a series of marked up Parson's problems text files to be completed in the app. This could allow for an educator to set Parson's problems for students to load into the app and complete as part of an assignment (Source: novel inference on how to advance a Parson's problem education system).

5. Design

This chapter documents the process by which system components are designed. Section 5.1 presents an architectural design for the system, illustrating the shape of the system from a perspective that considers storage, logical, and graphical components. Section 5.2 presents the design process for the graphical interface from the wireframe design through to looking at alternatives and Section 5.3 describes and justifies the design for the student modelling algorithm.

5.1 System Architecture

The way in which the components that make up the system interact and exist within the system relative to each other should be understood in order to effectively implement the system. This process can be described by the development of a ‘System Architecture’ that lays out the structural model for the developed system in order to better understand exactly how the Parson’s problem app works in the way that it does.

From a high level, almost any user-oriented system can be described by analysing the interaction between components that manage system data, system logic, and system interface (Fig. 5A). The system developed in this project is not an exception to this generalisation, as the system will need to be able to store Parson’s problems, present the user with a GUI to solve Parson’s problems, and have an intermediate means for facilitating the interactions between the data and the interface (logic, in the form of method calls).

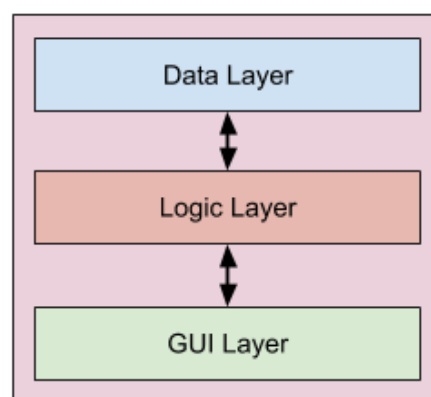


Fig. 5A. High level architectural model

While the model presented in Fig. 5A accurately describes the developed Parson's problem system, it is too general to precisely and meaningfully describe the way the system works. The architectural model presented in Fig. 5B 'zooms-in' on the model presented in Fig. 5A and describes in more detail how each component within the data, logic, and GUI layers interacts with other components.

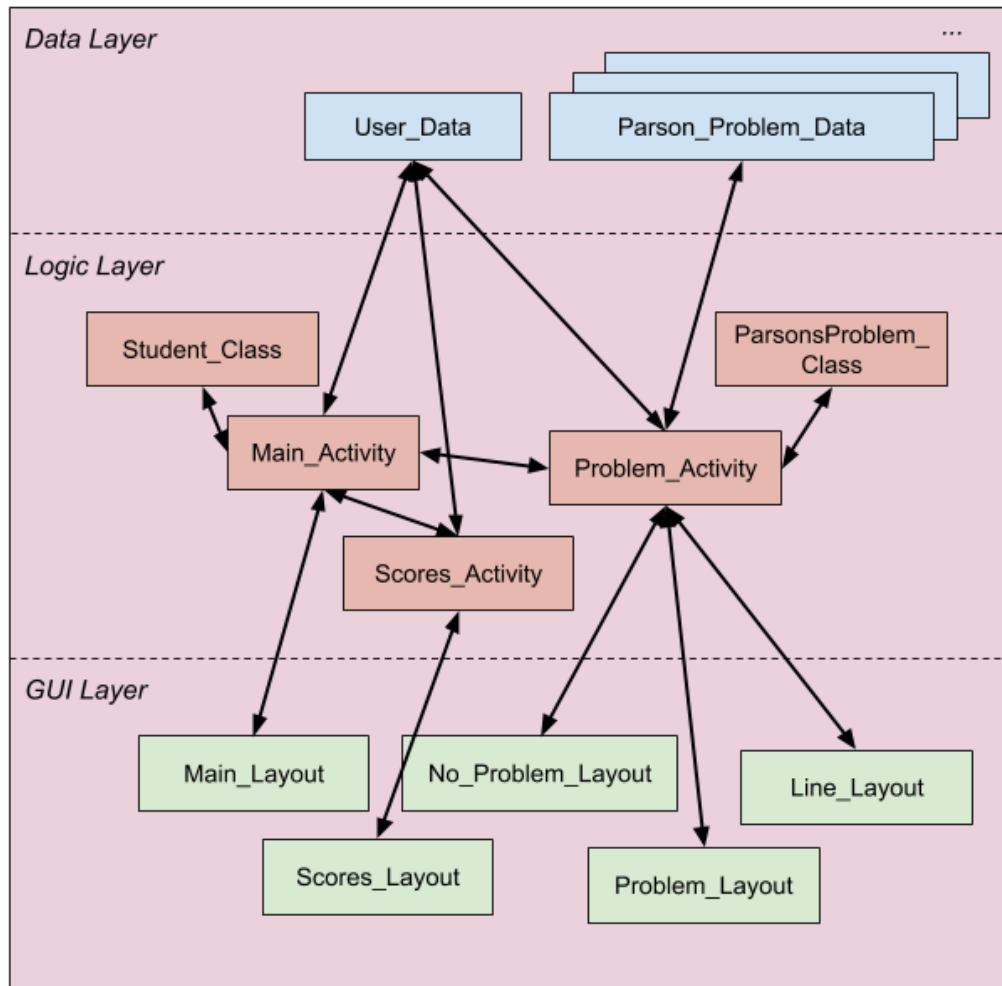


Fig. 5B. System Architectural Model (lower level).

In the 'Data Layer' of the model the interactions of the user data and Parson's problems data are described, and are ultimately represented by '.txt' files in the implementation. The 'User_Data' component stores the information which describes the user's skill level for each of the Parson's problems topics. It interacts with 'Main_Activity' and 'Problem_Activity' in order to provide the user with skill appropriate problems, and interacts with 'Scores_Activity' to facilitate presenting the user their scores for each problem topic. The 'Parsons_Problem_Data' components are used to describe the attributes which form each Parson's problem (the problem prompt, valid lines, and distractor lines) the user interacts with, and as such these components interact with the 'Problem_Activity'. There are

multiple of these components (as indicated by the ellipsis in the diagram) as each file corresponds to one Parson's problem and there many problems.

The 'Logic Layer' describes components which drive the way in which the system functions. Logic components ending with '_Activity' represent user-interacting classes and logic layer components ending with '_Class' indicate logical structures concerned with data representation but are not directly interacted with by the user. As such it can be seen that 'Main_Activity' interacts with the GUI component 'Main_Layout', 'Scores_Activity' interacts with 'Scores_Layout', and 'Problem_Activity' interacts with 'Problem_Layout', as each of the layouts describe the GUI screen for which each logical component uses. 'Problem_Activity' also interacts with the 'No_Problem_Layout' component in exceptional cases where the system attempts to fetch and present a problem that does not exist; and also interacts with the 'Line_Layout' component as the total number of lines for any given problem cannot be statically defined within the 'Problem_Layout' GUI component, so the 'Problem_Activity' component must use both the 'Problem_Layout' component in tandem with the 'Line_Layout' component dependant on the number of lines the problem has. In the implementation, the logic components are written using '.java' files.

The 'GUI Layer' represents components which the user uses to interact with the system i.e. they form the user interface by which the user can start Parson's problems, solve Parson's problems, and check their score. Such components may have statically defined features or features which are programmatically changed by the logic layer. The GUI components are implemented with '.xml' files.

5.2 User Interface

One of the most important factors in producing a user-interaction-based system is the process of user interface design. As this system employs a graphical user interface, this section of the report describes the processes that especially pertain to GUI development such as wireframe modelling (in Section 5.2.1) and producing and comparatively evaluating alternate GUI mock-up designs (in Section 5.2.2).

5.2.1 Wireframe Model

In order to understand how the system changes with user interaction a wireframe model of the system was constructed. Fig. 5C provides the wireframe model for this mobile Parson's problem system.

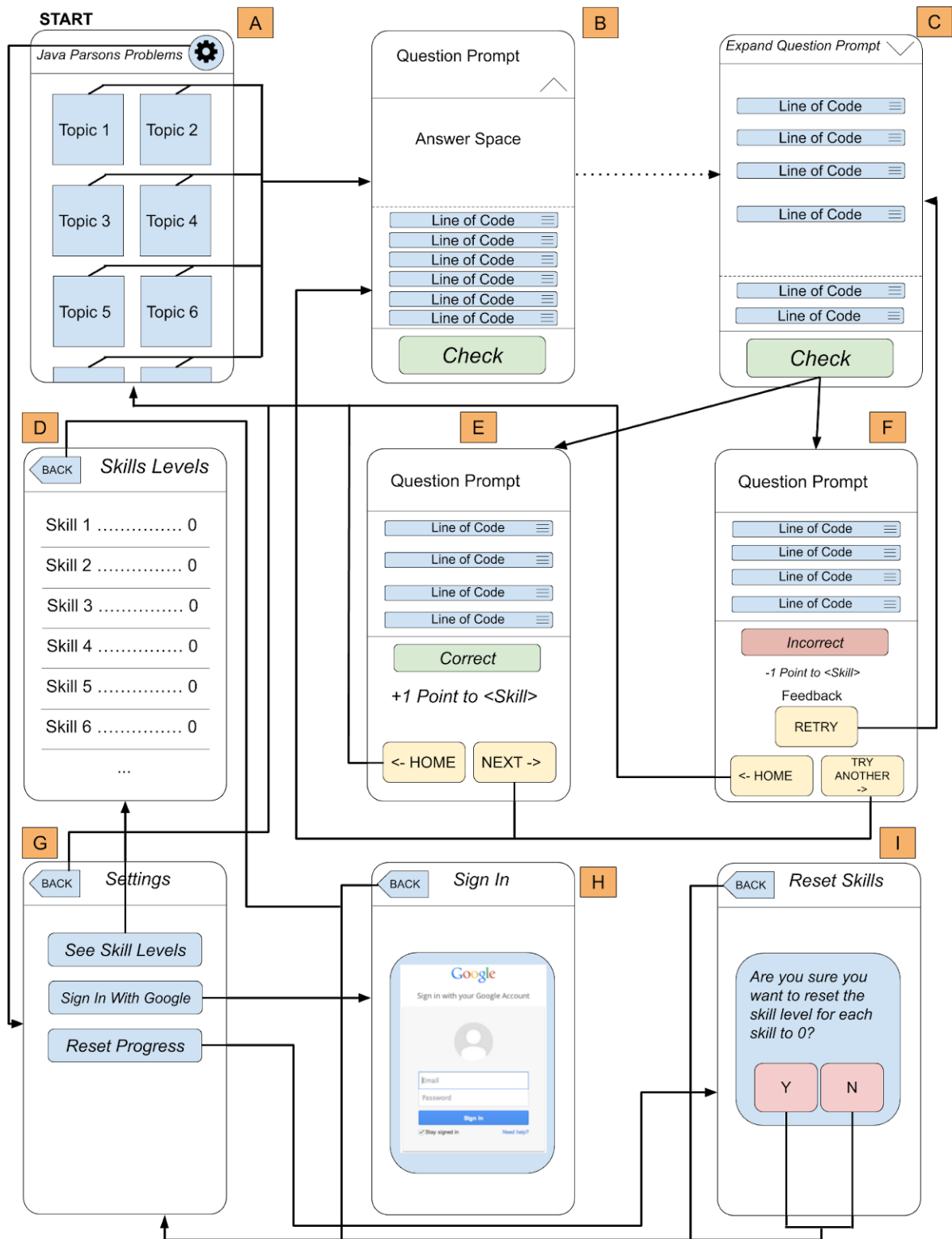


Fig. 5C. A wireframe model describing the interaction flow for the system.

Developing this wireframe assists with the understanding of the different activities that are needed, and what is needed from each activity, in order to satisfy the project requirements and to ensure the effective and intuitive navigation across the interactive components of the app. As typical with wireframe modelling, the stylistic factors of each screen in the diagram are less important to consider

here than the *structure* of each screen and how interactions flow from screen to screen. The more aesthetic factors of GUI design are considered in Section 5.2.2.

The process flow for the model starts with the screen in the top right of the model (Screen A), indicated by the ‘START’ label. This screen is what is presented to the user after the app is launched and should enable the user to initiate the core activities provided by the system (launch problems of different topics, access settings). Screens B and C indicate the problem activity before checking the correctness of the attempted solution. In the transition between B and C it can be seen how the user has moved some lines of code to the answer space as well as minimised the question prompt view. Screens E and F consider what happens when the user’s attempt is correct or incorrect (respectively) and Screen G describes a settings page which gives access to a skills levels page (Screen D), a Google Sign-In page (Screen H), and a reset skills page (Screen I).

5.2.2 Comparing Alternative GUI Designs

While the design for each screen in the wireframe model provides a basic GUI that can be used to deliver some of the requirements of the project, these designs can be improved upon by applying more thought to aesthetic and usability factors to improve the GUI. This section will describe and justify those improvements.

One of these improvements is to make effective use of colour to indicate state changes within the app after some user interaction. This is best made use of in the activity where the user is solving a Parsons problem, as colours can be used to indicate which lines are correct, incorrect, or of undetermined correctness. Fig. 5D shows how a blue colour is used to indicate the answer space, and how lines of code have their background colour set to white when they are placed into the answer space. This provides a form of visual information and feedback to the user as they are placing lines.

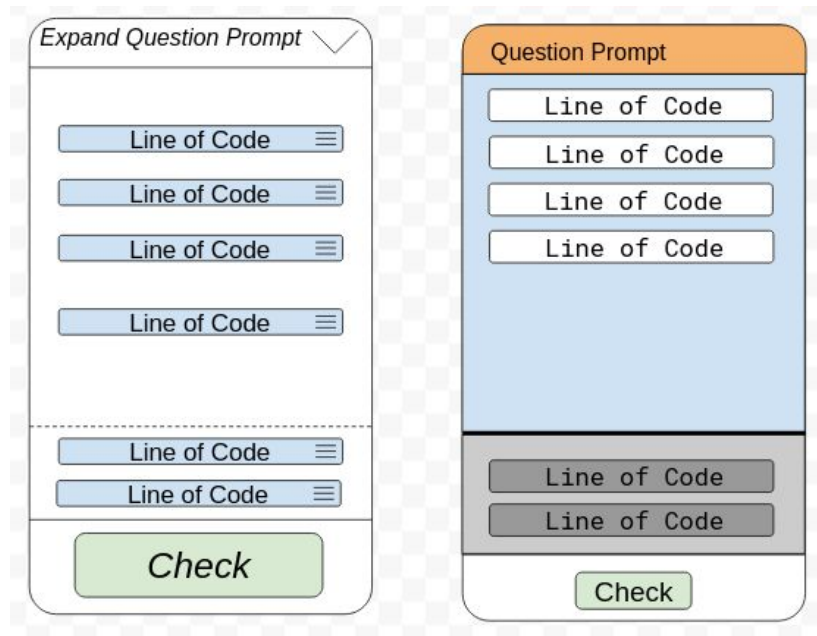


Fig. 5D. Pre-answer-check problem screen.

This principle of visual feedback should be continued in the screens after the user has selected to check the answer to their solution in order to provide some information on the correctness of their attempt. In Fig. 5E and Fig. 5F it is demonstrated how a yellow colour can be used to indicate that a line is correct (in the correct position to form the answer to the solution), and the same grey colour as the lines in the ‘given lines’ space can be used to indicate an incorrect line. These colours are chosen against the more intuitive green for correct and red for incorrect in order to consider the effects of red-green colourblindness.

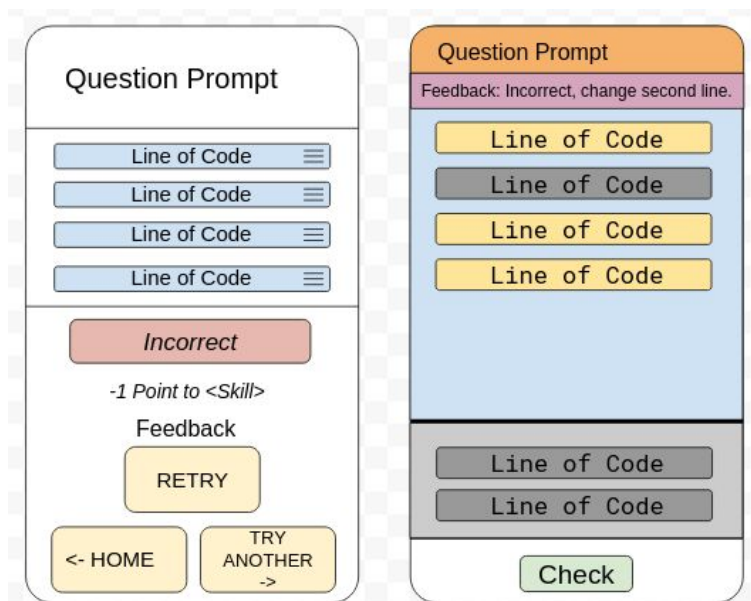


Fig. 5E. Incorrect problem attempt screen.

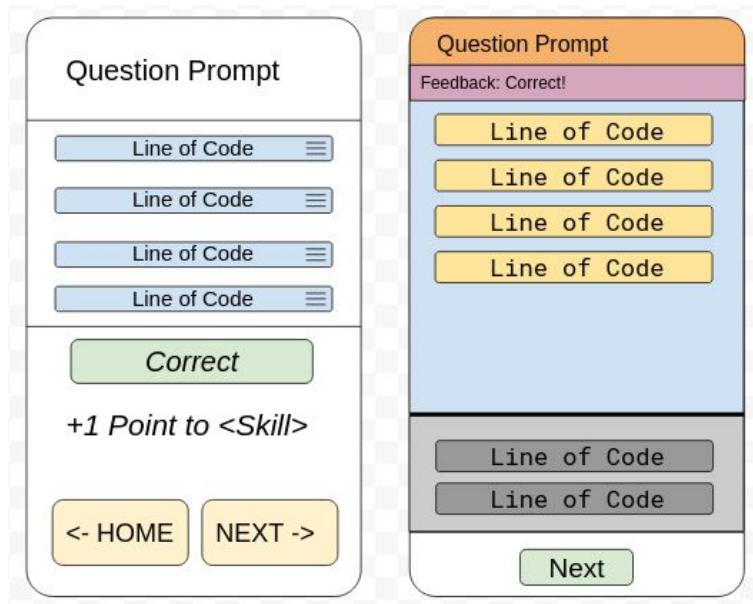


Fig. 5F. Correct problem attempt screen.

As well as colour indicators, the Parson's problems screens can be improved by making some simplifications such as reducing the screen space taken up by the prompt to that minimisation is not needed and reducing the size of the 'check' answer button in order to provide more space for lines of code to form the answer to the problem. This will allow for the forming of more complex problems that may require more lines to form the solution. It is worthwhile noting, also, that the design decision for having a prompt and answer space above the set of given lines in a singular column as opposed to the design using 2 columns in Fig. 2A is made in order to cater to the portrait nature of a smartphone held in the upright/vertical position.

As well as considering the use of colour in the problem screens, the GUI can be improved by simplifying the settings page interactions, so that viewing and resetting skill levels can be accessed from the same screen. This removes the necessity to have a dedicated settings page, and enables the user to sign in and view scores from the same 'main screen' that they can access the problems from, as indicated in Fig. 5G and Fig. 5H. This is justified by way of the fact that it is convenient to have the scores reset button on the same screen that can view the scores and that having a dedicated settings page to access only two settings (sign in and see/reset scores) is wasteful, in terms of the number of interactions required to complete tasks within the app.

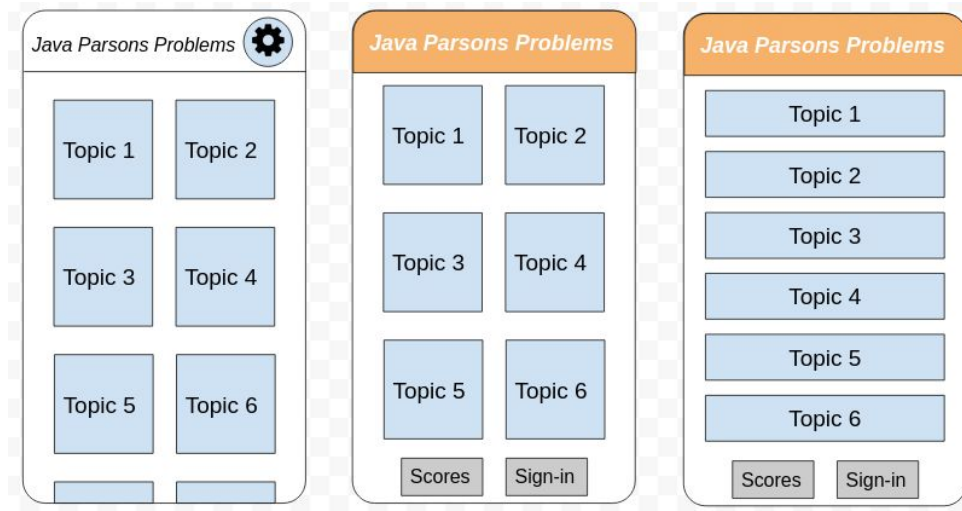


Fig. 5G. Main screen alternate designs

In evaluating a 2-column or 1-column approach to the topic buttons which launch Parson's problems for each topic (Fig. 5G), the benefits of each can be seen. While the 1-column approach may offer a greater sense of linearity or progression through a series of topics, the 'squareness' of the 2-column approach's buttons may enable a greater ease of use when selecting a topic. While the single column may be more conducive to scrolling, should a greater number of topics be added in the future, it would still be possible to enable scrolling using 2-columns of topic buttons. With this in mind, it can be seen how in order to place a greater focus on usability, the 2-column topic button approach should be used.

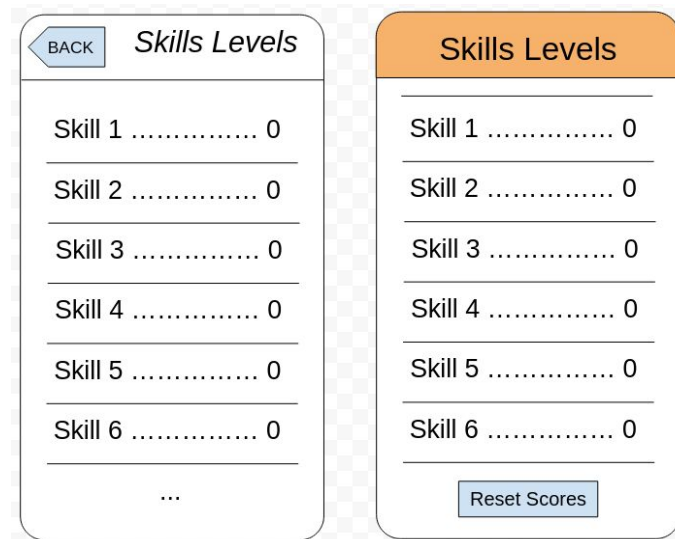


Fig. 5H. Skills page alternate design.

Additional, more minor, GUI design decisions include consistently featuring a predominantly orange and blue colour scheme as to correspond with colours prominent in the Java programming language

logo. As well as this, ‘back buttons’ used to return to a previous screen can be removed in place of using the native Android back button which can complete the same task while using less screen space.

5.3 Student Modelling Algorithm Design

One of the key algorithms this system uses is the algorithm which drives the student modelling process. By using a flow chart (Fig. 5I), the design for this algorithm can be described in such a way that is easy to read.

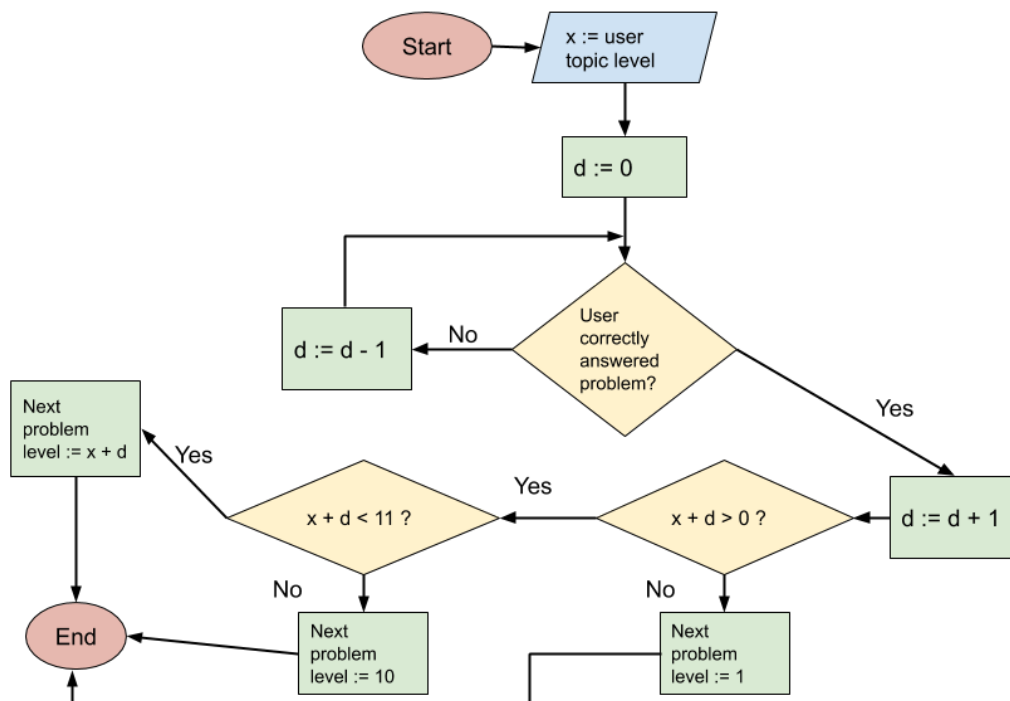


Fig. 5I. A flowchart design for the student modelling algorithm.

This flowchart defines the value for the difficulty of the next problem the user is given after they successfully complete the problem they are currently solving. The variable ‘x’, as indicated in the diagram, is the user’s topic skill level at the time of starting the problem, and consequently the skill level of the problem they are attempting to solve. The variable ‘d’ indicates the difference from the current problem skill to the next problem skill, and is decremented each time the user incorrectly answers the problem and always incremented when they eventually correctly answer the problem.

The algorithm can be simplified by the formula below (1) where n is the next problem level, x is the current problem level, and where $d = (-1 * \text{the number of incorrect attempts}) + 1$. The value for n is limited between 1 and 10 (inclusive) to cater to the intended number of difficulty levels for each topic.

$$n = \begin{cases} x + d, 0 < (x + d) < 11 \\ 1, (x + d) < 1 \\ 0, (x + d) > 10 \end{cases} \quad (1)$$

An intended side effect of this algorithm is that: should the user fail their first attempt and then succeed in their second attempt at solving the problem, their level for that topic is unchanged and they are served another problem of the same difficulty as the problem they just solved. This is to ensure that the user has been able to successfully answer, in one attempt, problems of all previous difficulty levels before providing the user with any more difficult problems.

This algorithm is chosen in part due to its simplicity and fairly intuitive quality, but also for the control that can be given to the implementer through modifying the constant in the ' $d := d + 1$ ' line of the flowchart to determine the algorithm's 'generosity'. When the constant is '1' it can be said that the algorithm is in its least generous state as the user must complete a problem on their first attempt to progress. However if this constant is set to '2' or '3' then it can be said that the generosity is increased as it affords the user to make one or two mistakes (respectively) and still progress onto the next problem.

6. Implementation

This chapter discusses and evaluates the process of the code artifact development and testing. In Section 6.1 the implementation process is broken up into individual system development issues whose solutions construct the code artifact for the system and in Section 6.2 the process by which testing of the code artifact is designed and implemented is described.

6.1 Issues and Solutions

This section states development issues and describes the ways in which technical methods have been used to solve them, consequently describing the implementation process. Such a development process features a chronology, simply due to the fact that some issues must be resolved in the implementation before others, as such the order in which the issues are presented closely match the order in which implementation issues were approached.

6.1.1 Parson's Problem Data Representation

For each Parson's problem to be programmatically accessible, the attributes which describe the problem but be described in such a way that code can be written in order to fetch the problem information then present it. The attributes that must be considered to fully describe a Parson's problem within the context of the system presented include: the problem prompt, the set of valid solution lines, the set of distractor lines, the order in which the solution lines must be placed, the problem difficulty, the problem topic, some means of uniquely identifying each problem.

Some of this information (such as the problem prompt) must be stored explicitly, however some of this information can be stored implicitly, in that 'the order of the lines' in itself does not require its own data representation, rather this implicit information is stored by the way in which the valid lines can be stored (in order). Furthering this, the problem difficulty, topic, and unique identifier do not need to be included in the text data of the file that describes the problem itself as this does not dictate the problem information that should be presented to the user. Rather, this information can be stored as the filename by which the problem is identified and indicates the location where each problem can be accessed. The filename schema chosen identifies each problem as such: XX_YY_ZZ.txt, where: XX

identifies the problem topic, YY identifies the problem difficulty, and ZZ identifies the variation of a problem for that topic and difficulty combination. Consequently, the filename for each problem is its unique identifier, and as such a text file can be used, using delimiting tag words ('[prompt]', '[valid lines]', etc), to describe each problem's information. An example for the representation of the second variant of a problem of the topic 'input/output' with a difficulty level of 1 is given below, and is uniquely identified by its file name: IO_01_02.txt

```
[prompt]
Print the words 'Water' and then ' Ice' on two separate lines
[valid lines]
System.out.println("Water");
System.out.println("Ice");
[distractors]
System.out.print("Water");
System.out.println(Ice);
[end]
```

6.1.2 GUI Markup

Before being able to present a user with a Parson's problem, the system must contain the necessary essential GUI components through which the user can select to initiate solving a problem. The first GUI component that should be described is the 'Home Screen', the screen that is presented to the user on launching the app which can be used to initial further activity. The design for this is presented in Fig. 5G and the implemented screen is presented in Fig 6A.

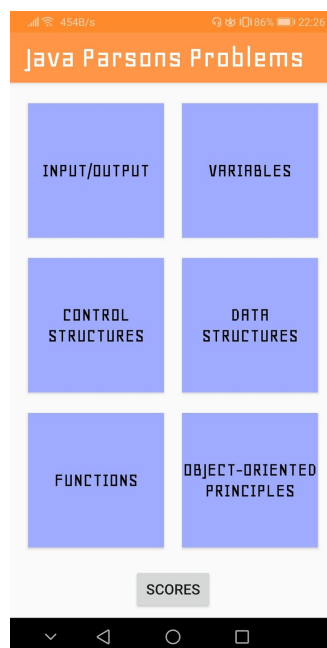


Fig. 6A. The implemented 'Home Screen'

The implemented screen closely follows the design but as a stylistic choice, a different font was chosen for the title bar and buttons. The Android Studio IDE has been effective in implementing designed GUI screens through using the graphical editor to intuitively move and edit GUI components as well as using the XML editor to more precisely make changes.

The development of a fully functioning GUI for the system is furthered by implementing the Problem activity screen (Fig 6B). This is the screen through which a user will solve each Parson's problem. A yellow TextView is created at the top of the screen to contain the problem prompt; a blue LinearLayout component is created below the problem prompt for the user to move lines into to provide the solution to the problem; a black TextView is used as a divider between the set of given lines and the answer space; and a grey LinearLayout component is used below the divider to contain the set of given lines which has a minimum height that is greater than the height of one line so that lines can be easily placed back into the given space.

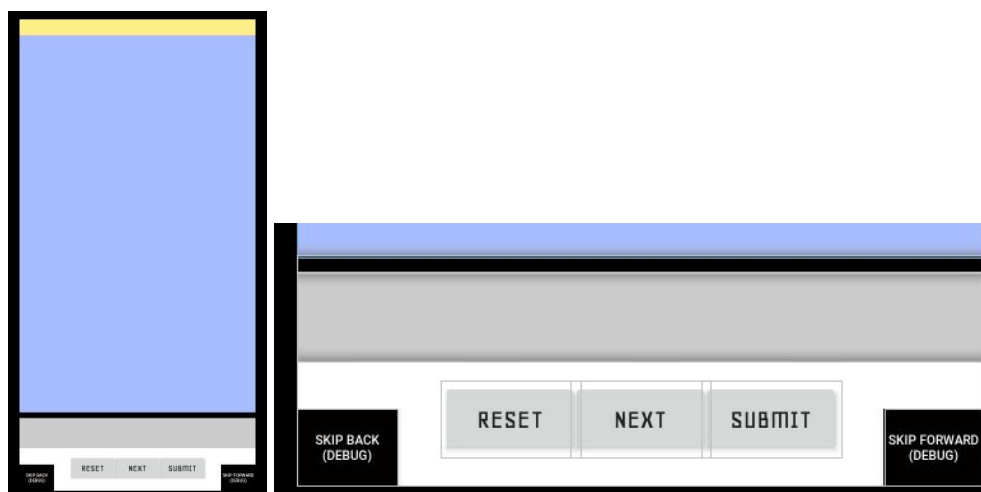


Fig. 6B. Design view for implemented problem activity screen.

As well as this, each of the necessary buttons needed to interact with the Parson's problems are implemented. The 'RESET' button is used to send all of the lines in the answer space back to the given answer space. This button was not included in the designs for the problem screens as its necessity was not apparent until the problem screen was being tested and it was observed that it can be frustrating to return each line from the answer space to the given line space individually. The 'SUBMIT' button is used to check the solution to the user's solution to the problem and will trigger the feedback system. The 'RESET' and 'SUBMIT' buttons are hidden after the user correctly answers the problem. The 'NEXT' button will launch the next problem whose difficulty is decided by the student modelling system but it is hidden until the user correctly answers the problem. In the bottom

left and right of the problem screen are skip forward and skip backwards buttons which have the effect of increasing or decreasing the user's topic score and progressing to a higher or lower skilled problem. These are used for debugging purposes to test problems of different difficulties and should be removed when using the app as part of a student's learning. As part of the error handling process for delivering problems, a layout is developed that considers the occurrence of the error of a problem of a given difficulty/topic combination not yet existing. This is described in the file 'no_problem.xml'.

Finally, a GUI must be implemented to present to the user their scores, as designed in Fig. 5H. This implementation is shown in Fig. 6C. The differences in the implementation from the design include the use of orange boxes to contain each topic score and using the '=' sign to space between the score and the topic name. Both of these help to ensure a visual consistency throughout the screen that enables easier reading. The same font: 'Geo' is used in the title bar as in the MainActivity to ensure a stylistic consistency.

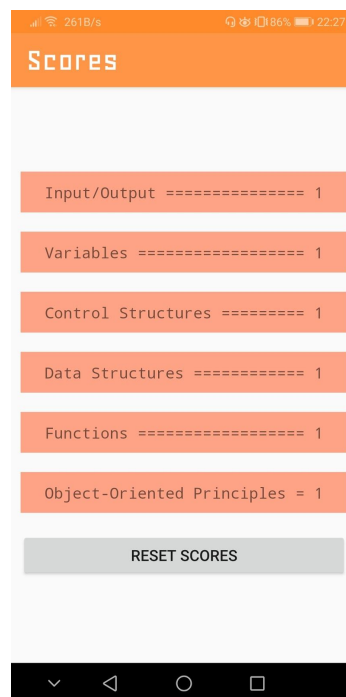


Fig. 6C. Implemented screen for user scores page.

6.1.3 Displaying a Problem

The approach taken to solve the problem of displaying a Parson's problem the user is broken down into four substages:

- i. Fetch the file containing the appropriate Parson's problem from the assets folder.

- ii. Parse the file in order to build a ParsonsProblem object.
- iii. ‘Inflate’ the line layout for every line in the Parson’s problem.
- iv. Use the ParsonsProblem object to set the text of GUI components in order to display the problem.

As the problem must be displayed before the user can interact with the problem, each of these steps must occur within the ‘onCreate’ method for the ProblemActivity class. Steps 1 and 2 take place within the ‘generateParsonsProblem’ method and step 4 is done using the ‘displayProblem’ method, both methods are in the ProblemActivity class. Step 3 takes place in the lines between where ‘generateParsonsProblem’ and ‘displayProblem’ are called in the ‘onCreate’ method of ProblemActivity as the number of line views required to display the problem is determined by how many lines the problem has (both valid lines and distractors). As such, step 3 uses the ‘Inflator’ class to reproduce the line_layout.xml (Fig. 6D) component for every line which comprises the problem. (The code for this process is described by the procedure under the comment “//Inflate the line_layout xml for every line in the problem” in the ‘onCreate’ method of the ProblemActivity class of the code artifact.)

```
<?xml version="1.0" encoding="utf-8"?>
<TextView
  xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/line_view"
    android:layout_width="match_parent"
    android:layout_height="45dip"
    android:layout_margin="3dip"
    android:background="@color/colorLine"
    android:fontFamily="@font/ubuntu_mono"
    android:gravity="center|fill"
    android:lineSpacingExtra="0sp"
    android:paddingLeft="8dip"
    android:paddingRight="8dip"
    android:text="LINE"
    android:textColor="@color/colorLineText"
    android:textSize="14sp">
</TextView>
```

Fig. 6D. XML for inflated line layout TextView

This method of programatically reproducing the line layout for every line in the problem was chosen as opposed to having many different layouts for problems of different line quantities primarily for the sake of future extensibility and ease of maintenance of the problem layout XML.

6.1.4 Dragging, Dropping and Ordering

To facilitate the dragging and dropping process, two methods: 'onTouch' and 'onDrag' are adapted from a resource available at from the D'CODE blogpost 'Android Drag and Drop' (2013). The 'onTouch' method triggers the 'onDrag' method when a line component that can be dragged (problem line) is touched. When the 'onDrag' method is called, it handles what happens when a line is being dragged depending on the discrete drag event, which can be either: drag started, drag entered, drag exited, drag ended, or when line is dropped. These two methods enable the dragging of TextViews between two layouts, which allows for (in this implementation) the user to drag the problem lines from the 'given layout' to the 'answer layout' to form their solution.

However, a limit to these methods as described by the 'Android Drag and Drop' (2013) resource is reached as they do not allow for the ordering of lines within a layout. To do this, the `DragEvent.ACTION_DROP`: case in the case switching procedure in the 'onDrag' method is modified to allow for the ordering of lines. The procedure that allows for the correct ordering of lines requires accessing the `DragEvent`'s Y coordinate position to determine where in the vertical axis the line that is being dragged is dropped. This procedure can be described as such, whereby child elements of a layout (problem lines) are indexed according to the order in which they exist in the layout (the first line in a layout is indexed 0):

- i. If the line is dragged into an empty layout then just insert it with no regard to position.
- ii. Else, if the line is dragged into a layout above the line at index 0 then insert the new line at index 0.
- iii. Else, if the line is dragged into the layout below the line with the greatest index then insert a new line at the end (this is by default for the `layout.add(view)` method).
- iv. Else we must iterate through the children of the layout until a line of a greater Y coordinate than the `DragEvent`'s Y position and then insert the new line before that line.

6.1.5 Checking Answer and Providing Appropriate Feedback

Feedback that the user is provided with should reflect the answer they provide, in order for the user to make informed successive attempts. The 'checkAnswer' method in `ProblemActivity` progresses through the lines in the answer layout, checking against the set of valid lines in the `ParsonsProblem` object. Lines have their background colour set to yellow if they are in the correct position, or are set to grey if they are not. A running count of the number of distractors in the user's solution is kept which

helps with the conditional testing phase of the ‘checkAnswer’ procedure which helps to identify which feedback (presented in the pink TextView) will be of most use to the user. The procedure runs as follows:

- i. First, if the answer has too few or too many lines, then set the feedback text to state as such.

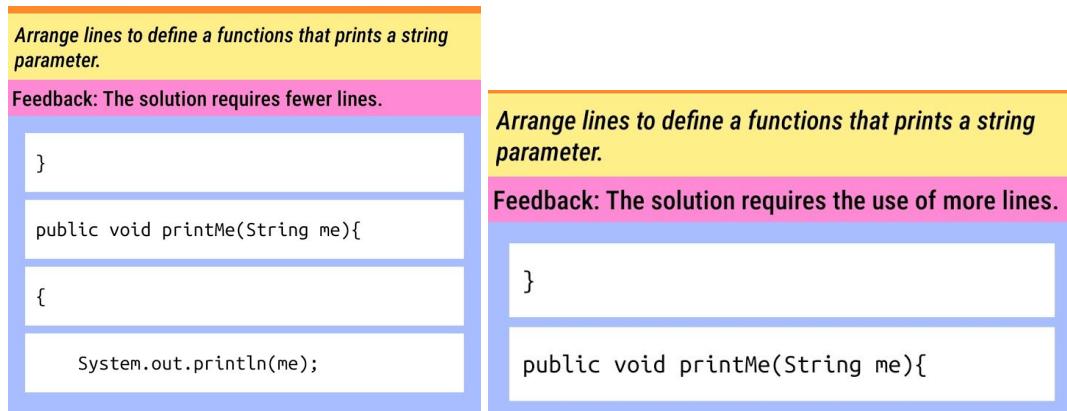


Fig. 6E. Feedback for when too few or too many lines are used.

- ii. Then, if the correct number of lines are used and all lines are in the correct order then the answer is deemed correct.

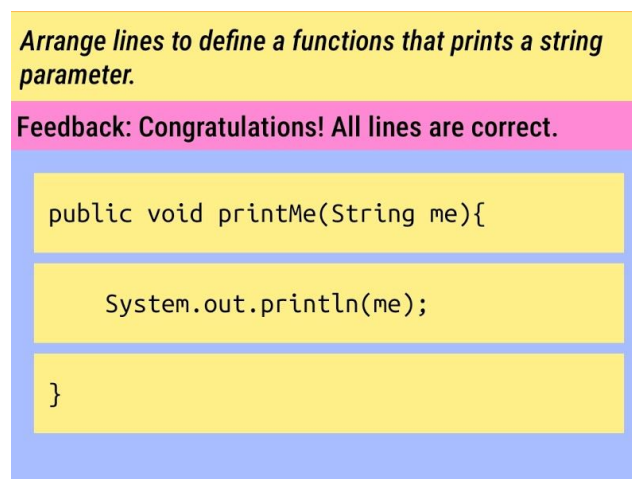


Fig. 6F. Feedback for when the answer can be deemed correct.

- iii. However if the correct number of lines are used and distractor count > 0 then the feedback states a distractor line is used, and will indicate as such using line colours.

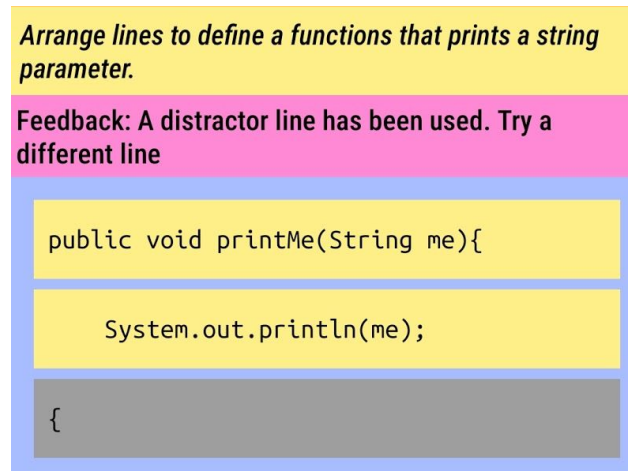


Fig. 6G. Feedback for when a distractor line has been used.

iv. And if correct lines are used and no distractors are used but order is wrong then feedback will state that correct lines have been used in the wrong order. The colour response of the lines will likely indicate which lines.

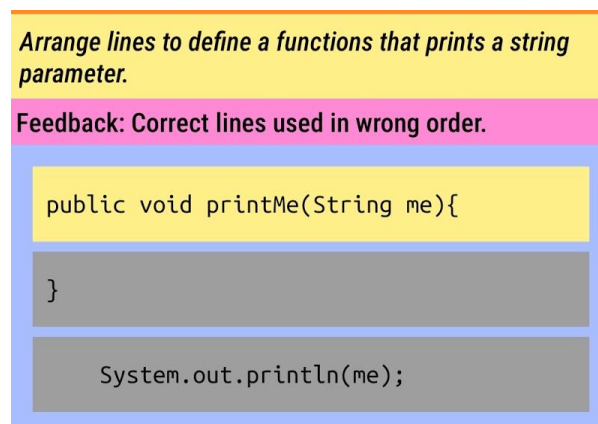


Fig. 6H. Feedback for when the correct lines are used in the wrong order.

6.1.6 Advancing Usability

As the GUI was designed with usability concerns in mind, it was intuitive to implement these designs to achieve a functioning GUI that catered to basic usability needs. However during the implementation process, more ways in which the interface could be improved became apparent and so appear despite not being included in the designs.

One such interface feature is the implementation of a 'RESET' button in the ProblemActivity screen which enables the quick return of lines from the answer space to the given line space. To achieve this, a simple routine is used to iterate through every item in the answer layout, remove it from the answer

layout, and then add it to the given layout. This is done quite simply by using a while loop that removes elements of the answer layout and places them in the given layout and ends on the condition that the answer layout is empty (has a child count of 0). This is implemented as the 'resetLines()' method of the 'ProblemActivity.java' class in the code artifact.

The user interface is further advanced by the use of sound that is played as a reaction to user interaction. One sound has been chosen to be played when navigation components are clicked (click.mp3); two sounds are used as part of the dragging interaction: one for when the dragging is initiated (up.mp3) and another for when the line being dragged is dropped (down.mp3); two sounds are also used as part of the feedback mechanism to indicate when the attempt is correct (correct.mp3) or incorrect (incorrect.mp3). A final sound is used for when the user selects to reset their scores (woosh.mp3) in order to emphasise the action of resetting their scores. To use these sounds, they are stored in the 'raw' subfolder of the 'res' subfolder of the app and are driven using the MediaPlayer class.

6.1.7 Student Modelling

The student modelling process is enabled through the use of a student skills file which is structured simply as the comma separated values of the user's skill level in each topic. Each of the skill levels are modified on the successful completion of a problem, based on the model presented in Fig. 5I. To do this, file reader and writer methods are written in order to modify the user's skill level file, and an instance variable in the ProblemActivity class is used to keep track of the number of failed attempts at the question to determine the change in user skill level and, subsequently, the difficulty of the next problem the user is served.

Bonus skill points for other topics can be gained for problems that test multiple multiple topics. This is done by tagging the bonus skill topic code, prefixed by a hash symbol (eg "#FUN" if the problem also tests for skills using functions, see Fig. 6I) in the question prompt so that the 'writeNewStudentLevels' method can check for any bonus skills to attribute points to. This bonus point is only attributed if the user successfully completes the problem within one attempt as to only attribute points where the user has demonstrated a definitive level of understanding.

Arrange lines to define a functions that prints a string parameter. Bonus Skill: #FUN.

Fig. 6I. Problem prompt for a problem in the category of 'I/O' that indicates the testing of a bonus skill for the topic of 'Functions'.

6.1.8 Code Maintenance (Documentation and Revision/Version Control)

To ensure effective development over an extended amount of time, certain implementation practices such as maintaining clear documentation and employing a revision control process should be adhered to. The implementation uses the JavaDoc notation to annotate every developed method to describe what each method does and its parameters and return variables. This documentation tool is chosen as the Android Studio IDE can automatically generate the documentation using HTML/CSS in a way that is easily readable. This documentation is contained in the '/docs/' directory of the repository.

The Git/GitHub revision control paradigm is chosen to maintain the project over time. The primary reason this specific paradigm is chosen is due to the author's familiarity. Using Git, commits can be made to mark the development of specific features, completion of requirements or other markers of progress. A branch for the development of each requirement prioritisation category (MustHave, ShouldHave, CouldHave) is used in order to provide milestone development checkpoints that signal the progression of the app. Once each of the requirements in the prioritisation category is completed, the changes can be committed and pushed to the working branch. A common series of git bash commands used to achieve this process of revision control are described as follows:

```
git checkout -b ShouldHaves
... changes to code take place ...
git add -A
git commit -m "Describe the changes."
git push origin ShouldHaves
```

Then GitHub is used to create a pull request of the working branch in order to merge changes from the working branch to the master branch.

6.2 Approach to Testing

This section considers the way by which the system's implementation is tested. It starts by outlining the necessity for a testing table which describes the parameters of testing the implementation and then justifies the use of automated JUnit testing as the means by which parts of the testing schema can be implemented and describes how methods that cannot be unit tested are validated.

As the project was not implemented using a test driven development strategy, whereby tests are written before methods which form the implementation, methods which are created must be revisited so that unit tests can be designed and then implemented. Designing a testing table (Appendix E) that can describe every test conducted allows for the systematic analysis of the correctness of how system procedures function for expected, boundary and anomalous inputs against an expected output. The testing table can then be used to easily develop automated unit testing methods (see next section). The use of unit testing is justified as it helps with ensuring that the way in which methods function is consistent even as developing the implementation may evolve. Unit testing is implemented using the principle of 'assertions' (in the JUnit scheme) which evaluate to 'true' if for some expected output the same is gained as the output for a method, and otherwise evaluates to 'false' causing the test to not pass. Each test case in the testing table is tied to a method in the 'GeneralTesting' class which is commented with the matching test ID. The structure of testing assertions are as follows:

```
assertEquals(expectedValue, actualValue);
```

However, unit testing will not be applicable to every method as not every method will have discrete inputs and outputs, but rather act to serve as the driver of some other procedures. This is true of the 'Activity' classes which largely consist of methods which take some GUI component as a parameter (e.g. a button) and perform some function (e.g. change the text of a TextView), but no data is returned at the end of the method. To test these methods then, a different approach is taken whereby each method responsible for some interactive procedure is used under varying combinations of conditions to test whether they function as expected. The mechanical tool that becomes most useful here is the Android Studio logging system as it can help to identify the reason why a method may not be exhibiting its expected behavior. One example of this 'debugging' process would be the resolution of a crash that would occur when invoking the ProblemActivity class which would produce an error message describing a 'NullPointerException' due to attempting to invoke a method on a 'null object reference'.

This error occurred due to not ensuring the 'ContentView' (of the correct layout) was not properly set before trying to interact with the GUI components, and so the 'onCreate' method was trying to interact with layout components it could not access. This was fixed by making sure that the 'setContentView' method was correctly invoked before attempting to interact with any GUI layout components. Through going over this process of reading the log output as crashes and bugs arise, and then fixing the bugs, the methods that cannot be unit tested can still be deemed as validated in an indirect way through demonstrating consistent and correct performance. The testing of more 'interactive' activity methods could have been more rigorously conducted by developing Android 'Instrumentation' tests and GUI tests that use the Espresso framework. This method was not chosen as it would have required a significant time investment to learn the framework that would have compromised the time allotted for other areas of the project, without producing any meaningful gains to the system's reliability or correctness.

7. Evaluation against Requirements

This chapter evaluates and discusses the quality to which system requirements (described in Section 4) are completed. Similarly to how the requirements are structured in Section 4, this chapter will progress through each requirement prioritisation category, using the requirement codes and stating the requirement title to associate specific analysis and evaluation with specific requirements.

7.1 Must Have Requirements

It is determined that the developed system completely satisfies all of the ‘Must Have’ requirements. A justification for why these requirements can be deemed as satisfied follows below.

MH1 - The system must present the user with a Parson’s problem (functional). This requirement has been fully satisfied as in using the system in such a way, the user is presented with an exercise that follows the form of conventional Parson’s problems. Each exercise presented to the user contains a prompt, a set of given lines that occasionally use distractor lines, and a differentiation of spaces for where answer lines and distractors lines should be placed (an answer space and given line space). One way by which the GUI could be improved would be to enable scrolling within the given and answer layouts to enable presenting the user with bigger problems (in terms of problem line count). In its current state, the maximum number of problem lines the system can present is dependent on the height of the device screen. As well as this the GUI could be improved by supporting use in the horizontal device orientation.

MH2 - The system must allow the user to drag lines into the answer space (functional). This requirement has been fully satisfied as the user can make use of a touchscreen device to touch and drag lines from the given lines space to the answer space, from the answer space to the given lines space, and from and to the same line space (answer space to answer space, given space to given space).

MH3 - The system must verify the correctness of the user’s attempt at a Parson’ problem (functional). This requirement has been fully satisfied as the system can determine if the user’s answer matches the solution described in the problem file. A minor concession needed to be made

here is that problems must be defined in such a way that only one ordering of given lines can be deemed as valid. This resulted in sometimes needing to make the problem prompts arbitrarily specific (see the problem described in file IO_02_01.txt where prompt states to define one variable before another when in a realistic scenario the order of definition would not necessarily matter.)

MH4 - The system must make use of a touch screen interface on an Android based device (non-functional). This requirement has been fully satisfied. This is enabled through developing the system using the Android Studio IDE, which is designed for developing apps on Android systems, as well as using an Honor 9 Lite Android based device to install the app onto the determine that the touch screen interface functions correctly. It could be argued that this requirement could be further evaluated by testing the functionality of the touch screen interface on different Android devices using different versions of Android OS, however this was not done due to a lack of resources.

MH5 - The system must make use of the Java programming language as the example problem language (non-functional). This requirement has been fully satisfied as the solutions to each of the problems make use of valid Java 11 code and were run in a development environment to ensure that each of the problem solution codes output what the prompt describes.

7.2 Should Have Requirements

It is determined that most of the ‘Should Have’ requirements are satisfied to a complete degree. The following includes justifications of why these requirements are satisfied and explains the reasoning for why some of the requirements (specifically SH2, SH6 and SH8) are completed to a slightly less-than-complete extent.

SH1 - The system should load the Parson's problems from a statically defined file (functional). This requirement has been fully satisfied as the system fetches and presents problems which are stored in ‘.txt’ files. Using a fairly simple bespoke mark-up scheme to describe a Parson’s problem, the system is able to correctly fetch and present the user a problem, provided that the correct notation rules for marking-up the problem and naming the file that contains the problem are followed. For maintenance purposes, these rules are described in the README.md file for the project under the ‘Conventions When Defining Problems’ heading.

SH2 - The system should store and use at least 30 problems for each problem topic (functional). This requirement has been satisfied to the extent that it proves the concept that it tries to establish. For

the topic of 'Input/Output' (I/O) 30 problems have been developed, 3 for each of the 10 levels of difficulty. This helps to demonstrate how having this many problems is useful for providing exercises of a varying difficulty, with some variation in context within the same level of difficulty. Consequently, this provides the basis for the student modelling system as the system can adapt to serve problems that match the user's skills in order to provide an appropriate challenge. 30 problems were not developed for *every* topic as this would become time consuming and would not provide any additional value to the project. Should this system be eventually used as a teaching aid then it would be appropriate to develop at least 30 problems for each of the topics that are taught.

SH3 - The system should organise the problems comprehensively by topic and difficulty (non-functional). This requirement has been fully satisfied as a structured scheme for the naming of files of different difficulties and topics is implemented such that problems of any topic and difficulty combination can be easily located. However this could be improved by separating problems into subfolders in the assets folder based on their topic. This would require a re-implementation of how problems are fetched but it would ensure a greater level of scalability as more problems for the system are developed.

SH4 - The system should present the user with problems reflective of their skill, and change in skill over time, in the concerning topic (functional). This requirement has been fully satisfied as the user is presented with problems which match their skill level. A fairly simple student modelling algorithm is developed as the means of changing the user's skill level and consequently causing the system to adapt to the user's interaction. This requirement could be improved by allowing the user to manually set the 'generosity' of the student modelling algorithm in order to allow for progression to higher levels even if the user makes a mistake on their first attempt, if the mistake is minor; or to keep the system as currently implemented where the user can progress only if they correctly solve a problem on their first attempt. This requirement expanded on and developed further in the requirement CH3.

SH5 - The system should save skill level to a local file (functional). This requirement has been fully satisfied as the system makes use of a text file to store the user's skill levels which enables resuming the completion of problems of an appropriate skill level after closing and reopening the app. This could be improved by introducing a mechanism by which a user can transfer their skill levels across devices either through the use of a remote/online system as described in requirement CH2 or through the use of a skill level export/import system where the user could export their skills to a file from one device and import these skill levels to another device.

SH6 - The difficulty of the problems should be designed to be appropriate for beginner programmers (non-functional). This requirement has been partially satisfied as contact was made with the project's moderator who was able to provide resources that describe the Java programming content learned by first year university programming students and as such problems could be developed that considers the limitations of catering to beginner programmers. This requirement could have been better validated by conducting the planned observational study which needed to be cancelled as described in Chapter 3.

SH7 - The GUI should consider common usability concerns (non-functional). This requirement has been fully satisfied through the use of a design process that considered usability concerns such as colour blindness. Feedback from the project supervisor and moderator helped to validate the usability of the app through verbal expressions of support or criticism of design decisions made in meetings. However this requirement lacks specific validation from a user group where usability is a primary concern (colour-blind prospective users).

SH8 - The system should provide a feedback message after attempting to complete the given problem (functional). This requirement has been satisfied as after any attempt at a Parson's problem the user is presented with the correct feedback to help the user improve their attempt. The system could be improved to better satisfy this requirement by offering a more reactive feedback system which changes the prompt even when the feedback conditions match the previous attempt. This particularly applies to the feedback stating 'the solution requires fewer/more lines' as it possible for the user get the same feedback after two attempts and may not be getting the best feedback possible (e.g. is the user submits 3 lines of code to a problem that requires 5 lines they will get a 'solution requires more lines' message; if their next attempt has 4 lines, they will get the same message, which wouldn't be as helpful as a feedback message saying 'solution requires *even* more lines' or simply 'solution requires 5 lines'.)

SH 9 - The system should be designed with an awareness of the implications of using distractors (non-functional). This requirement has been satisfied by using distractors more sparingly with increasingly difficult questions. As identified in the literature, improper use of distractors can have an adverse effect on learning, so they are primarily used in simpler problems. To further the progress on this requirement, thought could have been put into how to present problems in such a way that distractors could also be implemented using the valid line/distractor line pairings as presented in Fig. 2B.

7.3 Could Have Requirements

It is determined that only one of the Could Have requirements has been satisfied. What follows describes and justifies why most of the following requirements were not completed as well as describing the extent to which the satisfied requirement, CH4, is completed.

CH1 - The system could introduce the necessity for the user to arrange lines horizontally as well as vertically (functional). This requirement has not been satisfied. A similar effect has been achieved using tabs in the lines which should have indentation in order to expose the student to how best practice indentation should look. It is justified that this perhaps should be the least prioritised requirement, as indentation within Java makes no functional difference to the running of code, so it is understandable why this requirement was not completed. Should the current system be adapted to present problems using a different programming language, such as Python, where indentation is important, then completing this requirement would become more important.

CH2 - The system could make use of online user score storage, that can be accessed using Google Sign-In service (functional). This requirement has not been satisfied and as such, the system as it currently exists does not allow for the transfer of student profiles across devices. This requirement could have been fulfilled if time was managed in such a way that allowed for more fact finding on the process of how to develop a system that can save user information remotely, earlier in the development process. While this requirement would be useful to satisfy under the circumstance of having a large user base, the fact that this requirement was not developed does not impede on the system's ability to meet most other requirements within the context of one device per user.

CH3 - The system could further the Student Modelling procedure to consider multiplicity of skills which a problem from any one topic may develop (functional). This requirement has been satisfied. It expands on requirement SH4 by introducing a procedure for identifying whether any bonus skills can be attributed to the user based on a topic tagging mechanism. This fairly simple expansion on SH4 can allow for more accurate student modelling, because if a user is able to solve a problem that considers multiple skills - this requirement enables this to be reflected within the students scores, so that users can be served problems that are better able to appropriately challenge a their skills.

CH4 - The system could allow the user to load custom Parson's problems from a text file (functional). This requirement has not been satisfied, to do so would require the development of a subclass of the ProblemActivity class as well as a means for the user to write, save and load custom Parson's problems which could have been possible if time management was more effectively utilised. The author sees the potential for this requirement becoming more significant under the circumstance of the system being used as a teaching aid, where an educator intends to distribute to their students their own set of exercises in the form of Parson's problems.

8. Conclusion

This chapter concludes the project by discussing the contributions made to the study of informal computer programming education in Section 8.1, as well as discussing what could be done to further the work presented in the project in Section 8.2. Section 8.3 provides reflections more relevant to the project in-and-of-itself, considering what went well and what did not throughout the project as well as what was learned.

8.1 Summary of Contributions

Through conducting a literature review on the use of Parson's problems, a gap was found in the way in which Parson's problems are used. This project made progress towards closing this gap by developing a mobile system that offers Parson's problems in an informal education tool that also considers the implications of student modelling. As the literature directly informed the requirements generation for the project it can be said that the system is a synthesis of the latest academic understanding of the role Parson's problems can play in a beginner programmer's education. After the system was designed and implemented an evaluation of the system against the described requirements could be completed to analyse to what extent the implementation was a success.

The value this project possesses to any future work in the field stems from the development of a mobile Parson's system on a highly ubiquitous platform, such that this project could be easily taken and adapted by researchers to gain further knowledge and insight into the use of these problems as an educational tool, specifically within an informal setting that is not tied to any formal or summative assessment or even any particular institutional course.

8.2 Further Work

The immediate next step for the project would be to gain an insight from prospective users by conducting the observational study that was planned but not executed, to inform further refinements to the system.

The next-most pressing work to further the project would be to complete, or reevaluate the necessity of some of the ‘Could Have’ requirements. As only one out of the four ‘Could Have’ requirements were completed, it may be worthwhile to take the finishing of the project as an opportunity to reconsider each of these uncompleted requirements within the context of gaining prospective user feedback, before progressing work on them.

A final point of further work that could apply to this project or projects seeking to achieve similar goals would be to look into conducting a wider review of literature on informal learning in-and-of-itself, not just related to Parson’s problems or programming education. This would help in developing a wider understanding of more abstract principles of informal learning on mobile systems that could be leveraged to refine future mobile Parson’s problems systems.

8.3 Project Reflections

Through conducting this project, the author’s skills using the Java programming language have been advanced as well as developing new skills in applying this knowledge to the paradigm of mobile Android app development. For the most part, it can be said that time and expertise risks were able to be managed effectively as crucial components of the project were able to be completed. However, if greater discipline was employed such that more of the self-scheduled project work sessions were fully committed to, then perhaps more of the advanced features as described by the ‘Could Have’ requirements could have been completed. What worked particularly well regarding time management was the regular scheduling of meetings with the project supervisor which helped in gaining constant feedback on how to improve on the work that had been completed. While regular contact with the project supervisor and designing an observational study helped the author to consider the perspective of a prospective user, should the project be reapproached it would likely prove beneficial to more directly involve these prospective users to gain a greater variety of opinions that could be applied to the evaluation process.

9. References

Android Drag and Drop. (2013). Retrieved from a the Wordpress blog D'CODE website: <https://13mcec21.wordpress.com/2013/10/23/android-drag-and-drop/>

Bari, A.T.M., Gaspar, A., Wiegand, R., Albert, J., Bucci, A. & Kumar, A.. (2019). Evoparsons: design, implementation and preliminary evaluation of evolutionary parsons puzzle. genetic programming and evolvable machines. 10.1007/s10710-019-09343-7.

Denny, P., Luxton-Reilly, A., & Simon, B. (2008). Evaluating a new exam question: parsons problems. *ICER'08 - Proceedings of the ACM Workshop on International Computing Education Research*. 10.1145/1404520.1404532

Ericson, B., Margulieux, L.E., & Rick, J. (2017). Solving parsons problems versus fixing and writing code. *Koli Calling*.

Fabic, G. V. F., Mitrovic, A., & Neshatian, K. (2019). Evaluation of parsons problems with menu-based self-explanation prompts in a mobile python tutor. *International Journal of Artificial Intelligence in Education*. <https://doi.org/10.1007/s40593-019-00184-0>

Garcia, R., Falkner, K. & Vivian, R. (2018). Scaffolding the design process using parsons problems. 1-2. 10.1145/3279720.3279746.

Harms, K., Chen, J. & Kelleher, C. (2016). Distractors in parsons problems decrease learning efficiency for young novice programmers. 241-250. 10.1145/2960310.2960314.

Helminen, J., Alaoutinen, S., Ihantola, P., & Karavirta, V. (2013). How do students solve parsons programming problems? -- execution-based vs. line-based feedback. 55-61. 10.1109/LaTiCE.2013.26.

Ihantola, P., & Karavirta, V. (2011). Two-dimensional parson's puzzles: the concept, tools, and first observations. *Journal of Information Technology Education: Innovations in Practice*. 10. 1-14. 10.28945/1394.

Karavirta, V. & Helminen, J. & Ihantola, P. (2012). A mobile learning application for parsons problems with automatic feedback. *Proceedings - 12th Koli Calling International Conference on Computing Education Research, Koli Calling 2012*. 11-18.

Kernighan, B., & Ritchie, D. (1988). *The C programming language (2nd ed.)*. London: Prentice Hall International

Matthes, E. (2016). *Python crash course: a hands-on project based introduction to programming*. San Francisco: No Starch Press

Morrison, B. & Margulieux, L. & Guzdial, M.. (2016). Subgoals help students solve parsons problems. 42-47. 10.1145/2839509.2844617.

Parsons, D. & Haden, P. (2006). Parson's programming puzzles: A fun and effective learning tool for first programming courses. *Conferences in Research and Practice in Information Technology Series*, 52, 157–163.

Schildt, H. (2019). *Java: a beginner's guide (8th ed.)*. London: McGraw-Hill Education

10. Appendices

A - Project Initiation Document



School of Computing Final Year Project

Aaryan Jay Ragoo

PJE40

Project Initiation Document

A Mobile System Using Student Modeling with Parsons Problems

1. Basic Details

Student name:	Aaryan Jay Ragoo
Draft project title:	A Mobile System Using Student Modeling with Parsons Problems
Course:	Computer Science
Client organisation:	
Client contact name:	
Project supervisor:	Matthew Poole

2. Degree Suitability

This project is suitable to the Computer Science degree as it will require skills and knowledge of and around programming, problem solving and information system design. The focus of using Parsons Problems with student modelling is particularly conducive to a Computer Science course as it will require a reading of research papers within the computing field.

3. Outline of the Project Environment And Problem to be Solved

3.1 Prospective Client/User

The prospective user would be a person early in the development of their knowledge of programming, with beginner programming skills. The user would likely already be familiar with using mobile apps on android devices, and perhaps already be familiar with mobile apps used for education.

3.2 Problem to be Solved

Parsons Problems are an exercise for learning programming concepts which involve organising lines of code within a short program, so that the organised lines of code correctly

perform a prompted task. This problem type has been shown to be a promising tool in teaching beginner programming concepts and the syntactic and logical structures of a programming language.

A dedicated Parsons Problems app for learning introductory programming concepts in Java is still yet to exist on the Google Play Store or F-Droid repository, despite several apps existing develop an education in programming and programming languages. In developing an app which uses student modelling techniques, and with a central focus on solving Parsons Problems, progress can be made towards the deployment of a widely available app which could prove to be beneficial to a novice programmer's learning.

4. Project Aim and Objectives

4.1 Project Aims

The overall project aims to provide a mobile system by which a user can use Parsons Problems as an effective means for learning introductory programming techniques in Java. The novel approach of developing a student performance modelling system is intended to develop upon the already recognised pedagogical value of Parsons Problems in programming education.

4.2 Objectives in Meeting Aims

To meet this overall aim, several objectives need to be completed.

- A literature review must be undertaken.
- A requirements elicitation process must be utilised that involves primary and secondary research.
- These requirements must be used to drive the design and system modelling process.
- The objective of system implementation must be driven by the requirements specification and system design.
- To meet the aim of the system being mobile, further knowledge must be gained in the paradigm of Android mobile development using the Android Studio IDE.

5. Project Deliverables

5.1 Deliverable System Artifacts/Documents

As the project progresses, several artifacts and documents will be developed. In the earlier stages of development, a literature review should be produced to evaluate the currently understood knowledge around Parsons Problems, and how this knowledge can be applied to the development of a new system. After this, questions should be asked about the requirements of the system, so that requirements can be elicited, engineered and then documented in the form of a System Requirements Specification.

Later in the project, design documents will be produced that will model the system's architecture and the system's interactions. This will include a Software Architecture Models, Use Case Diagrams and Sequence Diagrams which will demonstrate how the Parsons Problems are delivered by the system and how the student modelling system functions. Also,

GUI mock-ups will need to be made as part of the design process to visually illustrate the way in which a user will interact with the system.

Documents developed later in the process will include testing reports to describe testing strategies and test case analysis. This will likely be written in tandem with the development of the code artifact, as developing test cases as the system is implemented will insure that functionality is only put into effect if it can pass the necessary tests.

The Final Report will effectively be an organisation of the documents produced throughout the development of the project (literature review, requirements, design, testing) as well as a discussion of methodology and a conclusion that reflects on the project's development. The final deliverable will be the project presentation and system demonstration where the system will be shown to work and the project's development will be discussed.

5.2 MoSCoW Prioritisation of Code Artifact Functionality

In the development of the code artifact, a prioritisation of the system's features and functionality must be considered. The MoSCoW prioritisation method is utilised to evaluate these features in the order of which they should be developed. Doing this is helpful in the development of a project plan (§ 12) as it can be directly imposed onto a Gantt Chart, such that the features of greater importance can be developed sooner, with more time dedicated to them, than other features.

5.2.1 Must Haves

- The system must present the user with a way to solve a Parsons Problem through dragging and dropping lines of code from a bank of lines to correctly answer a given question.
- The system must be able to verify whether or not the user has correctly answered the problem and respond accordingly once the user has submitted their answer.
- The system must feature a functioning GUI for the user to complete each problem.

5.2.2 Should Haves

- The system should store a wide selection of problems in a file which are of varying difficulty which are grouped by the types of skills the user wants to practise.
- The student modelling system should work so that if a user completes a problem successfully, the user's skill level, concerning the technique the Parsons problem is targeting, is incremented by '1' or decremented by '1' should the user incorrectly solve the problem.
- The system should present the user with a Parsons problem which matches their current skill level for a particular skill.
- The system should save a user's profile information to a local file that is accessed when returning to the app so that the difficulty of each problem continues from the user's last session.

5.2.3 Could Haves

- The system could have the functionality for some online student profile storage, such that the student's profile is accessible remotely and independent of the device.

- The system could implement the function for remote profile storage using the Google sign in system.
- The system could include a brief tutorial page for when key new concepts are introduced.
- The system could allow the user to create and share their own Parsons Problems by giving a valid piece of code as well as some invalid lines.
- The Parsons Problems could be implemented such that not only does the user need to drag and drop the lines of code up and down to put them in the correct order, but also encourage the user to drag the lines of code left and right to encourage standard indentation practices.

6. Project Constraints and Risks

One constraint on the project is: the currently understood academic knowledge on Parsons Problems. As this project does not seek to synthesise any new knowledge around the way in which Parsons Problems work to develop skills in programming, the project must rely on what is already understood by current academics to guide the ways in which the problems should be implemented so that the project is an effective educational tool. This can be understood greater once a literature review has been conducted.

A further constraint is my current level of expertise in skills relevant to the project. This is easily managed by evaluating the current skills as well as evaluating which skills can be learned throughout the process of development. While skills in using the Java programming language are possessed, better knowledge on how to use these skills in Android App development should be acquired, and can easily be acquired within the given timeframe.

The project's strict timeline is a further constraint on the development of the system. As the project should be finished by May 1, effective time management must be employed. This is discussed further in § 12.

As the project evolves, it is possible that new risks are introduced and recognised risks change in their severity and likelihood. Therefore a log of risks should be maintained throughout the projects development.
<https://docs.google.com/document/d/1sPLedIpW8k-9krqVoGZJaIzzDDZpq63s8mKo4gtz7rU/edit?usp=sharing>

7. Project Approach and Plan

7.1 Approach

As the project takes place over the course of seven months, certain engineering processes should be employed. As only one person is conducting the project over the course of a short period of time (relative to other digital education systems that may be developed and maintained over the course of several years), it can be seen how the Waterfall process model would be conducive to the development of this project. This becomes more obvious when it is seen how that the requirements engineering process supports the design process, which supports the implementation process, which supports the verification process.

7.1.1 Requirements Stage

A literature review and an evaluation of current software should be conducted to elicit the discrete user requirements for the project. After that, the user requirements should be specified and divided into functional and nonfunctional requirements, and it should be made clear that there is a trace back from the specified requirements, to the elicited user requirements, to the data which elicited the user requirements.

7.1.2 Design Stage

Once the requirements are formally specified, the necessary architectural patterns should become more apparent. With design models should be produced to evaluate the overall function of the system from different perspectives and contexts. The design documentation deliverables are discussed further in § 5.

As part of the GUI design, a wireframe model of the flow of the system's interface changes as the user interacts with the system. This should illustrate what happens when the user starts the app, selects each of the presented buttons, and what happens when the user either incorrectly or correctly solves a problem. This model should be developed in a highly granular, if not, exhaustive way, so that all of the system's functionality is accounted for, so that the wireframe provides an effective guide for implementation.

7.1.3 Implementation Stage

Implementation of the code artifact should follow naturally on from the design and requirement specification processes. The wireframe model should be used as a guide of how the GUI is implemented and the architectural models should be used to guide how certain system objects (user profiles, Parsons problems) should be implemented. Revision control methods using Git and GitHub should also be utilized to ensure that working versions of the system are always available, and to keep track of any implementation issues that arise.

7.1.4 Verification Stage

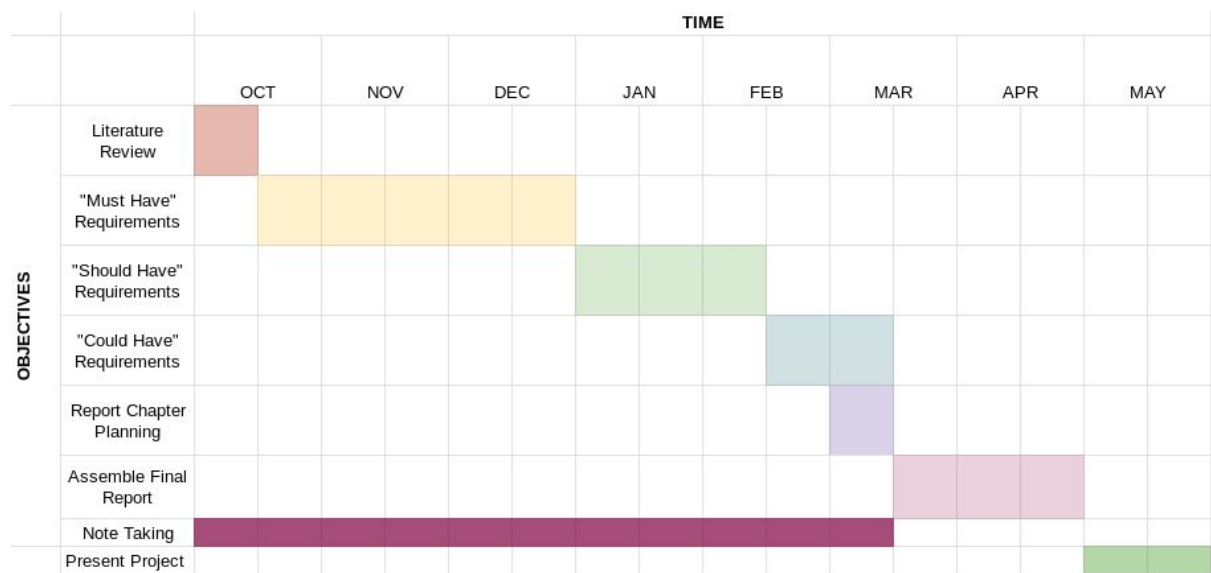
Through developing test cases and a testing plan of the methods and classes developed, the system can be verified to function as intentioned. Automated testing framework such as JUnit should be used in order to demonstrate the effective and correct function of methods quickly and frequently. Ideally, automated tests should be developed as their respective methods are implemented, however it is likely that some retrospective testing will be needed once the project is near completion in order to demonstrate the system working as a whole.

7.2 Plan

7.2.1 Gantt Chart

The Gantt Chart demonstrates how the project has been planned. Key stages of development are described on the right-hand-side of the chart, and they are broken up into sections by major project milestones (deliverable submission dates). The chart progresses through time, illustrating the amount of time each project task should require as well as considering which tasks can operate simultaneously and which tasks need to be completed before others can start. The Gantt chart is produced in two forms, one at a higher level, and one at a lower level. This helps to ensure that the project has reached an appropriate level of development at

any one time (easily seen in the simpler chart) as well as being able to see a granular break-down of the tasks that need to be completed at each stage.



https://docs.google.com/spreadsheets/d/1BiTnfbFOfapw8aQOI2g88yKN7eL_XQqKIWFWh67-Aw0/edit?usp=sharing

7.2.2 Planning for Risk

Despite planning the course of a project, the risk of missing self-set deadlines still exists. This has been mitigated by dividing the number of notational hours recommended for the project (400) by the number of days until the final submission date for the project presentation (1Oct to 1May = 213 days) to give the average number of hours per day to consistently work on the project. This ($400/213$) equates to an average of just less than 2 (1.878) hours per day that should be spent on the project until May 1. Knowing this makes it easy to schedule seven 2 hour periods throughout the week to ensure that the project is constantly being worked on. As well as this, regular (weekly/fortnightly) meetings with the project supervision will ensure that another person can verify that effective progress is being made towards the completion of the project.

8. Facilities and Resources

It is not anticipated that the project should require any more than some basic resources and facilities. Access to a personal computer with access to the internet covers all the necessary resources for the development of the project. Should it be considered necessary to gain feedback from potential users on the final code artifact, this could be done through borrowing some Android devices from the university. The university library website provides access to several relevant academic papers. No funding is required and it is not anticipated that there will be any constraints on the availability of any resource.

9. Starting Point for Research.

Several papers exist which evaluate the efficacy of Parsons Problems as a teaching device in learning and assessing programming skills. The conference paper [Parson's programming puzzles: A fun and effective learning tool for first programming courses](#) from 2006 and the research paper '[Evaluating a new exam question: Parsons problems](#)' from 2008 would appear to be some of the earliest mentions of Parsons problems within an academic context, and would provide a good point of entry into evaluating the current understanding of the function of Parsons problems in programming education. As well as this research should be undertaken into the currently available programming education apps to evaluate educational techniques used which may be useful in the development of this project.

10. Legal, Ethical, Professional, Social Issues

An awareness of the licences under which different libraries operate that are used by the software artifact should be considered in order to mitigate any legal concerns. Should data be collected about potential users when acquiring feedback on the system, special care must be taken to ensure that this data is collected and stored in a way in which no personal information about anyone is vulnerable. This can be ensured by anonymising and feedback received. An ethics review form has been completed as a part of the initiation of the project. It is unlikely that the project will encounter any significant professional or social issues.

B - Ethics Certificate



Certificate of Ethics Review

Project Title: A Mobile System Using Student Modeling with Parsons Problems

Name: Aaryan Jay Ragoo

User ID: 850844

Application Date: 09-Mar-2020 09:11

ER Number: ETHIC-2020-387

You must download your referral certificate, print a copy and keep it as a record of this review.

You should **submit your certificate to your FEC representative for further review.**

The FEC representative for the School of Computing is [Carl Adams](#)

It is your responsibility to follow the University Code of Practice on Ethical Standards and any Department/School or professional guidelines in the conduct of your study including relevant guidelines regarding health and safety of researchers including the following:

- [University Policy](#)
- [Safety on Geological Fieldwork](#)

All projects involving human participants need to offer sufficient information to potential participants to enable them to make a decision. Template participant information sheets are available from the:

- [University's Ethics Site \(Participant information template\).](#)

It is also your responsibility to follow University guidance on Data Protection Policy:

- [General guidance for all data protection issues](#)
- [University Data Protection Policy](#)

Which school/department do you belong to?: **SOC**

What is your primary role at the University?: **Undergraduate Student**

What is the name of the member of staff who is responsible for supervising your project?: **Matthew Poole**

Is the study likely to involve human subjects (observation) or participants?: **Yes**

Will peoples' involvement be limited to just responding to questionnaires or surveys, or providing structured feedback during software prototyping?: **Yes**

Confirm whether and explain how you will use participant information sheets and apply informed consent.: **An information sheet will be used to inform the potential participant the extent to which they are involved in the study as well as to describe the process of the study. The information sheet will include that the study is part of a final year project and takes place within a topic that they are familiar with. The sheet will also describe that I intend to give the participant a mobile phone with the demo of the project app installed to use as I note the student's interactions with and opinions of the app demo. Finally, the information sheet will describe that the opinions provided by the participant will be used to evaluate the design of the app demo. Informed consent will be applied by providing the potential participant with information sheet so that they are aware of what would be involved in participating in the study, and then only performing the observation if the participant signs the information sheet.**

Confirm whether and explain how you will maintain participant anonymity and confidentiality of data collected.: **It will be ensured that no participant can be identified from observational notes by making sure that the notes do not contain any information that could be used to compromise the anonymity of the participant, such as their name, age, address, or student identification number. As the participant's signature is collected on the information sheet as part of the informed consent process, it must be ensured that this information sheet with the set of signatures is stored in a secure location, either in the project supervisor's office or in my possession.**

Will the study involve National Health Service patients or staff?: **No**

Do human participants/subjects take part in studies without their knowledge/consent at the time, or will deception of any sort be involved? (e.g. covert observation of people, especially if in a non-public place): **No**

Will you collect or analyse personally identifiable information about anyone or monitor their communications or on-line activities without their explicit consent?: **No**

Does the study involve participants who are unable to give informed consent or in are in a dependent position (e.g. children, people with learning disabilities, unconscious patients, Portsmouth University students)?: **Yes**

How do you plan to minimize risks? e.g. explain whether you will require the co-operation of a "gatekeeper" for initial/continuing access to the groups or individuals to be recruited? (e.g. students at school, residents of nursing home, members of a tribe): **Risk has been minimized by accessing the participants through the gatekeeper of the project moderator who is the coordinator of a class of students of the University of Portsmouth who are the intended participants being observed as part of the study.**

Are drugs, placebos or other substances (e.g. food substances, vitamins) to be administered to the study participants?: **No**

Will blood or tissue samples be obtained from participants?: **No**

Is pain or more than mild discomfort likely to result from the study?: **No**

Could the study induce psychological stress or anxiety in participants or third parties?: **No**

Will the study involve prolonged or repetitive testing?: **No**

Will financial inducements (other than reasonable expenses and compensation for time) be offered to participants?: **No**

Are there risks of significant damage to physical and/or ecological environmental features?: **No**

Are there risks of significant damage to features of historical or cultural heritage (e.g. impacts of study techniques, taking of samples)?: **No**

Does the project involve animals in any way?: **No**

Could the research outputs potentially be harmful to third parties?: No
Could your research/artefact be adapted and be misused?: No
Does your project or project deliverable have any security implications?: No
Please read and confirm that you agree with the following statements: **Confirmed**
Please read and confirm that you agree with the following statements: **Confirmed**
Please read and confirm that you agree with the following statements: **Confirmed**

Supervisor Review

As supervisor, I will ensure that this work will be conducted in an ethical manner in line with the University Ethics Policy.

Supervisor signature:



Date:

9/3/20

Review by FEC Representative

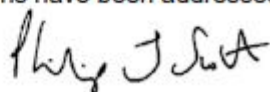
The supervisor must have reviewed and signed above before this section can be completed.

☒ Synopsis and supporting information has been provided.

Name of representative: Dr Philip Scott

Comments: All concerns have been addressed.

Representative signature:



Date: 09/03/20

C - Observational Study Information Sheet

Study Title: An observational study on the use of a mobile Parson's problems app by first year programming students, with the aim to gain insight on the app for use in evaluation.

Department: School of Computing

Researcher: Aaryan Jay Ragoo (email: up850844@myport.ac.uk)

Supervisor: Dr Matthew Poole (email: matthew.poole@port.ac.uk)

1. Invitation: Aaryan Jay Ragoo, a final year student at the University of Portsmouth, is inviting you to participate in a study as part of a final year project. The project supervisor is Dr. Matthew Poole. The following information sheet should take around 4 minutes to read and should answer any questions you may have on the study but feel free to ask any questions if anything is not clear.

2. Summary: This study is concerned with gaining insight on an educational mobile app developed using Parson's Problems, a form of code re-arrangement exercise. This is important due to research that has demonstrated their efficacy in learning to program. The participant should be a first year programming student and would require you to attend your regular programming class. The study should not require more than 5 minutes of your time, after the information sheet has been read.

3. Purpose: The purpose of this study is to gain insight on the opinions of beginner/first-year university student programmers as they interact with a programming education app using Parson's Problems developed as part of a final year project. You will be

provided with a mobile smartphone with a demo of the app installed and asked to complete a series of tasks using the app. The researcher will be taking notes on your reaction to using the app as well as notes on opinions you express. The study is included as part of a project that contributes towards a degree classification.

4. Why have you been invited?: You have been selected as you are part of the first year programming class at the school of computing and are analogous to the target audience for the demo app.

5. Do I have to take part?: No, participating is completely voluntary. If you agree to take part then you will be required to sign the consent form at the end of the information sheet.

6. What will happen to me if I take part?: You will be observed using a mobile app for about 5 minutes. During this time the researcher will observe and take notes on how you interact with the app and ask questions to gain your opinions on different aspects of the app. The main function of the app is to deliver to the user a series of code rearrangement exercises (known as Parson's problems), which you will attempt to complete as part of the study. The notes taken by the researcher will be published as part of the final report for the project, but no personal or identifying information will be published in any form.

7. Expenses and Payments: No expenses or payments are offered as part of the study.

8. Anything else I will have to do?: No, nothing is required of you before or after the study.

9. What data will be collected and / or measurements taken?: No data is collected other than notes made by the researcher that describe the opinions you mention or the way in which you interact with the demo.

10. What are the possible disadvantages, burdens and risks of taking part?: The study presents no heightened risk to health and safety. If you are medically adverse to using a smartphone for periods of around 5 minutes, the researcher should be notified.

11. What are the possible advantages or benefits of taking part?: You will be contributing to the evaluation of an app intended for others within a similar position as you, to help improve the education of future beginner programmers. No direct individual benefits are offered.

12. Will my data be kept confidential?: No raw data about you will be collected or retained. The only data collected will be the notes taken by the researcher. As such, special care will be taken to ensure that these notes are anonymous and do not contain any personal or identifying information. The notes taken by the researcher will be taken on paper then digitized by being typed into a Google Doc document accessible by both the researcher and the project supervisor. The notes taken by the researcher will be published as part of a final report for the project which will be publically shared. The signature you provide as part of the informed consent process will be stored securely by the project supervisor.

13. What will happen if I don't want to carry on with the study?: At any point in time during the study you can stop participating and withdraw from the study. You can withdraw any opinion you have provided at any time before 01/05/2020 by contacting the researcher. Once the study has been completed and the notes generated during observation have been published, it will not be possible for you to withdraw your data from the study.

14. What if there is a problem?: If you have a query, concern or complaint about any aspect of this study, in the first instance you should contact the researcher(s) if appropriate. If the researcher is a student, there will also be an academic member of staff listed as the supervisor whom you can contact. If there is a complaint and there is a supervisor listed, please contact the Supervisor with details of the complaint. The contact details for both the researcher and any supervisor are detailed on page 1. If your concern or complaint is not resolved by the researcher or their supervisor, you should contact the Head of Department:

The Head of Department
School of Computing
University of Portsmouth
Buckingham Building
Lion Terrace
Portsmouth
PO1 3HE

Dr Nick Savage
023 9284 2554
Nick.Savage@port.ac.uk

If the complaint remains unresolved, please contact:

The University Complaints Officer
023 9284 3642 complaintsadvise@port.ac.uk

15. Who is funding the research?: The researcher or supervisor are not receiving any specific financial reward for the completion of this study

16. Who has reviewed the study?: To ensure your dignity and wellbeing, this study has been reviewed by the School of Computing Faculty Ethics Committee representative: Philip Scott and been given favourable ethical opinion.

Thank you for taking time to read this information sheet and for considering volunteering for this observational study.

Consent Form:

By signing the information sheet, you are consenting to being observed using the demo app as well as your opinions provided being noted in written form and used in an evaluative discussion on the app as part of the project report.

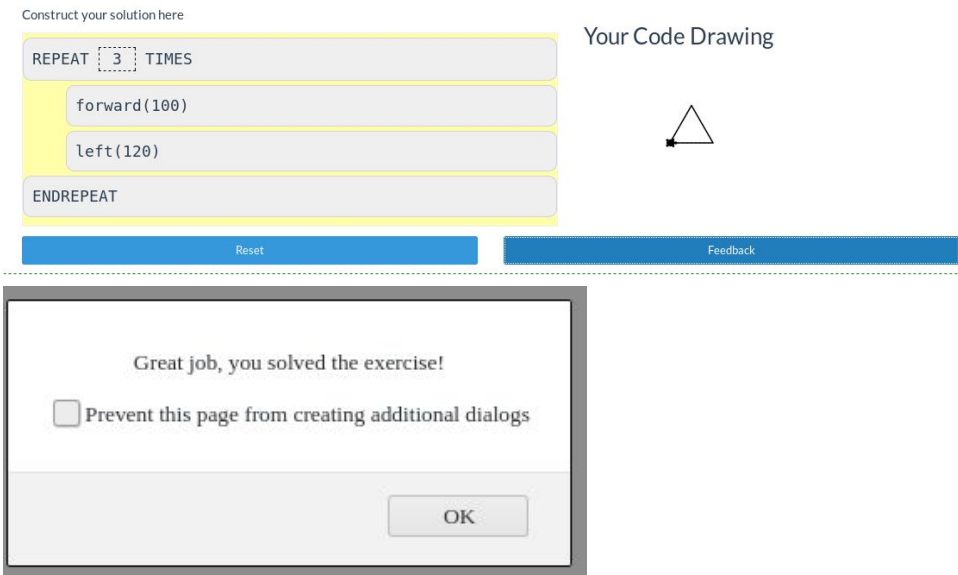
Participant Number	Signature	Date
1		
2		
3		
4		
5		
6		
7		
8		
9		
10		
11		
12		
13		
14		
15		
16		
17		
18		
19		
20		

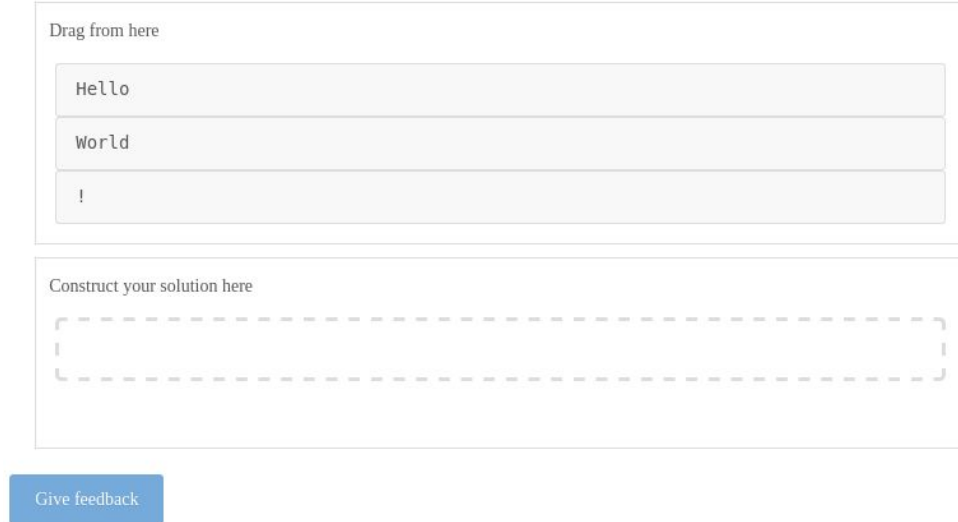
D - Brief Extant Software Survey

Name	Runestone Academy
Source	https://runestone.academy/runestone/static/instructorguide/Directives/mixedUp.html

	https://runestone.academy/runestone/books/published/apcsareview/ArrayBasics/ArrayParsonsPractice.html
Screenshot	<p>7-22-1: The following program segment should double each element in the array then print out the new value -- so (1,2,3,4,5) should become (2,4,6,8,10). But, the blocks have been mixed up. Drag the blocks from the left and put them in the correct order on the right. Click the <i>Check Me</i> button to check your solution.</p> <div style="display: flex; justify-content: space-around; align-items: flex-start;"> <div style="text-align: center;"> <p><i>Drag from here</i></p> <div style="border: 1px solid #ccc; padding: 5px; margin-bottom: 5px;">for (int i = 0; i < arr.length; i++) {</div> <div style="border: 1px solid #ccc; padding: 5px; margin-bottom: 5px;">System.out.println(arr[i]);</div> <div style="border: 1px solid #ccc; padding: 5px; margin-bottom: 5px;">arr[i] = arr[i] * 2;</div> <div style="border: 1px solid #ccc; padding: 5px; margin-bottom: 5px;">int[] arr = {1, 2, 3, 4, 5};</div> <div style="border: 1px solid #ccc; padding: 5px;">}</div> </div> <div style="text-align: center;"> <p><i>Drop blocks here</i></p> <div style="border: 1px solid #ccc; height: 100px; width: 100%; background-color: #ffffcc;"></div> </div> </div> <div style="display: flex; justify-content: center; margin-top: 10px;"> Check Me Reset Help Me </div> <p>7-22-1: The following program segment should double each element in the array then print out the new value -- so (1,2,3,4,5) should become (2,4,6,8,10). But, the blocks have been mixed up. Drag the blocks from the left and put them in the correct order on the right. Click the <i>Check Me</i> button to check your solution.</p> <div style="display: flex; justify-content: space-around; align-items: flex-start;"> <div style="text-align: center;"> <p><i>Drag from here</i></p> <div style="border: 1px solid #ccc; height: 100px; width: 100%; background-color: #d3d3ff;"></div> </div> <div style="text-align: center;"> <p><i>Drop blocks here</i></p> <div style="border: 1px solid #ccc; padding: 5px; margin-bottom: 5px;">int[] arr = {1, 2, 3, 4, 5};</div> <div style="border: 1px solid #ccc; padding: 5px; margin-bottom: 5px;">for (int i = 0; i < arr.length; i++) {</div> <div style="border: 1px solid #ccc; padding: 5px; margin-bottom: 5px;">arr[i] = arr[i] * 2;</div> <div style="border: 1px solid #ccc; padding: 5px; margin-bottom: 5px;">System.out.println(arr[i]);</div> <div style="border: 1px solid #ccc; padding: 5px;">}</div> </div> </div> <div style="display: flex; justify-content: center; margin-top: 10px;"> Check Me Reset Help Me </div> <div style="border: 1px solid #ccc; padding: 5px; text-align: center; margin-top: 10px; background-color: #e0f0ff;"> Perfect! It took you only one try to solve this. Great job! </div>
Notes	Runestone is an interactive textbook which integrates Parsons Problems as a ‘quizzing’ medium as a part of the education which they deliver. The software was accessed using a web browser.

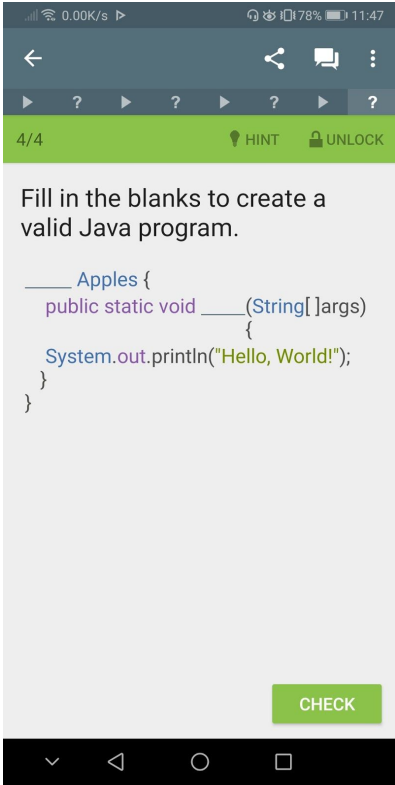
Name	js-parsons
Source	https://js-parsons.github.io/ https://github.com/js-parsons/js-parsons

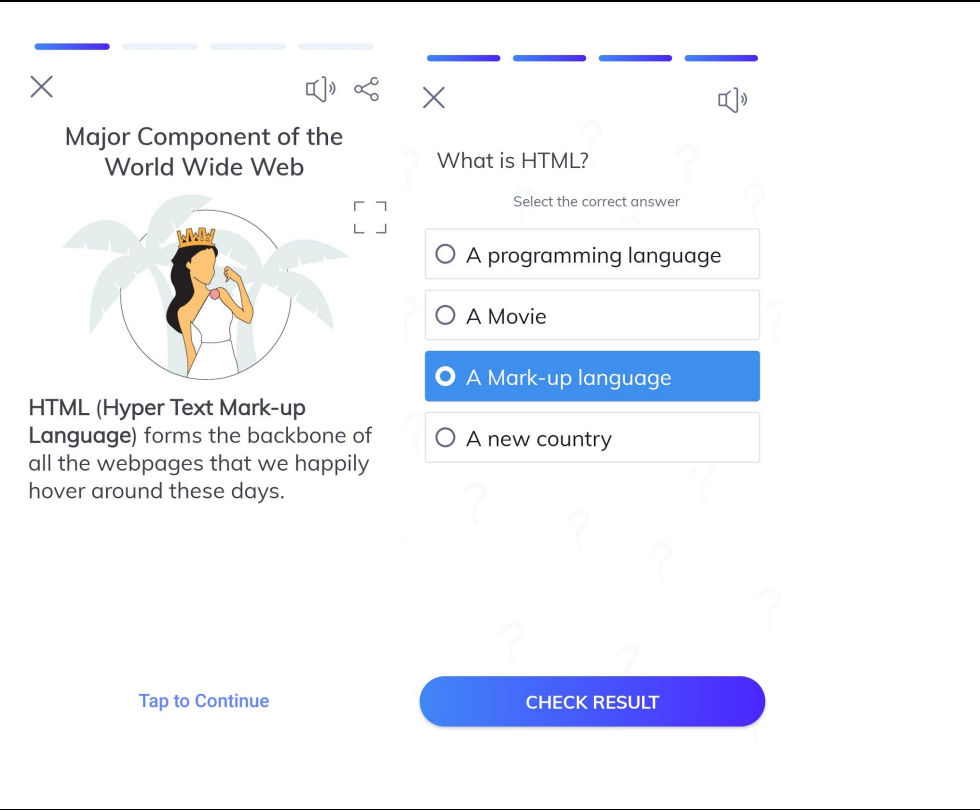
Screenshot	
Notes	<p>This system uses 2D or 1D Parsons Problems, as it also can consider whether the indentation is correct. This could be an idea which is explored, however, as the newly developed system will be using Java as the testing language, this could be an optional/lower priority consideration as indentation has no effect on code compilation in Java. Software was accessed using a web browser. Feedback given using browser dialogs.</p>

Name	rstudio/parsons
Source	https://github.com/rstudio/parsons https://rstudio.github.io/parsons/tutorials/tutorial_parsons.html
Screenshot	


	<p>Here is an example. Drag the statements from the top panel to the bottom panel to produce "Hello World!".</p> <div> <div>Drag from here</div> <div>Construct your solution here</div> <div> <div>Hello %>%</div> <div>World %>%</div> <div>!</div> </div> <div>Correct!</div> <div>Next Topic</div> </div>
Notes	This system allows the user to develop their own Parson Problems for the R programming language. Software accessed using a web browser.

Name	SoloLearn
Source	https://play.google.com/store/apps/details?id=com.sololearn
Screenshot	

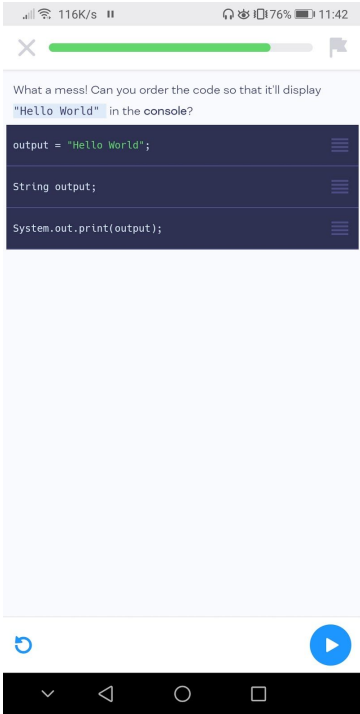
	
Notes	<p>Presents a course based around teaching various programming languages and other information technology skills. Integrates textbook-like presentation of information with different types of quiz questions. No use of Parson's problems. Also includes community forum as well as the ability to run user written code.</p>
Name	Programming Hub
Source	https://play.google.com/store/apps/details?id=com.freeit.java


Screenshot	
Notes	Similar in presentation to SoloLearn regarding the education process, no community forum.

Name	Programming Hero
Source	https://play.google.com/store/apps/details?id=com.learnprogramming.codecamp

Screenshot	<div style="display: flex; justify-content: space-around;"> <div style="width: 48%;"> <p>Output 0</p> <p>1/6</p> <p>Show Result</p> <p>When you use a software/website/ app, you expect it to do something and show you output. To display output, you need to use a special keyword <code>print</code>.</p>  <pre>1 print("Emma Watson")</pre> <p>Try it</p> <p>Next</p> </div> <div style="width: 48%;"> <p>Output 0</p> <p>3/6</p> <p>Quiz 10s</p> <p>What do you need to use to display output ?</p> <p><input checked="" type="radio"/> <code>print()</code></p> <p><input type="radio"/> <code>printing()</code></p> <p><input type="radio"/> <code>print it now()</code></p> <p>Next</p> </div> </div>
Notes	<p>Takes an approach likely geared towards younger audiences. Presents information describing programming concept > presents timed quiz on programming concept.</p>

Name	Mimo
Source	https://play.google.com/store/apps/details?id=com.getmimo

Screenshot	
Notes	<p>Prefaces any education content by assessing current knowledge (by user description of skill) and user intention (learning a language, building projects, advancing career).</p> <p>Presents course material in a way similar to other apps: fill in the gaps of a code snippet from a bank of possible answers after reading about a certain concept.</p> <p>Another question type: select which statements are true.</p> <p>Features achievements for completing certain tasks.</p> <p>Features daily challenges.</p> <p>Uses Parsons Problems as a question type.</p>
Name	Codemurai
Source	https://play.google.com/store/apps/details?id=com.zenva.codemurai


Screenshot	 <p>Can you drag the following blocks of code and place them in the correct order to do the following.</p> <ol style="list-style-type: none"> 1. Declare a variable called <code>canJump</code>. 2. Give it a value of <code>true</code>. 3. Give it a value of <code>false</code>. <pre>canJump = false;</pre> <pre>boolean canJump;</pre> <pre>canJump = true;</pre> <p>♥ ♥ ♥ ♥</p>
Notes	<p>As with other programming education apps, it follows the process of presenting a screen of information, followed by a quiz with the main question type being: to select answer/s from a multiple choice group for a question. Also includes a Parsons Problem implementation, but is not the main focus.</p>

E - Testing Table

Test ID	Class	Method Name	Input	Expected Output	Case Type/Passing?/Other Notes
01	MainActivity	getVariantsMatrix()	none	{{3,3,3,3,3,3,3,3,3,3},{1,0,0,0,0,0,0,0,0,0},{1,0,0,0,0,0,0,0,0,0},{1,0,0,0,0,0,0,0,0,0},{1,0,0,0,0,0,0,0,0,0},{1,0,0,0,0,0,0,0,0,0},{1,0,0,0,0,0,0,0,0,0},{1,0,0,0,0,0,0,0,0,0}}	No inputs for the method, so only one test case. Passing.
02	ParsonsProblem	ParsonProblem()	"[prompt]\nTesting Prompt\n[valid lines]\nline1\nline2"	ParsonsProblem object with correct attributes	Expected case. Passing.

			\nline3\n[distractor s]\ndist1\ndist2\n[e nd]"		
03	ParsonsP roblem	ParsonProble m()	“”	ParsonsProblem object with null attributes.	Anomalous case, empty input string. Passing.
04	ParsonsP roblem	ParsonProble m()	“[prompt]\n[valid lines]\n[distractors] \n[end]”	ParsonsProblem object with empty attributes.	Anomalous case, valid input string with missing attribute data. Passing.
05	Student	Student()	“1,2,3,4,5,6”	Student object with correct attributes.	Expected case. Passing.
06	Student	Student()	“0,0,0,-1,-2,-3”	Student object with skill attributes set to 1.	Anomalous case, skill levels lower than allowed. Passing.
07	Student	Student()	“11,22,33,44,55,66 ”	Student object with skill attributes set to 10.	Anomalous case, skill levels higher than allowed. Passing.
08	Student	Student()	“-1,-2,0,11,22,66”	Student object with skill attributes set to 10 if greater than 10 in input string and 1 if lower than 1 in input.	Anomalous case, skill levels higher and lower than allowed. Passing.
09	Student	Student()	“-1,0,1,5,10,11”	Student object with skill attributes set to 10 if greater than 10 in input string and 1 if lower than 1 in input and same value within allowed range.	Anomalous case, skill levels higher and lower than allowed but with also allowed values. Passing.
10	Student	Student()	“”	Student object with skill attributes set to 1.	Anomalous case, empty skill string. Passing.

11	Student	Student()	"1,2,3,4,5"	Student object with skill attributes set to 1.	Anomalous case, skill string with not enough values. Passing.
12	Student	Student()	",,,,,"	Student object with skill attributes set to 1.	Anomalous case, 6 comma separated values containing no integers. Passing.

 Tests passed: 12 of 12 tests - 72ms		/opt/android-studio/jre/bin/java ...	
▼	✓ GeneralTesting (com.example.javaparsonproblems)	72 ms	
✓	getVarsTest	63 ms	
✓	parsonsProblemObjectAnomalous	1 ms	
✓	parsonsProblemObjectAnomalous2	7 ms	
✓	parsonsProblemObjectTestExpected	0 ms	
✓	studentObjectTestAnomalous	0 ms	
✓	studentObjectTestAnomalous2	1 ms	
✓	studentObjectTestAnomalous3	0 ms	
✓	studentObjectTestAnomalous4	0 ms	
✓	studentObjectTestAnomalous5	0 ms	
✓	studentObjectTestAnomalous6	0 ms	
✓	studentObjectTestAnomalous7	0 ms	
✓	studentObjectTestExpected	0 ms	

All unit tests passing as of 2020-05-03