Solheim, Sindre
Næss, Henrik Rudi

Cooperative Parsons problems to promote learning and socialization in higher education

Master's thesis in Computer Science
Supervisor: Sindre, Guttorm
June 2022

Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Computer Science

NTNU
Norwegian University of
Science and Technology

Solheim, Sindre
Næss, Henrik Rudi

Cooperative Parsons problems to promote learning and socialization in higher education

Master’s thesis in Computer Science
Supervisor: Sindre, Guttorm
June 2022

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Computer Science

NTNU
Norwegian University of
Science and Technology

Sammendrag

Frafall og strykprosent har vært et problem i introduksjonsfag for programmering. Studier antyder at noen faktorer som bidrar til frafall er mangel på motivasjon og sosialt nettverk. Studier hevder også at programmering er vanskelig å lære. Dessuten er ensomhet et utbredt problem blant studenter i høyere utdanning, noe som kan ha implikasjoner for akademisk studie og studentenes velvære. Studier har antydet at samarbeidslæring kan være nyttig for å øke prestasjoner og sosiale relasjoner. Studier har også antydet at spillifisering kan fremme motivasjon og læring. Videre så har en kodefullføringsoppgave kalt *Parsons problems* blitt anbefalt ettersom den er hevdet til å være mer tidsbesparende og like effektiv som tradisjonelle programmeringsoppgaver. Parsons problems har hovedsakelig vært en individuell aktivitet så langt. Forskningsmålet, for dette studiet, er å undersøke om samarbeids Parsons problems kan kombinere de pedagogiske fordelene av Parsons problems med de sosiale fordelene av samarbeid; for å gi studenter en effektiv, morsom og sosial måte å lære programmering.

Vi undersøkte eksisterende Parsons problems programvareløsninger, men fant ingen samarbeidsløsninger. Dermed bestemte vi oss for å utvikle et digitalt samarbeids Parsons problems spill. Utviklingen fulgte en iterativ prosess hvor vi gjennomførte brukertester for å samle tilbakemeldinger på prototypen vår og dens mulige fordeler. Våre resultater antyder at studentene oppfattet samarbeids Parsons problems som en lærerik, morsom og sosial aktivitet. Implikasjonene av denne studien er begrenset, ettersom den ikke tester faktisk læringsutbytte. Fremtidig arbeid bør gjøres for å teste læringsutbyttet av et samarbeids Parsons problems spill, for studenter som er nybegynnere i programmering; ved å bruke vår prototype eller lignende løsninger.

Abstract

Dropout and failure rates have been an issue for introductory programming courses. Studies on course dropout rates suggest that some of the factors that contribute to dropout are lack of motivation and social networks. Studies also claim that programming is a difficult subject to learn. Additionally, loneliness is a prevalent issue among students in higher education, which can have implications for academic study and students' overall well-being. Cooperative learning has been shown to increase achievement, as well as social relatedness. Additionally, studies suggest that gamification can promote motivation and learning. Furthermore, a code completion exercise called *Parsons problems* has been recommended as it is claimed to be more efficient and just as effective as traditional programming exercises. However, Parsons problems have mostly been an individual activity thus far. The research objective for this study is to investigate whether cooperative Parsons problems could combine the pedagogical advantages of Parsons problems with the social benefits of cooperation; to provide students with an effective, fun, and social activity to learn programming.

We reviewed existing Parsons problems software solutions. However, we could not find any cooperative solutions. Therefore, we decided to develop a digital cooperative Parsons problems game. The development followed an iterative process where we performed user tests to gather feedback on our prototype and how students perceived its possible benefits. Our results suggest that the students perceived cooperative Parsons problems as an educational, fun, and social activity. A limitation of this study is that we did not test the actual learning outcome. Further work should be done to test the learning outcome of a cooperative Parsons problems game for novice programming students; by using our prototype or similar solutions.

Preface

This master thesis was written by Henrik Rudi Næss and Sindre Solheim; as a part of the course TDT4900 at NTNU. We want to give a special thanks to all of our user testers who made this thesis possible. When we started this project, we worried that finding user testers would be difficult. However, students were positive and showed interest in our game. We want to thank our supervisor, Guttorm Sindre, for the guidance throughout the project. Thanks for the pleasant meetings and your thorough feedback on our thesis. We also want to thank Ragnhild Fjærbu Solheim, Sindre's wife, for her help with the game design. Ragnhild provided insightful feedback from her background as a game artist. The submission of this thesis also means that we say goodbye to our university, NTNU. Thanks for five great years, both at campus Gjøvik and campus Gløshaugen in Trondheim. Thanks to all of our professors and fellow students who made our student years memorable.

Terms

- **Fun:** the feeling of being amused or entertained [1]
- **Learning:** the alteration of behavior as a result of individual experience [2].
- **Parsons problem:** a programming exercise where students rearrange a set of code blocks into a problem solution[3, 4].
- **Program:** the piece of code the teams are trying to assemble during a game of TPP.
- **RQ:** acronym for research question.
- **Socialization:** the process whereby an individual learns to adjust to a group [5].
- **TA:** acronym for thematic analysis.
- **TPP:** acronym for Team-based Parsons Problems; the game that is developed in this project. TPP is also referred to as our cooperative Parsons problems solution.
- **Task-creation tool:** a separate piece of software that we developed to create tasks and task-sets to be utilized in TPP.
- **Team:** a group that plays TPP with a shared goal of solving a set of programming exercises.
- **The project:** the work that went into the development of TPP and this thesis.

Table of Contents

1 Introduction **1**

1.1 Background and motivation . . . 1

1.2 Problem statement . . . 2

1.3 Research questions . . . 3

1.4 Project goals . . . 3

1.4.1 Result goals . . . 3

1.4.2 Effect goals . . . 3

1.5 Project scope . . . 4

1.6 Target group . . . 4

1.7 Reader guide . . . 4

2 Literature review **5**

2.1 Different types of programming exercises . . . 5

2.2 Parsons problems . . . 6

2.2.1 Distractors . . . 6

2.2.2 Pre-scaffolded and student-scaffolded . . . 6

2.2.3 Line-based and execution-based feedback . . . 6

2.2.4 Part-complete problems . . . 6

2.3 Existing Parsons problems software solutions . . . 7

2.3.1 Js-parsons . . . 7

2.3.2 Parsons Problem Webapp . . . 7

2.4 Learning and cooperation . . . 8

2.4.1 Cognitive load . . . 9

2.4.2 Transactivity . . . 9

2.4.3 Cooperative gamification . . . 9

2.5 Design thinking . . . 9

2.6 Design science . . . 10

2.7 Universal Design . . . 10

2.8 Peer-to-peer . . . 11

2.8.1 Technology . . . 11

2.8.2 Peer-to-peer in online games . . . 11

2.9 System Usability Scale . . . 12

2.10 Thematic analysis . . . 13

2.11 Game theory and game design . . . 14

2.11.1 The four fun keys 14
2.11.2 Buchinger and Hounsell's 19 design guidelines 15
2.12 Zhang's ten design principles 17

3 Methodology **18**
3.1 First iteration 18
3.1.1 Requirement specification 18
3.1.2 Design 19
3.1.3 Development 20
3.1.4 User testing 20
3.2 Second iteration 21
3.2.1 Requirement specification 21
3.2.2 Design 21
3.2.3 Development 23
3.2.4 User testing 23
3.3 Third iteration 24
3.3.1 Requirement specification 24
3.3.2 Design 25
3.3.3 Development 29
3.3.4 User testing 30
3.4 Fourth iteration 32
3.4.1 Requirement specification 32
3.4.2 Design 32
3.4.3 Development 35
3.4.4 User testing 35
3.5 Task-creation tool 36
3.6 Thematic analysis 39

4 Results **41**
4.1 RQ1: Can cooperative Parsons problems be a fun and social activity for novice programming students? 41
4.1.1 Post-questionnaire polls 41
4.1.2 TA1: How do you think that this game could impact socialization among students? . . 42
4.2 RQ2: What are the players' perceptions of the potential learning outcome from solving cooperative Parsons problems? 43
4.2.1 Post-questionnaire polls 43
4.2.2 TA2 : What are your perceptions on potential learning outcomes from this game? . . 45

4.3 RQ3: How does a team work together to solve cooperative Parsons problems in a digital space? 46

4.4 RQ4: How could a cooperative multi-player Parsons problems game be designed? 47

4.4.1 User interface and features 48

4.4.2 Architecture 55

4.4.3 Performance 57

4.5 SUS score 58

4.6 Other findings 59

5 Discussion **61**

5.1 Evaluation of findings 61

5.1.1 RQ1: Can cooperative Parsons problems be a fun and social activity for novice programming students? 61

5.1.2 RQ2: What are the players' perceptions of the potential learning outcome from solving cooperative Parsons problems? 62

5.1.3 RQ3: How does a team work together to solve cooperative Parsons problems in a digital space? 63

5.1.4 RQ4: How could a cooperative multi-player Parsons problems game be designed? . 64

5.2 Evaluation of methodology 66

5.2.1 Evaluation of the iterative process 66

5.2.2 Evaluation of the development process 67

5.2.3 Evaluation of the user tests 68

5.2.4 Evaluation of the thematic analysis 70

5.3 Evaluation of TPP 71

5.3.1 Evaluation of functionality 71

5.3.2 Evaluation of architecture 73

5.3.3 Evaluation of performance 74

5.3.4 Choice of technology 75

5.4 Evaluation of project goals 75

5.5 Limitations 75

5.5.1 User tests on the wrong target group 75

5.5.2 Risk of misinterpreted data 76

5.5.3 Risk of misinterpreted questionnaires 76

5.5.4 Risk of dishonest feedback 76

5.5.5 Poor basis for comparison between cooperative and non-cooperative Parsons problems 76

5.5.6 Unorganized literature search 76

5.6 Further work 77

6 Conclusion **79**

Appendix **86**

A Requirement specification 86

A.1 Functional requirements 86

A.2 Non-functional requirements 89

A.3 Requirements priority 89

B User tests 90

B.1 Iteration 1 90

B.2 Iteration 2 91

B.3 Iteration 3 92

B.4 Iteration 4 96

C Prototypes 102

C.1 Initial sketches 102

C.2 Prototype 1 102

C.3 Prototype 2 102

C.4 Prototype 3 102

C.5 Prototype 4 103

D User test tasks 104

D.1 User test 1 and 2 104

D.2 User test 3 104

D.3 User test 4 105

List of Figures

1 A Js-parsons example task. 7
2 Feedback in Js-parsons. 7
3 Parsons Problems Webapp editor for writing Parsons problems. 8
4 A generated task with Parsons Problems Webapp. 8
5 Client-server architecture (left) and peer-to-peer architecture (right). 11
6 Formula for calculating the System Usability Scale score. 13
7 The four fun key Player Experience spirals 15
8 The four phases of our iterative process. 18
9 Timeline of the iterative process. 19
10 Design A: a simple and streamlined design. 20
11 Design B: a board game design. 20
12 Design created in Figma in iteration two. 22
13 Mockup of the user interface in iteration three 25
14 The implemented interface in iteration three 26
15 Adaptive interface layout based on screen size. 26
16 The React component hierarchy model in iteration three. 27
17 Initial sketch of the React components 30
18 Sequence diagram of the communication with the player-as-host design 33
19 Sequence diagram of the communication with previews. 33
20 Sequence diagram of the communication between a host to a non-host. 34
21 A player indicator example. 34
22 The game interface when another player locks in. 34
23 The game interface after a player is *locked in*. 35
24 Task format example. 37
25 The create-task interface. 38
26 Distractor indication in the task-creation tool. 38
27 Interface for selecting tasks to be combined into a task set. 39
28 Bar chart of the responses to Q1 41
29 Bar chart of the responses to Q2 42
30 Bar chart of the responses to Q3 42
31 Bar chart of the responses to Q4 42
32 Bar chart of the responses to Q5 44
33 Bar chart of the responses to Q6 44

34 Bar chart of the responses to Q7 . . . 44
35 Bar chart of the responses to Q8 . . . 45
36 Bar chart of the responses to Q9 . . . 45
37 The lobby of TPP. . . . 48
38 The game's user interface. . . . 49
39 The player fields from the blue player's perspective. . . . 49
40 The solutionfield featuring a potential solution. . . . 50
41 Drag-and-drop of a code block. . . . 50
42 Visual feedback when hovering code blocks. . . . 51
43 The topbar features the task description. . . . 51
44 The sidebar. . . . 51
45 The hint modal window. . . . 52
46 The clear modal window. . . . 52
47 The interface when two players have locked in. . . . 53
48 Line-based feedback for a submitted solution. . . . 54
49 Player indicator for a code block interaction. . . . 54
50 The screen presented after a finished task set. . . . 55
51 The final version of the React component tree. . . . 56
52 Flow chart illustrating how communication is implemented in TPP. . . . 57
53 Bar chart of the responses to Q10 . . . 59
54 Bar chart of the responses to Q11 . . . 59
55 Bar chart of the responses to Q12 . . . 59
56 Bar chart of the responses to Q13 . . . 60
57 Initial sketches for TPP . . . 107
58 Figma prototype, second iteration of TPP . . . 111
59 React prototype, third iteration of TPP . . . 115
60 TPP's lobby in iteration 4 . . . 116

List of Tables

1 Grades and adjective ratings of System Usability Scale scores. 13
2 The six phases of thematic analysis. 14
3 Buchinger and Hounsell's 19 guidelines for designing games. 16
4 Zhang's ten design principles for motivational affordances. 17
5 Communication protocol in iteration three. 28
6 JSON file format for a Parsons problem task. 37
7 A code block's class properties. 38
8 Characteristics for how teams work together. 47
9 Redux store states. 57
10 Communication protocol in iteration four. 58
11 Performance benchmarks for moving code blocks. 58
12 System Usability Scale scores from final user test. 58
13 Complete- and incomplete requirements. 71
14 Requirement prioritization table. 90

1 Introduction

Dropout has been an issue for introductory programming courses [6, 8, 7]. For example, the Helsinki University of Technology reported dropout rates of 30-50% for their introductory programming course [6]. Studies on course dropout rates suggest that some of the factors that contribute to dropout include lack of motivation, lack of social network, lack of time, lagging behind, and ineffective study techniques [6, 7]. Dropout can also be measured at a study program level. Heunlein estimates that 30% of undergraduate programming students terminate their studies; at German higher education [9]. However, dropout is not the only measurement of struggling students in introductory programming courses. Watson and Lie claimed that the worldwide failure rates for programming courses were 30% [10]. Bennedsen and E. Caspersen claim that failure rates are "not alarmingly high" in introductory programming and warn that a rumor of high failure rates can lead to a declining number of computer science students [11]. However, passing grades in introductory programming courses might not be satisfactory since grades in introductory programming courses often predict grades in later programming courses [12]. The following chapter shall go deeper into the background of these issues and lay out the motivation for our project.

1.1 Background and motivation

This thesis can be seen as a continuation of our student report from December 2021 [13]. There is no need to read the student report beforehand; since we included relevant content from our student report in this thesis.

Struggling programming students and Parsons problems

Studies claim that programming is a difficult subject to learn [14, 15, 16, 17] . While there are many contributing factors, some studies have reported *syntax* and *logic* to be difficult programming concepts [16, 14, 18]. Morin and Kecskemety theorize that struggling students in introductory programming courses are linked to traditional programming courses having the students learn syntax and logic simultaneously [18]. Morin and Kecskemety suggest that one way to separate syntax and logic is through the use of *Parsons problems.* A Parsons problem is a code completion programming exercise where students rearrange a set of code blocks into a problem solution [3, 4]. We review Parsons problems in more detail in chapter 2.2. Studies have indicated positive effects of Parsons problems, suggesting that it is not just a more efficient way of practicing programming; but also just as effective as traditional programming exercises, such as code writing [19, 20, 21, 22].

Morin and Kecskemety explain how they implemented cooperative Parsons problems into their first-year programming course. The students would move physical strips of paper with lines of code to solve the Parson problem tasks. However, due to Covid-19, the university had to move its course to a virtual environment. To address this, they chose to implement a website where students could solve Parsons problems together. Students would cooperate through telecommunication applications such as Zoom to solve these tasks as a team [23]. One student would interact with the website at a time. Morin and Kecskemety reports that the students' experiences with cooperative Parsons problems skewed positively [18]. In addition, they claim "[in regards to feedback]... if we could reduce cognitive overload, students and faculty would be receptive to continued use of these activities in the classroom" [18, p. 9].

Cooperative learning and gamification

Laal et. al and Slavin claim that cooperative learning can result in increased achievement and improved social relationships [24, 25]. Susan A. Yoon et al. claims that transactivity, the depth of social interaction between participants in cooperative activities, is empirically tied to improved learning outcomes [26, p. 355]. In addition, studies suggest that gamification can promote learning and motivation [29, 28, 27]. A paper by Morschheuser et al. points to studies that "indicate that cooperative structures can promote greater efforts than individualistic or competitive approaches" [30, p. 2]. Morschheuser et al. also

claim that group work scenarios with positive goal interdependence can promote the creation of positive relationships and support psychological health. However, "gamification approaches that engage individuals to cooperate and, therefore, to strive toward a shared goal or purpose have been of minor focus in gamification and game-design thus far" [30, p. 1].

Loneliness and socialization

Loneliness appears to be a prevalent issue among students in higher education. Several universities have reported that a significant number of their students are feeling lonely, ranging between 27 and 65% [31, 32, 33, 34]. Additionally, a Norwegian survey from 2021 found that students are feeling increasingly lonely, with over 54% of participants answering yes to at least one of three questions regarding loneliness and isolation [35]. Universities UK claims that mental health and well-being can have implications for academic study and the student's overall university experience [36]. Kaufmann and Vallade also claim that loneliness can negatively impact a student's learning experience [37]. Studies also point to loneliness being a prevalent issue in online education [38, 37, 39]. In fact, Ali and Smith report that online programming students experience higher levels of isolation compared to students enrolled in campus courses [38]. Ali and Smith also claim that the feeling of isolation can influence a student's decision to withdraw from enrolled courses.

Additionally, the university is a transitional period for many young adults as they move away from family and established connections [36, 33, 40]. Many will have to adjust to new environments and people, and make substantial efforts to create personal connections, which can be difficult for some. Vasileiou et al. state that this transition is often characterized by an increased risk of loneliness, which can negatively impact mental health [40]. McIntyre et al. found loneliness to be the strongest overall predictor of mental distress [41]. The study concludes that establishing strong social connections is an important factor for mediating mental distress. Another study by Özdemir and Tuncay found that 46.9% of the participants reported that social interaction is an essential need during university education [33].

Vasileiou et al. suggest that universities should develop appropriate structures that can mitigate and intervene with these psychological issues among students [40]. Morschheuser et al. describe how cooperative video games can satisfy the need for *social relatedness*. To quote: "Especially, socializing with other players, the desire to form *meaningful relationships with others*, as well as satisfaction from *being part of a group effort* have been identified as important motivational gratifications of players in online games with cooperative features" [30, p. 4].

The need for a digital game

Paper-based cooperative Parsons problems require the students to be physically present. However, the recent Covid-19 pandemic has shown that physical attendance can be challenging. Even without Covid-19, a digital implementation will have advantages such as scalability, data gathering, and availability for online students. Regarding scalability, it could be easier to create many different tasks quickly by not having to print and cut paper for various tasks. Data gathered from the game, such as how students play and solve problems, can be used for learning analytics. Learning analytics can provide instructors with insight into what students tend to struggle with. Kanika et al. recommend developing specialized tools for teaching programming as it can motivate students [42].

1.2 Problem statement

With the motivation for this project laid out, we started to review existing Parsons problems software solutions. However, we could not find any cooperative solutions. Therefore, we decided to develop a multi-player cooperative Parsons problems game; with the intent to:

1. Increase learning outcome by reducing cognitive load and making use of cooperative learning.
2. Increase the use of social school activities.

In our solution, the players shall be able to solve Parsons problems cooperatively. The code blocks shall be distributed among the players to encourage communication. The players shall be able to interact with code blocks in various ways, including drag-and-drop and double-clicking. The game shall also provide feedback on task submission. We hope to combine the pedagogical advantages of Parsons problems with the social benefits of cooperation to provide students with an effective, fun, and social activity to learn to program. This game will, throughout this thesis, be referred to as *TPP; Team-based Parsons Problems*.

1.3 Research questions

- RQ1: Can cooperative Parsons problems be a fun and social activity for novice programming students?
- RQ2: What are the players' perceptions of the potential learning outcome from solving cooperative Parsons problems?
- RQ3: How does a team work together to solve cooperative Parsons problems in a digital space?
- RQ4: How could a cooperative multi-player Parsons problems game be designed?

We formulated these four research questions (RQ) to further explore the effects of Parsons problems used in a cooperative context. To answer RQ1 and RQ2, we will look at the possible benefits of using cooperative Parsons problems as a fun, social, and educational learning activity. We will use qualitative research methods to discuss whether cooperative Parsons problems exercises are worth further exploring. Additionally, to answer RQ3, we will perform multiple user tests to observe how a team works together to solve cooperative Parsons problems. To perform these user tests, we will need a minimum viable product; with the necessary functionality for solving tasks cooperatively. Lastly, RQ4 will be answered by discussing and reflecting on TPP's design- and implementation choices; based on feedback from the user testers and our observations.

1.4 Project goals

The project goals are categorized as effect goals and result goals. The result goals are the artifacts the project shall produce, whereas effect goals are the benefits these artifacts shall bring to the target group.

1.4.1 Result goals

- A game where students can solve cooperative Parsons problems; the game will be referred to as *Team-based Parsons problems (TPP)*.
- A tool for creating cooperative Parsons problem exercises; referred to as the *task-creation tool*.

First and foremost, we want to develop a game for solving Parsons problems cooperatively. We also decided to develop a tool to simplify the creation of Parsons problem exercises.

1.4.2 Effect goals

- Make programming more fun and engaging for novice programming students.
- Increase learning outcomes for novice programming students.
- Help students socialize through school activities.

TPP shall help students to learn programming through code completion exercises. Furthermore, TPP shall facilitate a cooperative learning environment in order to make programming more fun and engaging. Additionally, TPP shall provide an activity for students to socialize through cooperative work.

1.5 Project scope

In this project, we shall focus mainly on the game; TPP. We shall use qualitative testing methods to test whether the game is perceived as a social and fun learning activity. However, measuring the social benefits of this game goes beyond the scope of this project. We shall also focus solely on the programming language Python; for in-game programming exercises. We chose Python as it is widely taught at our university, especially in novice programming courses within the natural sciences.

Although we intend that TPP should increase learning outcomes, we shall not test this. Testing this would require a thorough study comparing different learning methods, involving a great number of participants to show statistical significance. Time prevents us from performing such a task. Data gathering for learning analytics was also a motivation for TPP. However, we did not have the time to implement such features.

We use Encyclopedia Britannica to define *learning*, *fun*, and *socialization*. Learning is defined to be the 'the alteration of behavior as a result of individual experience' [2]. Socialization is defined to be 'the process whereby an individual learns to adjust to a group' [5]. Finally, fun is defined to be 'the feeling of being amused or entertained' [1].

1.6 Target group

The main target group for TPP is novice programming students; in higher education. However, TPP could just as well be used in other levels of education. Additionally, we want all novice programming students to benefit from TPP, even though some of our motivation comes from struggling students.

1.7 Reader guide

This thesis is structured as follows: literature review, methodology, results, discussion, and conclusion. Our literature review in chapter 2 presents the concepts, theory, and research articles relevant to our research questions and project goals. Chapter 3 describes our methodology; as we worked on the various TPP iterations. Each subchapter in chapter 3 represents a software development iteration, which consists of requirement specification, design, development, and user testing. In chapter 4 we present our results relating to our four research questions; based on feedback and observations from our user tests. We discuss our findings and evaluate our project in chapter 5. Chapter 5 also includes our recommendations for further work. Lastly, we summarize the thesis and make our conclusions in chapter 6.

2 Literature review

This chapter presents the concepts, theory, and research articles relevant to our research questions and project goals. First, we will look at different programming exercises, including Parsons problems. Learning more about the uses and variants of Parsons problems will help us identify necessary in-game functionality. We will also review some existing Parsons problems software solutions. Next, we will look at learning and gamification concepts to help us understand how to design TPP; to be an engaging and cooperative experience that can promote learning. We will also look at the design thinking and design science approaches; to help us make informed decisions on how to approach the development and definition of TPP itself. We will also present the design principles of Universal Design, which will act as guidelines for how to make TPP as accessible and usable as possible. Next, we will look at the peer-to-peer network architecture, the selected method of communication for our game. To help us perform our analysis, we will look at System Usability Scale and thematic analysis. Finally, we will look at game- and software design to guide our design process.

2.1 Different types of programming exercises

In the following two paragraphs, we present two papers that attempt to classify different types of programming exercises. This classification is included to put Parsons problems into context, which is the relevant programming exercise for this thesis.

Kanika et al. performed a literature study on papers discussing different types of programming exercises [42]. The survey resulted in five classifications for different types of programming exercises: *visual programming*, *game-based*, *collaborative programming*, *robot programming*, and *assessment systems*. *Visual programming* is an exercise where students interact with code using a visual programming language. The students will drag and drop necessary programming constructs on a canvas representing a flowchart-like diagram. Kanika et al. claim that visual programming can be used effectively as an introduction for schoolchildren and teachers who lack previous programming experience. However, developing an IDE to facilitate these exercises requires substantial time and effort. *Game-based exercises* have the students learn programming by writing programs to play visual games in suitably designed environments. Kanika et al. refer to a study, by Leutenegger and Edgington, where programming was taught by making students write Java code to display and manipulate objects [43]. Game-based exercises can be personalized according to the requirements of individual learners, but as with visual programming exercises, having to develop an IDE is a disadvantage. *Pair and collaborative programming* exercises have students write programs in pairs or small groups. Collaborative programming allows students to get feedback from each other and discuss difficult concepts. However, finding the right group composition can be a challenge. *Robot programming* has students write programs to control robots and make them perform simple tasks in the real world. These exercises are more hands-on and less abstract than other exercise types. Robot programming has been proved to be quite effective in motivating students to learn to program, but the availability of robots can be a limitation. Lastly, *assessment systems* are programs written by students that are checked and graded automatically. Automatic grading allows students to get more feedback when writing a large volume of programs; that otherwise would have to be checked by teaching assistants or other peers. Automatic grading increases the scalability of the course, but certain aspects of programming can be difficult to assess automatically. Conclusively, Kanika et al. claim that developing specialized tools for teaching computer programming can motivate students and should be utilized more in programming courses [42].

Chirumamilla and Sindre categorize programming exercises into three categories: code tracing, code writing, and code completion [44]. In *code tracing exercises*, the code is given, and it is the student's task to explain what the code does. Code tracing could vary from exercises that require a brief explanation to exercises that require a detailed line for line explanation. In *code writing exercises*, it is explained what the program is supposed to do, and it is the students' task to write the code. In *code completion exercises*, it is explained what the program is supposed to do, and some code is provided, but the code is not complete. The student's task is to fill in, select the missing parts or rearrange the code lines in the correct order. Code completion exercises could also include exercises where all the code is given in a shuffled order, and it is the student's task to rearrange the code to a correct solution. These rearranging

exercises are often referred to as *Parsons problems* [44].

2.2 Parsons problems

> *Parsons problems are the code exercises students will solve with TPP. By reviewing Parsons problems, their uses, and variants, we will gain a better understanding of how users will interact with TPP and thus help us define our requirement specification.*

A Parsons problem is a programming exercise where students rearrange a set of code blocks into a problem solution [3, 4]. Studies suggest that solving Parsons problems can lead to the same learning outcome as writing equivalent code. In addition, solving Parsons problems is measured to take less time than writing equivalent code [19, 20]. This could lead to a faster feedback cycle for the students.

2.2.1 Distractors

Distractors are code blocks that are not useful for solving the Parsons problem. Distractors can be syntactically or semantically incorrect. For example, a distractor could have a missing semicolon or have the wrong boundary condition. Distractor blocks can be randomly mixed with the correct blocks or paired such that for each correct block, there is a distractor [20].

2.2.2 Pre-scaffolded and student-scaffolded

In *pre-scaffolded* problems, students only need to focus on the code block order. On the other hand, *student-scaffolded* problems require the student to consider code formatting. For example, a student-scaffolded problem could require braces in Java or indentation in Python. Research indicates that student-scaffolded problems are more challenging [46, 4, 45].

2.2.3 Line-based and execution-based feedback

The original creators of Parsons problems had a *line-based feedback* system where code blocks in the wrong absolute positions were highlighted [3]. However, later designs also marked incorrectly indented lines and only showed the first code block in the wrong absolute position. Developers have also implemented feedback systems for partial solutions by finding the longest common sequence between student and model solutions. However, the feedback system for a partial solution is not yet tested on students, according to the study [45].

Some critics of the line-based system claim that it encourages trial-and-error behavior. Line-based systems also assume that a problem has a single unique solution, which limits the amount of Parsons problems that can be made [47]. Another feedback system that has been tried with Parsons problems is *execution-based feedback*. A study suggests that the students had fewer states where the code would fail to execute with execution-based feedback. In addition, feedback was requested less frequently when using execution-based feedback [48, 4].

2.2.4 Part-complete problems

Part-complete Parsons problems provide either completed code lines or partially completed code lines that need to be completed by the student. A study suggests that part-complete Parsons problems can be used to reduce cognitive load even further compared to fully-complete Parsons problems, see chapter 2.4.1 [49, 4].

2.3 Existing Parsons problems software solutions

We did some research on existing Parsons problems software solutions. While we did not find any existing solutions that allowed for cooperation, we found some solutions that allowed for creating and solving Parsons problems. In this chapter, we review these and summarize the ones that seemed the most relevant to our project.

2.3.1 Js-parsons

Js-parsons is a JavaScript library that appears in multiple Parsons problems applications [50]. Js-parsons includes various features relating to solving Parsons problems. Figure 1 shows an example of a Parsons problem from their website. This figure shows two fields that can contain code blocks. A user can move code blocks between these two fields; using drag-and-drop. The left side represents the handout code, and the right side represents the user's answer. The answer field considers indentation. Visual feedback on where a block can be placed is provided by freezing the code block at each indent. The user can check their answer by pressing the 'Feedback' button. Js-parsons provides various feedback options, including line-based and execution-based feedback, see chapter 2.2.3. Each code block will be highlighted in green or red based on whether a placement is correct or incorrect. Figure 2 shows feedback for an incorrect position and incorrect indentation.

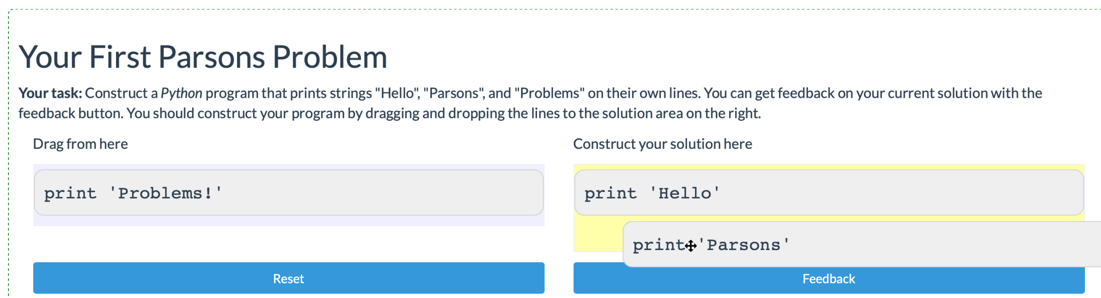
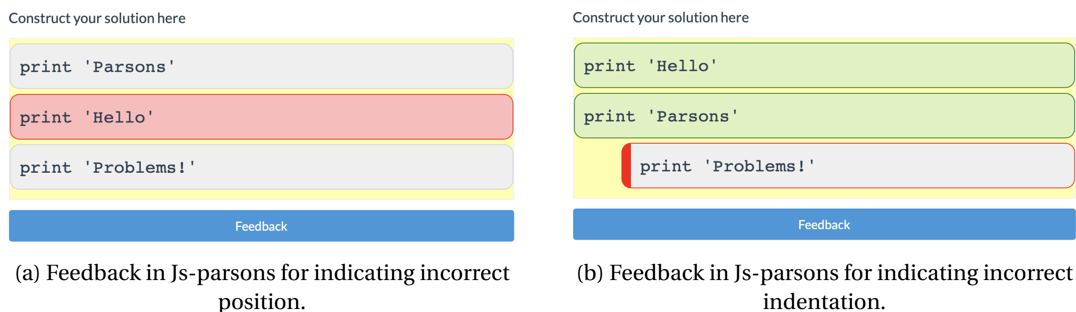


Figure 1: A Js-parsons example task.



(a) Feedback in Js-parsons for indicating incorrect position.

(b) Feedback in Js-parsons for indicating incorrect indentation.

Figure 2: Feedback in Js-parsons.

2.3.2 Parsons Problem Webapp

Parsons Problem Webapp is an open-source application that allows users to write code into an editor. The user can then use this code to generate a Parsons problem [51]. Figure 3 shows the editor where the user has written a program that the Parsons problem will be generated from. This editor includes basic syntax highlighting and will mark where each indent begins. After providing a title, the user can click 'Create Puzzle' to generate a Parsons problem. This will open a new page where the user can solve the task, see figure 4. Each line of code from the editor becomes a code block; in a shuffled order. The page for solving the Parsons problem utilizes the Js-parsons library, which we covered in detail in chapter

2.3.1. As the creators of Parsons Problem Webapp mentioned, the URL for the generated Parsons problem can be used as-is, or it can be embedded into an IFrame as part of a course, for example, in a *Jupyter Notebook* code cell [52].


```
def sum_list(items):
    sum_numbers = 0
    for x in items:
        sum_numbers += x
    return sum_numbers
```


Figure 3: Parsons Problems Webapp editor for writing Parsons problems.

Figure 4: A generated task with Parsons Problems Webapp.

2.4 Learning and cooperation

In this chapter, we present relevant theories on the topics of learning and cooperation. We will utilize these theories to design TPP in such a way that it can promote learning and socialization. Additionally, we will review cooperative gamification, which will act as an inspiration for how to make TPP effective and enjoyable as a user experience.

2.4.1 Cognitive load

Cognitive load refers to the mental resources that are required by a particular task [53]. Information can only be stored once the working memory has processed the information. Morin and Kecskemety claim that a heavy cognitive load can negatively impact one's ability to solve a task. Morin and Kecskemety suggest that one way to reduce cognitive load is by using Parsons problems, as it separates syntax and logic [18].

2.4.2 Transactivity

Transactivity can be used to measure the depth of social interaction between participants in cooperative activities. Transactivity occurs when a learner engages with and builds on a peer's learning contribution or reasoning [26, p. 355]. A paper published by the International Journal of Computer-Supported Collaborative Learning claims that higher occurrences of transactivity have been empirically tied to improved learning outcomes [26, p. 352]. In a transactive discussion, the learner can be encouraged to support their claims with evidence and explain their reasoning. Transactivity can further be divided into two categories: *dialogic transactivity* and *dialectic transactivity*. Dialogic transactivity is based on building on the ideas of others; from the point of agreement or elaboration. Dialectic transactivity is based on building new ideas through conflict and resolution.

2.4.3 Cooperative gamification

Gamification is the process of adding game elements to non-game activities. The purpose of adding game elements is to give similar experiences as games commonly do. Studies suggest that gamification can promote motivation and learning [29, 28, 27]. When gamification is added to a cooperative activity, it is called cooperative gamification. This could, for example, be in a learning system, online community, or business intranet. The goal of cooperative gamification is to encourage participants to share and interact with each other to solve a shared goal [54, 55, 30]. A study from 2017 compares two information systems used within an engineering company. Only one of the information systems utilized gamification features. Employees that tried the version with gamification expressed that they were more likely to use the system. According to a questionnaire, the same employees also showed more enjoyment using this system [30]. Another study from NTNU claims that cooperative gamification can support idea generation, critical analysis, articulate thinking, and collaboration strategies. The study used a collaborative card game that was developed for idea generation in the field of internet of things [56].

2.5 Design thinking

> *Design thinking seemed suitable for incorporating continuous feedback from the target group throughout development. By utilizing design thinking, we wanted to create a positive user experience with our game.*

Design thinking is a human-centered approach to developing software [57]. In design thinking, the designers seek to understand the user's challenges, assumptions, and level of understanding. The iterative process is utilized to acquire knowledge about the users. The Hasso-Plattner model separates the process into five stages: *empathize*, *define*, *ideate*, *prototype*, and *test*. The five stages are not always sequential, which makes design thinking an iterative and non-linear process [58].

- Empathize: Try to identify as the user.
- Define: What are the user needs and problems?
- Ideate: Generate innovative solutions.
- Prototype: Create an early version of the application.
- Test: What did/did not work with the prototype?

2.6 Design science

We used design science to validate TPP. Design science was relevant as we attempted to answer the research questions formulated in chapter 1.3. Design science also seemed to be an approach that would work well in joint with design thinking.

Design science is a research approach where the goal is to construct a new reality [59]. This distinguishes design science from research that tries to explain an existing reality. Constructing a new reality can be interpreted as solving problems. Design science serves two purposes. Firstly, to utilize knowledge to solve problems. Secondly, to generate new knowledge. Design science consists of three phases. Phase 1 is exploration, induction, and deduction of the problem, the context, the activities, and the hypothesis. Phase 2 is the design and testing of the solutions. Phase 3 consists of verifying the hypothesis, validating the research, and generalizing toward other applications. Design science includes six activities:

1. Identification: Defining the research problem and justifying the solution.
2. Definition: Goals and requirements for the solution.
3. Design and development of artifacts.
4. Demonstration: Use the artifact to solve the problem.
5. Evaluation: Compare the goals and requirements to the observed results.
6. Communication: Share the research with other researchers and practicing professionals.

Most design science research focuses on people. Therefore it is essential to use methods from the realm of user design, such as design thinking [59], see chapter 2.5.

2.7 Universal Design

When designing TPP, we looked at Universal Design as a set of principles to ensure we were making good design choices; in order to make TPP as accessible, intuitive, and easy to use as possible.

Universal Design (UD) is a set of principles that attempts to make the design and composition of a physical or digital environment accessible and understandable for as many people as possible. At its core, UD is about making design choices that meet the need of anyone who wishes to use it; by considering the diverse needs and abilities of all. To quote the Centre for Excellence in Universal Design: "Simply put, universal design is good design" [60]. UD has seven principles, developed by a working group of architects, product designers, engineers, and environmental design researchers at North Carolina State University in 1997. These principles act as a guide during the design process. The following list shows the seven principles. Their website also includes more detailed guidelines for each principle [60].

1. **Equitable Use:** the design is useful and marketable to people with diverse abilities.
2. **Flexibility in Use:** the design accommodates a wide range of individual preferences and abilities.
3. **Simple and Intuitive Use:** the design is easy to understand, regardless of the user's background.
4. **Perceptible Information:** the design communicates necessary information effectively to the user, regardless of ambient conditions or the user's sensory abilities.
5. **Tolerance for Error**: the design minimizes hazards and the adverse consequences of accidental or unintended actions.
6. **Low Physical Effort:** the design can be used efficiently and comfortably and with a minimum of fatigue.
7. **Size and Space for Approach and Use:** the user is provided with appropriate size and space, regardless of the user's physical attributes.

2.8 Peer-to-peer

We had to look into suitable network architectures for TPP's communication; that could facilitate a multi-user cooperative game. Peer-to-peer became a primary candidate for its low delay and maintenance.

Peer-to-peer (P2P) is referred to as a network architecture for communication and resource sharing between computers [61]. In a P2P network, each peer can freely share files and communicate directly without the need for a separate server, see figure 5. This can make direct messaging between peers faster compared to a more traditional client-server architecture, as payloads will not have to pass through an intermediate server. In addition, P2P has the clients take on the roles of both supplier and consumer. The primary goal of P2P networks is to share resources and help computers and devices work cooperatively, provide specific services, or execute specific tasks [62].

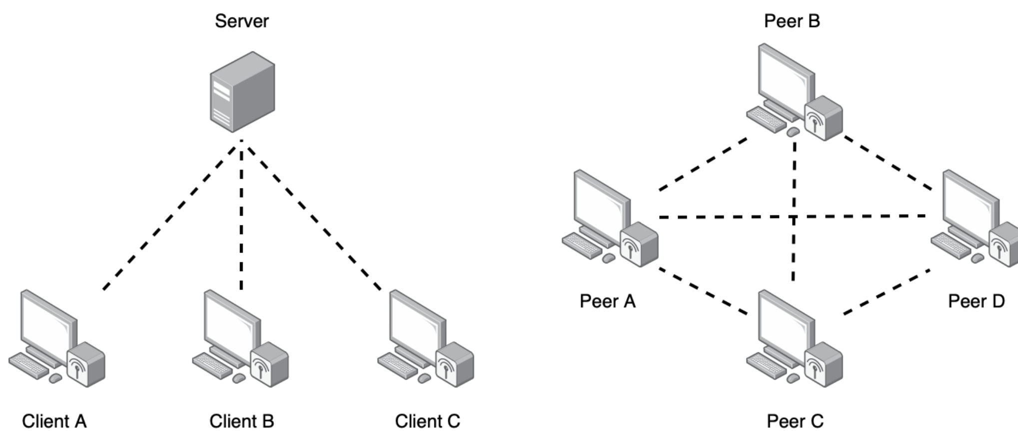


Figure 5: Client-server architecture (left) and peer-to-peer architecture (right).

2.8.1 Technology

A popular framework for facilitating P2P is WebRTC (Web Real-Time Communication) by Google Developers [63]. WebRTC is a free, open-source framework for the web that enables real-time communication between browsers. This framework consists of standards, protocols, and JavaScript APIs. Most popular browsers are compatible with WebRTC, which eliminates the need for any intermediate web servers once the connection is established [64]. However, some services must act as a matchmaker to manage the connection between the peers. A *signaling server* usually handles this. The signaling server does not handle the traffic, but it helps establish, reset and close connections [65].

A signaling server is also required for *network address translation (NAT) traversal* and Interactive Connectivity Establishment candidate exchange. NAT traversal is a networking technique to establish and maintain connections across gateways that implement NAT. This means that if a peer is behind a NAT, the other peers will not be able to communicate with it. One solution for NAT traversal is *Session Traversal Utilities for NAT (STUN)* [66]. As clients do not know their public IP address, they will have to send a request to a STUN server to return their public address. This address can then be shared with its peers to establish the connection [67].

2.8.2 Peer-to-peer in online games

Another use case for P2P communication is in online video games. Again, the benefit is that it eliminates the need for a separate server that bridges the communication between each peer. Some examples of games utilizing P2P are Dark Souls, Super Smash Bros Ultimate and Grand Theft Auto Online [68]. P2P has some benefits like reducing delay, avoiding server issues taking down games, and reducing the cost

of running and maintaining a central server. However, some drawbacks are cheating through manipulation of data and the requirement that all peers need a good internet connection for the game to function properly.

Another potential issue of P2P games is that there is no authorization of the game state; since each peer has it stored locally. A local game state can introduce some de-synchronization issues since there is no centralized, authoritative server that handles game state updates. Furthermore, if a peer receives a game state update before it has sent local changes, there is a risk that some actions can be overwritten. A solution to this is to declare one peer as the 'host', which will take the role of a dedicated server for the session. Depending on how the game is implemented, this could mean that each peer sends requests to the host to change the game state, in which the host forwards the updated game state to all peers [68].

2.9 System Usability Scale

> *For assessing the usability of TPP, we looked for a tried and tested method. After doing some research, we were made aware of the System Usability Scale, a reliable and valid tool for measuring the usability of a system or service.*

System Usability Scale (SUS) is a tool created by John Brooke in 1986 for measuring the usability of various products and services, including hardware, software, websites, and applications [69]. SUS is industry standard and is referenced in over 1300 articles and publications. Some of the benefits of SUS are that it is easy to scale, can be used on a small sample size with reliable results, and is valid in differentiating usable and unusable systems. When using SUS, participants are asked to score ten statements with one of five responses that range from 'Strongly agree' to 'Strongly disagree.' These scores are then condensed into a single score ranging from 0 to 100. The score rates the system's performance in effectiveness, efficiency, and overall ease of use. Table 1 lays out a general guideline on the interpretation of the SUS score according to UIUXTrend [69]. A score above 68 is considered *acceptable*. The following template shows the ten questions the participants are asked to answer:

1. I think that I would like to use this [product] frequently.
2. I found the [product] unnecessarily complex.
3. I thought the [product] was easy to use.
4. I think that I would need the support of a technical person to be able to use this [product].
5. I found the various functions in this [product] were well integrated.
6. I thought there was too much inconsistency in this [product].
7. I would imagine that most people would learn to use this [product] quickly.
8. I found the [product] very cumbersome to use.
9. I felt very confident using the [product].
10. I needed to learn a lot of things before I could get going with this [product].

Each question weighs 10 points. Every odd-numbered question yields a positive score, and every even-numbered question yields a negative score. Looking at the odd-numbered questions, if a user strongly agrees with one statement, the maximum point is 10. By subtracting 1 from each odd-numbered question, the minimum point is 0. By multiplying this point by 2.5, the maximum score of 10 is ensured for each question. For example, if a user strongly agrees with question 1, the calculation would be as follows: (5 - 1) x 2.5 = 10. This calculation works the same for the even-numbered questions, but here 'strongly agree' yields a minimum point of 0. Therefore, we subtract the points of each question from 5 to ensure the minimum is 0. Figure 6 shows how the SUS score is calculated.

$$X = \text{Sum of the points for all odd-numbered questions} - 5$$
$$Y = 25 - \text{Sum of the points for all even-numbered questions}$$
$$\text{SUS Score} = (X + Y)x2.5$$

Figure 6: Formula for calculating the System Usability Scale score.

SUS Score	Grade	Adjective Rating
>80.3	A	Excellent
68 - 80.3	B	Good
68	C	Okay
51 - 68	D	Poor
<51	F	Awful

Table 1: Grades and adjective ratings of System Usability Scale scores.

2.10 Thematic analysis

To answer our research questions, we looked at methods for analyzing the qualitative data from our user test surveys. After doing some research, we found thematic analysis to be cited as an accessible form of analysis for qualitative research, particularly for those who are unfamiliar with qualitative analysis methods [70, p.81].

Qualitative research often aims to generate knowledge from subjective information such as experiences, views and opinions. This information is usually collected from semi-structured interviews and open-ended surveys. However, qualitative research can generate a vast quantity of data that can be hard to deduct any meaning from. There are many methods for analyzing qualitative data, and *thematic analysis* (TA) is one of them [70]. TA aims to provide a flexible and intuitive approach that can be utilized for many kinds of research. TA works by recognizing patterns in the data to find overarching themes that can help identify its meaning. This process is driven by the research questions that help define what kinds of themes to look for.

TA consists of 6 phases as outlined by Braun and Clarke [70]. However, this guideline is not a rigid set of rules as researchers are recommended to alter them as they seem fit for their research. They also recommend not treating these phases as linear but rather as an iterative process where researchers are recommended to go back as they get a better understanding of the data. Table 2 gives an overview of the 6 phases [70, p. 89]. The first phase is to get familiar with the data. Familiarizing means reading and re-reading the data to get a good overview while also taking initial notes for later. Next is the coding phase, where the analyst highlights text sections and comes up with shorthand labels, or 'codes,' to describe their content. Each code describes an idea or feeling expressed in that part of the text. Finally, the data is collated together into groups, identified by their code.

Phase 3 is to generate themes by identifying patterns in the codes created in the previous phase. A theme will usually be a combination of several codes, but sometimes a code can work as a theme by itself. The next phase is to review the themes by comparing them to the data. The themes should accurately represent the data in order to be useful. If there are any issues with the themes, the analyst can split them up, combine them, discard them, or create new ones. When a final list of themes is constructed, the analyst can move to phase 5. Here the themes are further defined and named to avoid confusion or misinterpretation. Lastly, phase 6 is to write the report, including the described themes, and provide examples that meaningfully contribute to the research questions.

When performing TA, researchers must decide if they want to take an *inductive* or *deductive* approach. An inductive approach means the identified themes are strongly linked to the data. This means that the researcher will not try to fit the coded data into a pre-existing coding frame, as the identified themes may bear little relation to the specific questions that were asked. On the contrary, the deductive approach, also called 'theoretical' thematic analysis, would tend to be driven by the researcher's theoretical or analytic interest in the area. This makes the approach more explicitly analyst-driven [70, p.84]. In addition,

Phase		Description of the process
1	Familiarizing yourself with the data	Transcribing data (if necessary), reading and re-reading the data, and noting down initial ideas.
2	Generating initial codes	Coding interesting features of the data in a systematic fashion across the entire data set, collating data relevant to each code.
3	Searching for themes	Collating codes into potential themes, gathering all data relevant to each potential theme.
4	Reviewing the themes	Checking if the themes work in relation to the coded extracts (Level 1) and the entire data set (Level 2), generating a thematic 'map' of the analysis.
5	Defining and naming themes	Ongoing analysis to refine the specifics of each theme and the overall story the analysis tells, generating clear definitions and names for each theme.
6	Producing the report	The final opportunity for analysis. Selection of vivid, compelling extract examples, the final analysis of selected extracts, relating the analysis to the research question and literature, producing a scholarly report of the analysis.

Table 2: The six phases of thematic analysis.

the deductive approach is the most applicable if the data is coded for a specific research question. In contrast, the inductive approach allows the research question to evolve over time.

Another decision that must be made for TA is at which 'level' the themes should be identified at. With a *semantic* approach, the data is analyzed explicitly and on a surface level. In contrast, the *latent* approach attempts to identify the more profound meaning and subtext of the data. The researcher must then decide if they are interested in the participant's stated opinions or what their statements reveal about their assumptions and social context.

2.11 Game theory and game design

When designing TPP, we reviewed literature on the topics of game design and game theory. By utilizing game design and game theory, we were able to make more informed decisions regarding the game aspects of TPP.

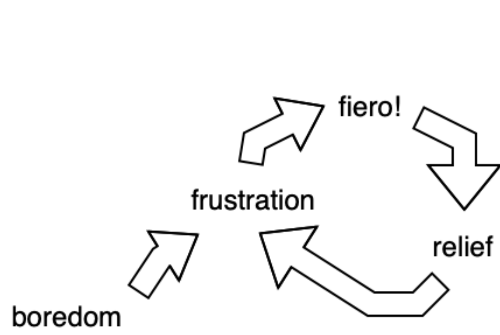
2.11.1 The four fun keys

Lazzaro tries to define what makes a game *fun* in the article 'The four fun keys.' [71]. She argues that games are played for fun, distinguishing them from traditional software where usability and productivity are more important. According to Lazzaro, *fun* corresponds to certain emotions. These emotions are categorized into four keys: *hard fun*, *easy fun*, *serious fun*, and *people fun*. Emotions can be triggered in games through choices and feedback. Choices and feedback are later referred to as game mechanics. See figure 7 for the emotions involved in each key.

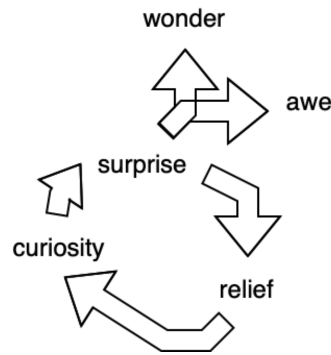
Hard fun is about providing challenges for the player. If the game is too easy, the player might get bored. If the game is too hard, the player might give up and quit in frustration. Different players prefer different levels of challenge. Examples of mechanics in hard fun includes goals, obstacles, puzzles, and score. These mechanics could lead to frustration, boredom and *fiero*. Fiero is an Italian word meaning triumph over adversity.

Easy fun is about giving the player the freedom to explore the game. Good games offer interactions outside the main challenges to inspire imagination and capture attention. Players often switch from hard fun to easy fun when a challenge becomes too hard. Examples of mechanics in easy fun include role play, exploration, experimentation, and fooling around. These mechanics could lead to curiosity, surprise, wonder, and awe.

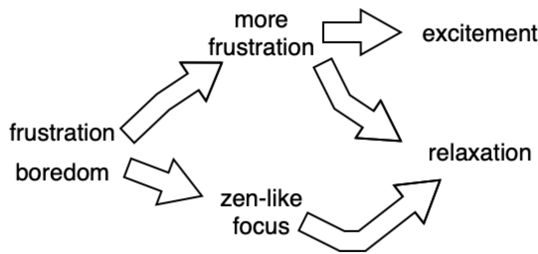
Serious fun is where players create value outside of the game itself. This could, for example, be relax-



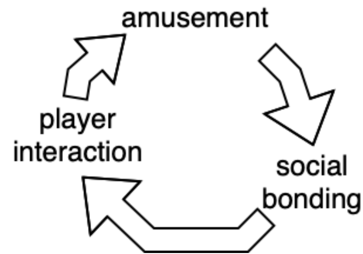
(a) Hard Fun PX Spiral. Fiero, meaning triumph over adversity in Italian, is crucial for hard fun.



(b) Easy Fun PX Spiral. Leave the player in awe and wonder through exploration.



(c) Serious Fun PX Spiral. Create something of value outside of the game itself.



(d) People Fun PX Spiral. Players experience amusement through player interaction, leading to social bonding.

Figure 7: The Player Experience (PX) spirals for the four fun keys.

ation after a hard day at work, exercising, or learning something new. Serious fun could be described as gamification the other way; see chapter 2.4.3. Serious fun adds traditional elements to a game, while gamification adds game elements to traditional software. Games that lack serious fun might feel like a waste of time. Examples of mechanics in serious fun include practice, completion, meditation, or study. These mechanics could lead to relaxation, excitement, and self-esteem boosts. Emotions in serious fun are related to values such as killing time, losing weight, or developing skills.

People fun comes from social interaction. People that play together in the same room express more feelings compared to people playing the same game in different rooms. Emotions can also be triggered through the use of non-player characters. Examples of mechanics in people fun could be cooperation, competition, and communication. These mechanics could lead to amusement, social bonding, envy, love, gratitude, generosity, and *schadenfreude*. Schadenfreude is a German word meaning happiness over others' suffering.

2.11.2 Buchinger and Hounsell's 19 design guidelines

Buchinger and Hounsell present a paper that provides a list of guidelines for designing games that involve collaboration, competition, and learning. The first nine guidelines are a result of a literature study. Buchinger and Hounsell used these guidelines as a baseline for designing a game. The last ten guidelines are observations from testing that game [72]. Table 3 shows the complete list of guidelines.

Number	Guideline	Explanation
1	Intra Players Interaction	Decide whether or not there are fixed teams or if the players are free to decide with whom to co-operate. In fixed teams, it needs to be decided if the players can choose their own team since friendship can both improve performance and act as a distraction. If they are free to decide, it is important to study player behavior.
2	Synchronization	Synchronous games provide quicker gameplay and feedback. Turn-based games give players more time to think about their actions, but long waiting times lead to boredom.
3	Roles	If collaboration is the most important, players should have different roles. Players should have the same role if the content is the most important.
4	Resources	Could be used for stimulating collaborative and competitive behaviors in the game.
5	Score	Could be used to motivate and stimulate competitiveness and discourage player error.
6	Challenge	Players that are challenged with the right amount of difficulty leads to motivation. In terms of competitiveness, balanced teams motivate players; unbalanced teams do not motivate.
7	Reward	Extrinsic motivation, such as ranking and achievement, works. Ranking means showing the best scores in public. Achievements in games relate to distinctions by completing objectives.
8	Artificial Intelligence	Can contribute to immersion and challenge.
9	Operationalization	Consider hardware requirements, the number of players that need to get together simultaneously, and the possibility for players to invent new ways of playing.
10	Number of players	The number of students in a classroom might not be a multiple of the number of players needed. Therefore the team size should be flexible. The game should also consider that players might quit during the game.
11	Amount of playtime	Study when players fatigue to find the ideal amount of playtime.
12	Communication	Study how players communicate.
13	Completeness	A self-contained game that explains itself is more likely to be used.
14	Infrastructure	Game designers should know about the available hardware for the target group.
15	User background	Game designers should consider the target group's knowledge about digital devices.
16	Domain experts and supporters	Welcome domain experts and supporters when designing.
17	Follow up	Arrange an activity in the classroom to inform students of the correct answers. Players can have misunderstandings after playing a learning game.
18	Assessment	It is interesting for researchers and educators to study how players performed. The players could be sorted into different groups, in the next game, based on assessment.
19	Conflict management	Collaborative, competitive, and pedagogical aspects need to be balanced since they can be in conflict with each other. Choose what to prioritize.

Table 3: Buchinger and Hounsell's 19 guidelines for designing games.

2.12 Zhang's ten design principles

While doing research for TPP's design, we found a paper from Zhang that proposed a variety of system design principles and characteristics related to human motivational needs. These principles would help us make more informed decisions when designing TPP to enhance its use.

Zhang presents ten design principles, see table 4, for creating motivational *affordances* [73]. Affordances are a design thinking term, meaning the property of an object that defines its possible uses [74]. For example, an affordance for a chair could be sitting.

Motivational Sources and Needs	Design Principles	Some examples
Psychological: Autonomy and the Self	**Principle 1**: Support autonomy. **Principle 2**: Promote creation and representation of self-identity.	Desktop skins, ring tones, online avatars.
Cognitive: Competence and Achievement	**Principle 3**: Design for optimal challenge. **Principle 4**: Provide timely and positive feedback.	Games and learning systems with various challenge levels and immediate performance feedback.
Social & Psychological: Relatedness	**Principle 5**: Facilitate human-human interaction. **Principle 6**: Represent human social bond.	Group chat, visualizations of email exchanges over a period of time to show both tasks and social related messages.
Social & Psychological: Leadership and Followership	**Principle 7**: Facilitate one's desire to influence others. **Principle 8**: Facilitate one's desire to be influenced by others.	Blogs, virtual communities where leaders emerge
Emotional: Affect and Emotion	**Principle 9**: Induce intended emotions via initial exposure to technology. **Principle 10**: Induce intended emotions via intensive interaction with technology	Slick/attractive look of cellphones, engaging games, technology that induce optimal flow experience.

Table 4: Zhang's ten design principles for motivational affordances.

3 Methodology

This chapter will describe our methodology as we worked on the various TPP iterations. Each iteration consisted of: requirement specification, design, development, and user testing, see figure 8. These relate to the phases in design thinking and design science; see chapter 2.5 and 2.6. Each iteration would start with a requirement specification. In this phase, we considered any feedback from the previous user test and updated our requirements accordingly. After this, we moved on to the design phase, where we sketched out and finalized a design that addressed the requirements. Next, we developed the iterated version of TPP by following the updated design documents. The development followed 2-week Sprint cycles, starting with a Sprint meeting and ending with a Sprint Review meeting with our supervisor. After we finished development, we moved to user testing to gather feedback from students in our target group. Once we completed the user test, the iteration was complete, and the next one started. Figure 9 shows a timeline of the development of TPP.

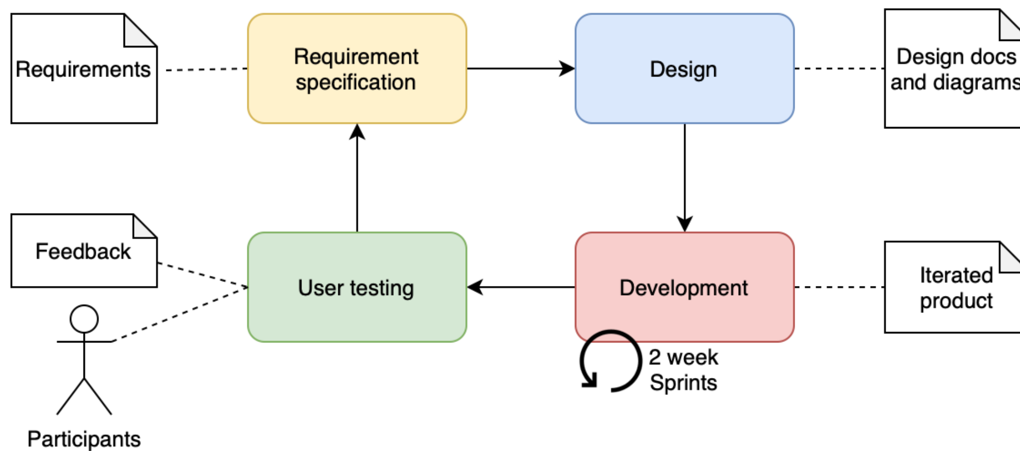


Figure 8: The four phases of our iterative process.

3.1 First iteration

In the first iteration, we followed the design thinking methodology to empathize, define, ideate, prototype, and test. We managed to empathize with the users as we had also been novice programmers at one point. However, it had been some years since we first started to learn how to code. We would define the students' needs and draft a few possible solutions. These hand-drawn drafts were then tested on students. We also followed the design science methodology during this iteration. Firstly, we identified the problem by gaining insight into the field. Secondly, a low-fidelity prototype was designed and evaluated. Lastly, we formulated a definition through early goals and requirements.

3.1.1 Requirement specification

During the first meeting with our supervisor, we discussed our main project goal: to create a multiplayer code-puzzle game where students cooperate to solve Parsons problems, see chapter 2.2. We specified a few requirements, which were used as a baseline for the first designs:

- The game shall be multiplayer.
- The players shall cooperate on solving Parsons problems.
- The code blocks shall be distributed between the players to encourage communication.

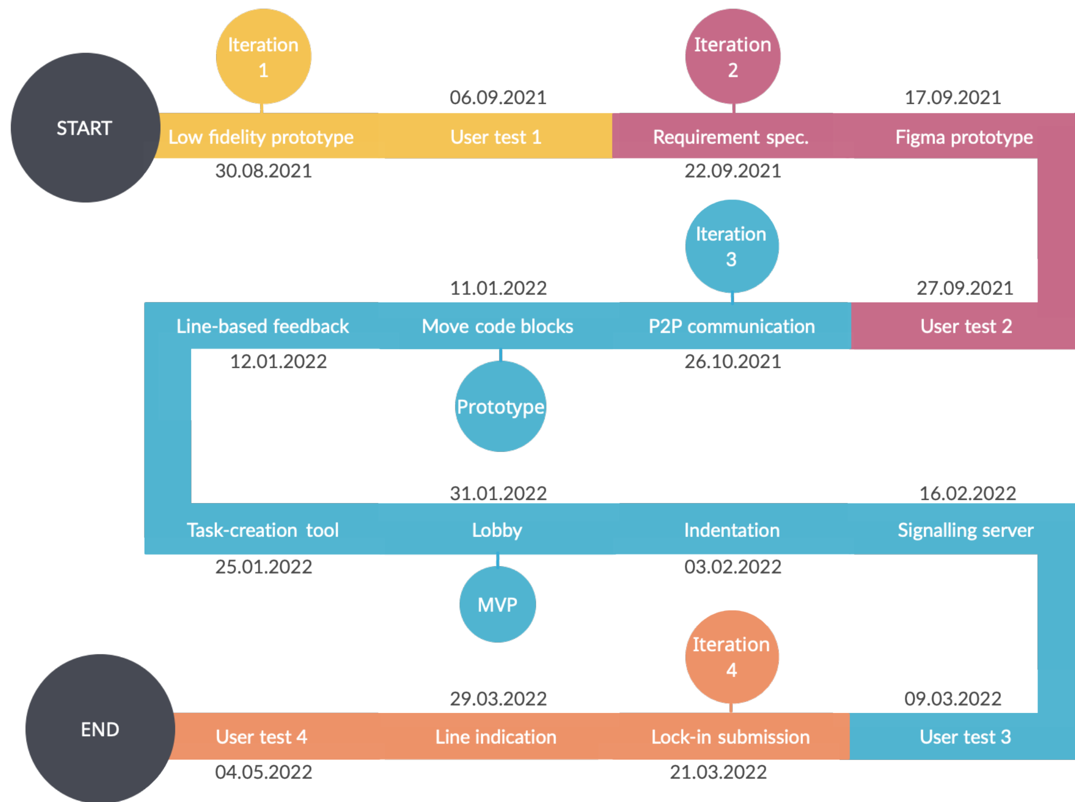


Figure 9: Timeline showing key events, when primary functionality was implemented, and the duration of the four iterations. It also shows when TPP fulfilled all of its requirements from a certain priority category (Prototype and MVP), see appendix A.3.

3.1.2 Design

We drafted two designs in this iteration which would later be used in an A/B test. Both designs consisted of initial sketches and refined versions, which acted as prototypes for user testing. Design A had a simple and streamlined look; see figure 10. The main feature of this design was the separation of the code blocks the player possessed (left) and the solutionfield where the blocks should be placed (right). This design focused on emphasizing these two elements to make it intuitive where the code blocks should go. Design B had a more game-like look; see figure 11. Initially, we discussed referencing traditional card- and board games where each player possessed cards with code that could be placed on a table. Design B implements this idea by placing a solutionfield in the center and surrounding it with player icons. This design emphasized the game's multiplayer aspect as each player was represented within the interface. At this point, we had also decided that we wanted TPP to include at least three out of the four fun keys, see chapter 2.11.1. Firstly, we wanted TPP to include *hard fun* through challenging puzzles and accomplishments. Secondly, learning would represent *serious fun*. Lastly, we wanted *people fun* to be represented through social interaction.

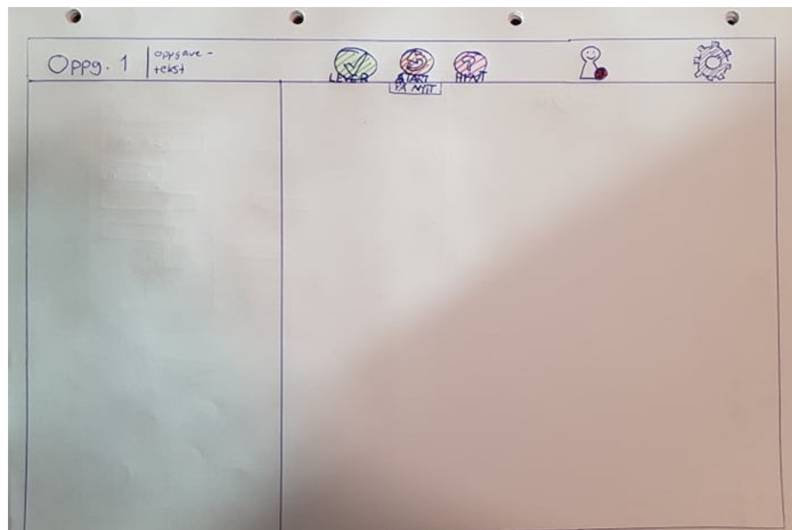
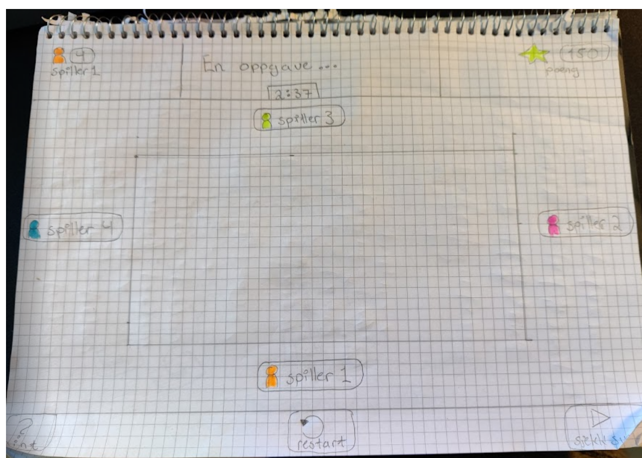
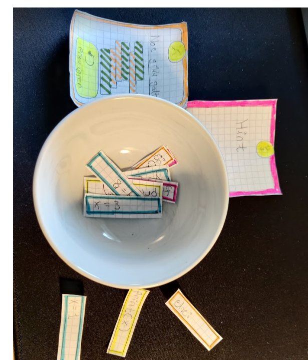


Figure 10: Design A: a simple and streamlined design.



(a) The solutionfield in the center with surrounding player icons



(b) Code blocks

Figure 11: Design B: a board game design.

3.1.3 Development

The development phase consisted of the design process of the low-fidelity prototypes. The prototypes were sketched and refined on paper. After some discussion, we chose a design based on the feedback from our user test participants.

3.1.4 User testing

We invited five students to user test the prototypes. These participants consisted of both well-experienced- and novice programmers. The tests were done digitally due to the Covid-19 situation. The goal of this user test was to get feedback on each design. This feedback would be crucial as it would dictate the design for future iterations. First, the participants were presented with prototype A (the simple and streamlined version) and then prototype B (the game-like version). An interview guide was created for this user test. The interview guide is presented below:

1. Ask what they initially thought the designs were supposed to convey in terms of functionality and use.

2. Explain the prototype in more detail and ask them to solve a simple Parsons problem.
3. Ask a set of questions regarding each design.
4. Ask which design they preferred and why.

The user tests suggested that the participants preferred prototype B. To quote one of the participants: “version B seems more game-like and fun”. In addition, prototype B showcased the multiplayer aspect more clearly. However, prototype A seemed more intuitive, according to the user testers. We also gathered feedback on other elements of the designs, such as buttons, whether other players’ code blocks should be visible, and interactions of the code blocks themselves. See appendix B.1 for a summary of the feedback from this user test.

3.2 Second iteration

In the second iteration, we used the feedback from the first prototype to get a better understanding of the users’ needs. Based on the feedback, we drafted an iterated version of the prototype using a digital design tool called Figma [75]. This new prototype would be used as part of the second user test to gather more feedback on the design.

3.2.1 Requirement specification

We started the second iteration by discussing the feedback from iteration one. This discussion led to a thorough requirement elicitation phase where we created scenarios, use cases, sequence diagrams, and a domain model. From these artifacts, we elicited functional and non-functional requirements. This process covered the ’definition’ activity from design science, see chapter 2.6. We categorized these requirements into four priority groups, where 1 is the highest priority, and 4 is the lowest priority. Functionality relating to moving code blocks was considered to be of the highest priority. See appendix A for the final version of the requirement specification. The following list shows the categories we defined to help prioritize the requirements. Figure 9 shows when TPP entered a new phase; when all requirements from the previous category were completed.

1. **Prototype** : Core functionality necessary to interact with blocks.
2. **Minimum Viable Product (MVP)**: Functionality for students to join games and solve tasks together. MVP is a term used for describing the minimal functionality a product can have and still be used. [76]
3. **Minimum Lovable Product (MLP)**: Functionality that improves the user experience but are not crucial for solving tasks. MLP is a term used for describing the minimal functionality a product can have and still be loved by the users. [76]
4. **Nice to have**: Functionality that further improves the user experience.

3.2.2 Design

After discussing the feedback from the user tests in iteration one, we agreed to continue with the board game design of prototype B, see figure 11. We then sketched a new design that improved on aspects of the initial version. We recreated this design in Figma, an interface design tool, see figure 3.2.2. In this snapshot, two blocks are moved into the solutionfield by player three. This design is a refined version of the board game design utilizing colors, grouping of buttons, and centering of relevant interface elements for problem-solving. See appendix C.3 for additional figures demonstrating other functionality.

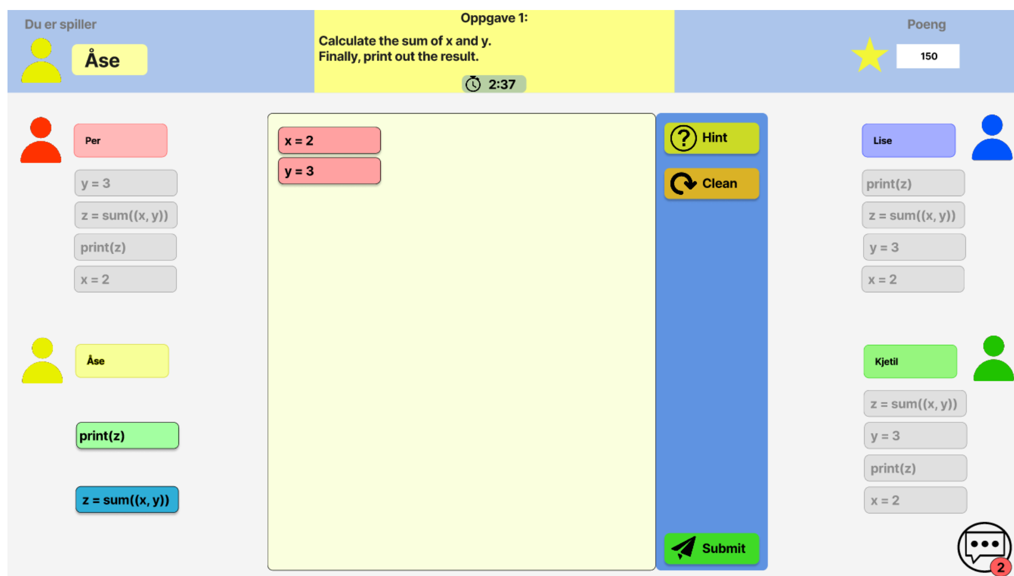


Figure 12: Design created in Figma in iteration two.

There were a few changes in this design compared to the previous version. First, the solutionfield now extends to the bottom of the screen to allow longer code sequences. We moved the player icons and their respective code blocks to the left and right sides. Additionally, we placed some of the buttons in a sidebar to the right of the solutionfield. The reason for these changes was that we wanted the solutionfield to be the main focus of the interface. Our goal was to make it unambiguous where the players should focus their attention and where the code blocks should go. This design decision, among others, attempts to follow principle three of Universal Design: making the user interface intuitive and straightforward to use, see chapter 2.7. This design also features a refined topbar. The topbar displays the player name, the player's assigned color, the current problem description, the remaining time for the current task, and the number of points the team has accumulated. The problem description and timer were placed in the center as we wanted to highlight the elements we deemed important for the players. We moved the team name and points to the side, which are not as relevant during problem-solving.

Lastly, this design also featured colors. The solutionfield and problem description fields received a light yellow color to make them stand out against the grey background and blue topbar. We made this design choice to direct the players' focus toward the center. Blue was used on the top-and sidebar as we felt it took on a more calm look. The code blocks also featured a color based on their code category; for example, red indicating variable declaration. We also decided to grey out the other players' code blocks to convey that the player cannot interact with them. A player should only be able to move their own code blocks into the solutionfield as we wanted to encourage communication between players in order to avoid one player taking control. The buttons on the sidebar received colors and icons. For example, the submit button features a paper plane icon to symbolize sending something, which is frequently featured in mail applications. Having the button be green also signals a sense of "finality" as the players submit the solution for assessment. On the other hand, the clean button, which clears the board of code blocks, effectively erasing the task's progress, received an orange color to signal that pressing the button has an irreversible effect. Finally, these buttons were moved to opposite ends of the sidebar to prevent a player from accidentally clearing the board when trying to submit. This acts as a *tolerance for error*, relating to Universal Design principle 5; see chapter 2.7.

We also consulted Buchinger and Hounsell's 19 design principles for this design. The following list shows the relevant guidelines at this point in the design:

- **Synchronization**: The game shall be *synchronous* as we want quicker gameplay and feedback for the players. We assumed *turn-based* gameplay would make the game too slow and frustrating.
- **Roles**: The guideline states that if the content is important, players should have the same role. Therefore we designed TPP to not assign different roles.

- **Resources**: The code blocks will act as resources to stimulate cooperative and competitive behaviors. TPP shall distribute blocks to each player to further enhance this effect as every player has to contribute.
- **Score**: TPP shall provide the teams with scores based on their performance. According to the guideline, this can be used to discourage player error.
- **Challenge**: The tasks need to provide the right amount of challenge to be motivating. This is also an important point in hard fun; see chapter 2.11.1.
- **Number of players**: According to the guideline, the team size should be flexible. TPP shall therefore be designed to allow 1 to 4 players.

Lastly, Zhang's design principles were also consulted during this phase. The following list shows what design principles we attempted to follow:

- **Support autonomy & Promote representation of self-identity**: The design shall feature icons and nicknames to allow for some amount of self-expression. For now, the same player icon is reused with different colors to separate them.
- **Provide timely and positive feedback**: Task submission will provide a modal window with feedback that highlights correct and incorrect code block placements.
- **Facilitate human-human interaction and represent human social bond**: These principles will be present through moving code blocks as a team to solve a task. The teams will also communicate with each other, either in person or digitally.
- **Induce intended emotions via initial exposure/interaction to/with technology**: Principles 9 and 10 should be fulfilled through an attractive and simple design, and the game shall give a positive user experience.

3.2.3 Development

After reviewing the requirements and the user test feedback, we proceeded to work on the updated design. We started by discussing possible changes to design B, the preferred look by the participants. After sketching a rough design on paper, we went on to re-create it digitally. With Figma, we created multiple pages of the interface, which would simulate an interactive interface. The prototype allowed users to interact with various elements such as the sidebar buttons, the chat, and the code blocks. We simulated the buttons and the chat with static overlays; to illustrate how they would work in an actual game. The drag functionality was simulated by creating one page for each dragged code block in the solutionfield. In addition, we added a transition effect to mimic the drag movement. This interactive interface would become the prototype for the second user test. We also developed an interview guide.

3.2.4 User testing

The goal for this user test was to get more feedback on the interface design. In this iteration, we wanted participants that represented our target group, see chapter 1.6. The Covid-19 situation was not as severe at this point, so we decided to perform the user test in person. We sought out a programming exercise class with novice programming students. Five students were asked to perform user tests, including one- and second-year computer science students. Each participant received a movie ticket as a reward for participating.

This user test followed a similar procedure as the first:

1. Ask what they think the game is supposed to be.
2. Explain the prototype in more detail and ask them to solve a simple Parsons problem; while talking aloud about their thought process as they performed each action.

3. Ask a set of questions regarding the design.

The feedback made it apparent that it was not entirely clear what kind of game it was. A few participants thought that the game's objective was to compete with the other players. It was also not apparent if the game was supposed to be turn-based. Additionally, we got feedback that the colors made the interface look "child-like". However, they thought the colors for the clean, hint, and submit buttons were fitting. In the end, everyone were able to solve the Parsons problem with little issue. A few attempted to click on the buttons instead of dragging them. They then expressed that the game should also allow users to click on blocks to move them. We should emphasize that the code block interaction in this prototype was limited as there was a set order the blocks could be moved in. However, this was not a big concern as we primarily wanted feedback on the interface design. We also received feedback on other aspects of the game, such as how the lobby system should work, leader boards, rewards, other modes, and communication preferences. The interview guide and feedback summary can be found in appendix B.2.

The following list shows a few of the main takeaways from this user test:

1. Update the color scheme to be more appealing to students.
2. Allow both clicking and dragging to move the code blocks.
3. Players shall be able to invite others through an invite link.
4. Include a skippable tutorial so players can get a quick overview of controls and the objective.

3.3 Third iteration

In the third iteration, we started the software development process. The software development process relates to the 'development of artifacts'; the last part of activity three in the design science methodology, see chapter 2.6. We used the prioritization table found in appendix A when implementing new features. The third iteration includes 'evaluation' and 'communications'; activities five and six in design science.

This iteration started with a roughly 1-month experimentation phase. After we had decided on technology and frameworks, we sought out libraries that could facilitate drag-and-drop functionality and communication between players. This experimentation phase also allowed us to try out different solutions on a smaller scale before making any big commitments. Next, we started developing the first fully playable version of TPP. We were able to cover all the requirements from the first two priority categories.

3.3.1 Requirement specification

Following the user test from iteration two, we reviewed our requirement specification. The feedback from our participants led to an updated version with new functionality, such as more expansive code block interactivity, better handling of game invitations, and interface changes to make the cooperative aspect more apparent. We reviewed our requirement prioritization table to decide what to include in this iteration. In order to get TPP to a playable state, we had to cover all the requirements from the first two categories. See appendix A for the final version of the requirement list.

The following requirements were prioritized in this iteration:

- Implement drag-and-drop functionality for code blocks.
- Implement multiplayer to allow for cooperation.
- Implement a simple lobby system to initiate games.
- Allow indentation in the solutionfield.
- Players shall be able to solve a task set.
- Provide feedback on task submission.

3.3.2 Design

This chapter is split into two: the user interface design and the technical design. The latter includes frameworks, libraries, and architecture. For more figures showcasing the user interface, see appendix C.4.

User interface

The user interface did not change drastically from the last iteration, with the layout remaining consistent. However, the color scheme got addressed due to the feedback from the second user test. See figure 13 and 14 for the mockup and final implementation of the improved design. Firstly, the interface received a blue and white-grey scheme. As the interface took on a more relaxed look, the code blocks now stood out more as they maintained their bright background colors based on their code category. In addition, code blocks now featured an outline with a color based on the player who owns it. We decided that we wanted to enhance each player's feeling of contribution as the team cooperates to create the correct solution. The solutionfield also featured a grey rectangle shape to indicate where code blocks can be placed. The implemented design also featured a thin line at the left of a code block to show the start of a new indent. With indentations, we could facilitate *student-scaffolded* Parsons problems, see chapter 2.2, which we believed would work well for Python due to the importance of correct indentation.

Secondly, Universal Design was applied to ensure the design was accessible to as many people as possible, see chapter 2.7. Furthermore, we took measures to avoid entirely black and white backgrounds and applied a 4.5:1 ratio or higher between background and foreground elements. We also reused colors to promote a more consistent look. An example of this was to reuse the green and orange colors of the code blocks for the clear and submit buttons. Lastly, we tried to make the interface work on a wide arrange of screen sizes, see figure 15. The interface adapts to the size of the screen by stretching and moving interface elements around to preserve a functional layout.

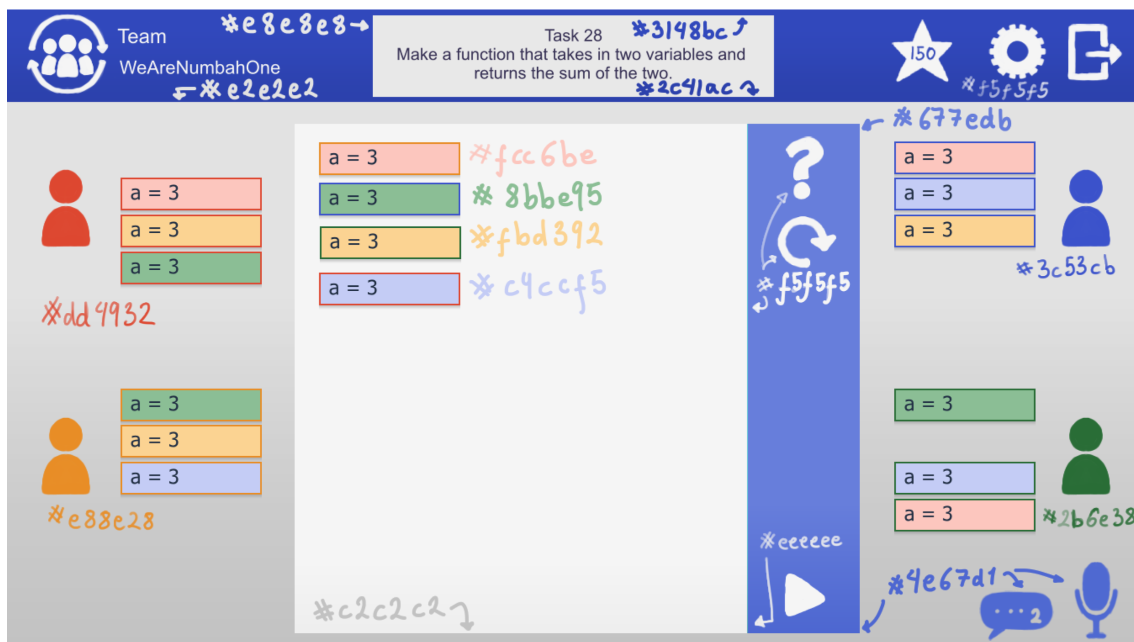


Figure 13: Mockup of the user interface in iteration three with improved color scheme and contrast.

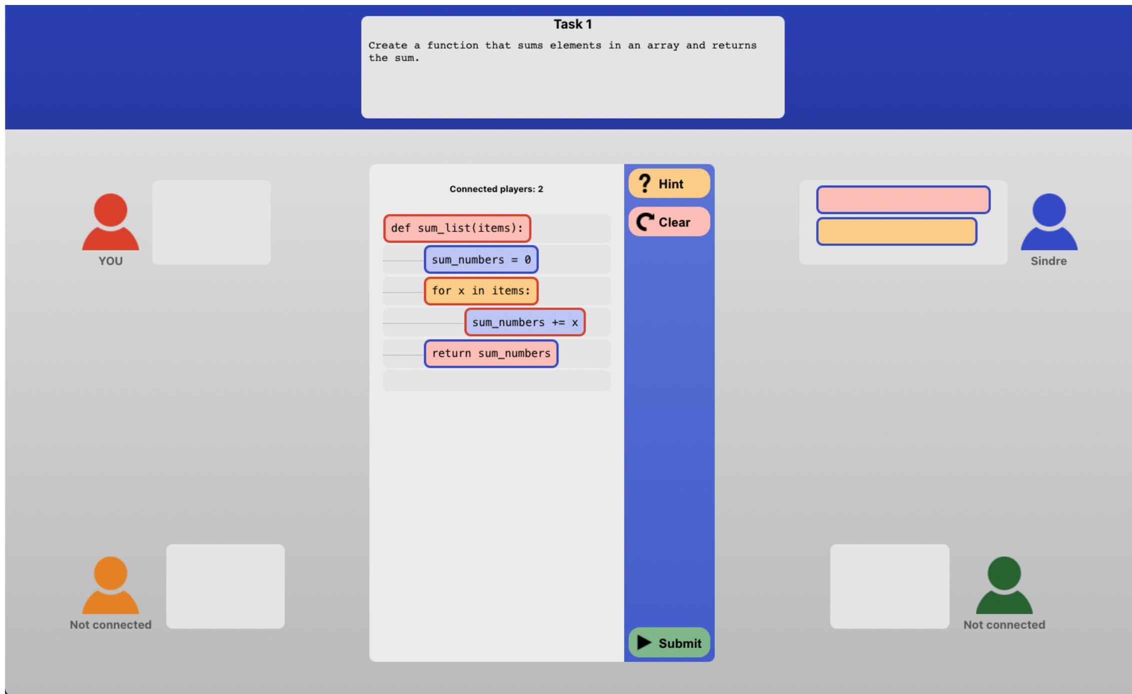
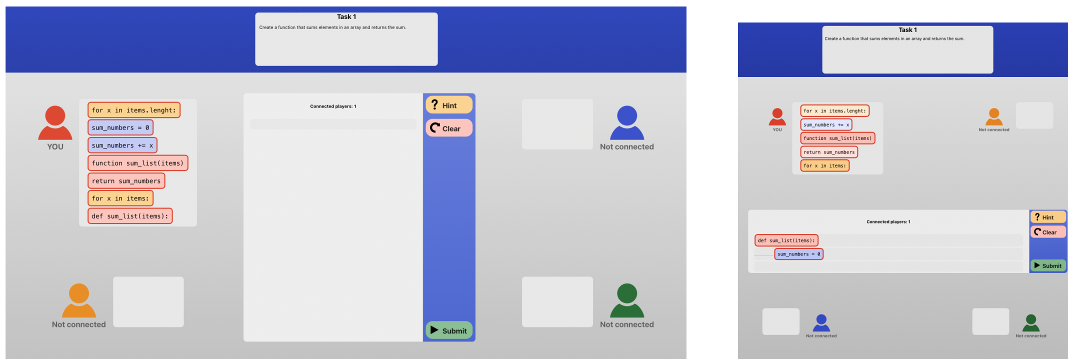


Figure 14: The implemented interface in iteration three with the improved design.



(a) Traditional horizontal layout where the screen width can fit players on the right and left.

(b) Adapted vertical layout if the screen width is too narrow.

Figure 15: Adaptive interface layout based on screen size.

Technical

We decided to develop TPP in *React*, a JavaScript library for developing user interfaces and web applications. One of the main reasons for this decision was that React seemed to have some existing solutions for real-time communication and drag-and-drop functionality. After the one-month experimentation phase, we decided on TPP's code architecture. See figure 16 for the React component tree diagram. This figure shows how we would structure our React components. We defined three key areas for our architecture:

- **Communication**: These components have the role of managing messages to and from other players.
- **Topbar**: These components have the role of displaying information about the game, the game options, and the left button.

- **Game**: These components handle how the game is played. Functionality for drag-and-drop, hints, clearing the board, and task submission are implemented in these components.

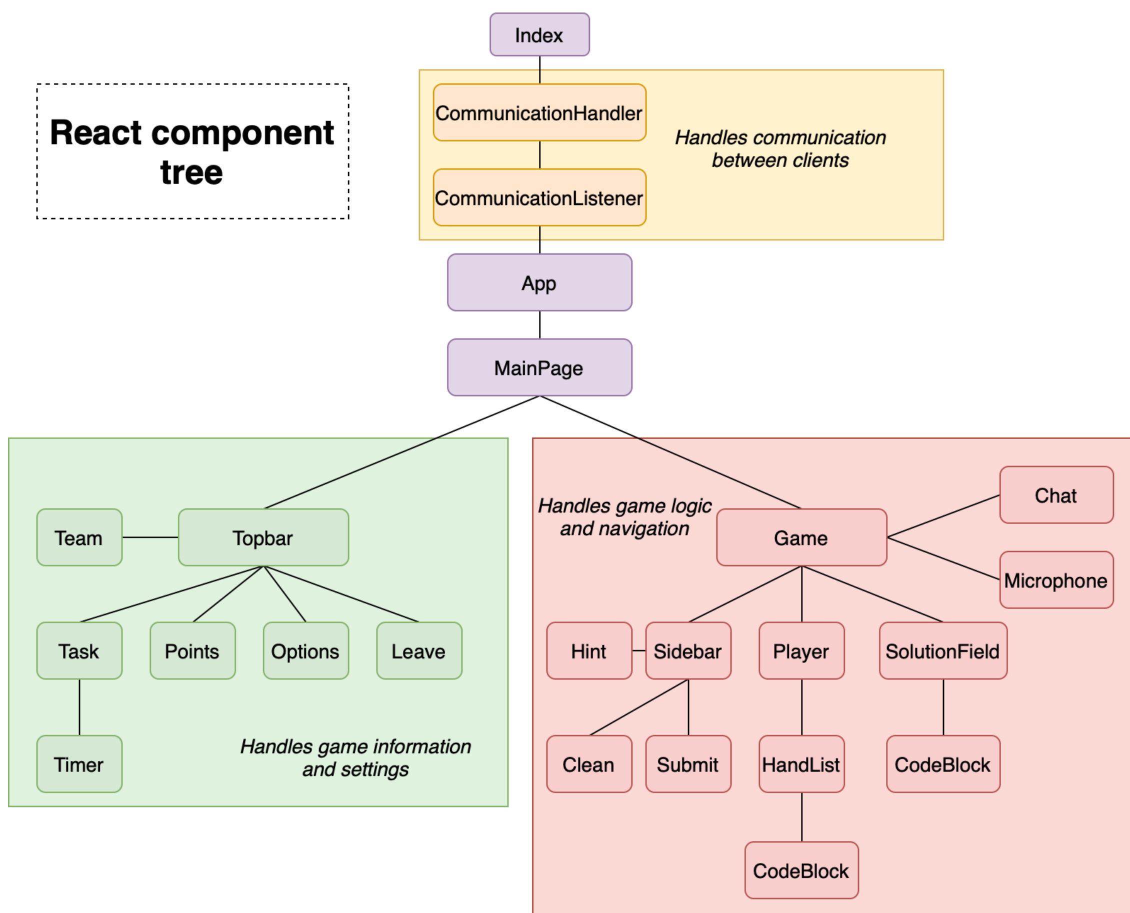


Figure 16: The React component hierarchy model in iteration three.

An important concept in React applications is *stateful* and *stateless* components. A *state* is a built-in React object which contains data or information about a component. Changes in the state cause the component to re-render and update the interface to account for user action or other changes. A principle of React is *data down*, which means that data is passed down as properties to child components. When a change is detected in a component, the updated data is passed down, which causes child components to re-render. However, we needed the option to pass data across components for our implementation. For example, when a block is moved from a player to the solutionfield, the components under the Game category are re-rendered to update the user interface. This change must also be broadcasted to the other players, which means that the data must be able to move up to the Communication components. Additionally, the topbar must be updated when a new task is initiated.

We decided to use *React-Redux*, a state management library that synchronizes applications [77]. Redux stores all component states into a single repository, the *Store*, which can make state management easier as applications grow in size. In order to update a state, a component will *dispatch* an *action* to the *reducer*. Dispatching an action represents an event that has occurred, which updates the store. The reducer manages the event and returns the updated state. In TPP, all stateful components use Redux. For example, when a player moves a block from their hand to the solutionfield, Redux ensures the game stays synchronized. Moving a block from the hand to the solutionfield requires both the HandList and the SolutionField components to be updated. Additionally, this information must be broadcasted to the other players, and Redux facilitates this. Redux will also track when it receives game state updates from other players and vice versa. We argued that this design would ensure a synchronized game state for all players.

During the experimentation phase, a few drag-and-drop libraries were considered. Many of these librar-

ies were designed to be used in list management applications. We wanted to use a library that allowed us to utilize the drag-and-drop functionality while also being flexible enough to be used in TPP. There were four libraries considered:

1. Beautiful React Dnd (BRD) [78].
2. React Grid Layout (RGL) [79].
3. Js-parsons [50].
4. React Dnd (RD)[80].

The first library we looked at was BRD, which was quick to set up and had good immediate results. However, the API was difficult to modify when trying to implement indentation. RGL was another library considered, but similarly to BRD; it was difficult to modify to our needs. Next, we looked at Js-parsons, which we reviewed in more detail in chapter 2.3.1. This library provided many features, including both drag-and-drop and various feedback options. However, after reviewing the source code, we deemed it too time-consuming to modify Js-parsons. Finally, we chose RD as it provided a simple API to create components that could take on the role of both draggable and droppable objects. Even though we had to develop more from scratch, the library proved to be flexible enough for us to implement what we had envisioned.

The communication between the players was implemented using the LioWebRTC Javascript library [81]. This library builds upon WebRTC, which provides APIs to facilitate peer-to-peer communication in the browser [64]. Even though servers benefit from being able to store and access game session information, we opted for the peer-to-peer architecture. We argued that a peer-to-peer implementation could provide a lower latency, which would result in smoother and more responsive gameplay.

TPP uses LioWebRTC to inform all peers when someone performs actions in the game. For instance, when a player moves a code block, the game will *broadcast* this event to all the other peers in the network, which was established before the game started. This is done by informing the receiver about what event is being broadcasted, including the necessary data to update the game state. Table 5 shows the communication protocol in iteration 3.

Event	Description	Data
Set List	Notify other peers that someone moved a block in or out of the handlist.	2D array with the updated handlists.
Set Field	Notify other peers that someone moved a block in or out of the solutionfield.	Array with the updated solutionfield.
Next Task	Notify other peers that someone initiated a new task. This should update the game state for everyone.	The current task number.
Clear Board	Notify other peers that someone cleared the board. This should move all code blocks from the solutionfield to their respective players.	-
Start Game	Notify other peers that someone started the game. This should direct the players from the lobby to the game page.	-

Table 5: Communication protocol in iteration three.

WebRTC needs to be facilitated with signaling in order to establish a connection for peer-to-peer communication, see chapter 2.8. Thus we had to set up a signaling server that TPP could connect to. For this, we opted for SignalBuddy [82], a scaleable socket.io server for WebRTC, NodeJS cluster, and Redis. Since SignalBuddy was developed by the same creator of LioWebRTC and was used as part of the LioWebRTC demo, it seemed like an appropriate choice. As for hosting the signaling server, we had a few alternatives: Netlify, Heroku, and Amazon Web Services (AWS) [83, 84, 85]. We were already hosting

TPP on Netflify, but according to their terms of service, we were not allowed to refer to a Netlify application within another. We also attempted to host it on Heroku, but the performance was unreliable. This led us to host the signaling server on AWS running on a Linux platform. This solution worked well and provided a solution for TPP to establish peer-to-peer connections.

3.3.3 Development

The user test-from iteration two led to an updated requirement specification. First and foremost, we had to decide what framework and language to use for developing TPP. We ultimately decided on React, as it was well documented, had many resources, and seemed to have some existing solutions for the drag-and-drop functionality and communication. Next, we agreed to set aside one month to explore some of these solutions. The experimentation phase was beneficial as we could try out various solutions and technologies before making any big commitments. Additionally, we learned more about React and how to develop more expansive applications using it.

After the experimentation phase, we started developing the first playable version of TPP. We wanted to follow test-driven development, so we decided on a test framework for React, including Jest and React Testing Library. This meant that components were primarily built following the test-driven approach by first writing test cases according to our requirements and then developing the code to pass these tests. However, due to some challenges with testing drag-and-drop functionality, we could only create tests for displaying correct information on rendering.

In order to get an overview of what components we needed for TPP, we sketched our design on paper and labeled what elements would work as components, see figure 17. The development of these components was primarily built from the bottom-up as we would not need to mock child elements compared to if we had started top-down. See figure 16 for the React component hierarchy. Later, some changes required refactoring, specifically how drag-and-drop worked. Initially, the code blocks acted as both draggable- and droppable objects, but later we separated these. Line-based feedback was also implemented for when a player submitted a solution, see chapter 2.2.3. The communication was implemented using WebRTC, and a communication protocol was established as development advanced, see chapter 2.8.1. A continuous integration pipeline was also implemented early to build and test TPP automatically. Whenever TPP was updated, deployment was done manually by uploading the build to Netlify, a cloud-based web application hosting service [83]. We used Git as our version control system for managing the source code.

As the interface, core functionality, and communication reached a more definitive state, we started planning for the user test. As part of this user test, we had to write a handful of tasks that the participants were to solve. To help create these tasks, we decided to create a simple task-creation tool in React. See chapter 3.5 for more information about the tool. This tool created tasks in a JSON file format that we coordinated with TPP to be compatible. At this point, TPP had most of its requirements fulfilled from chapter 3.3.1. Lastly, a signaling server was remaining. We utilized SignalBuddy [82] for this purpose and hosted it on Amazon Web Services.

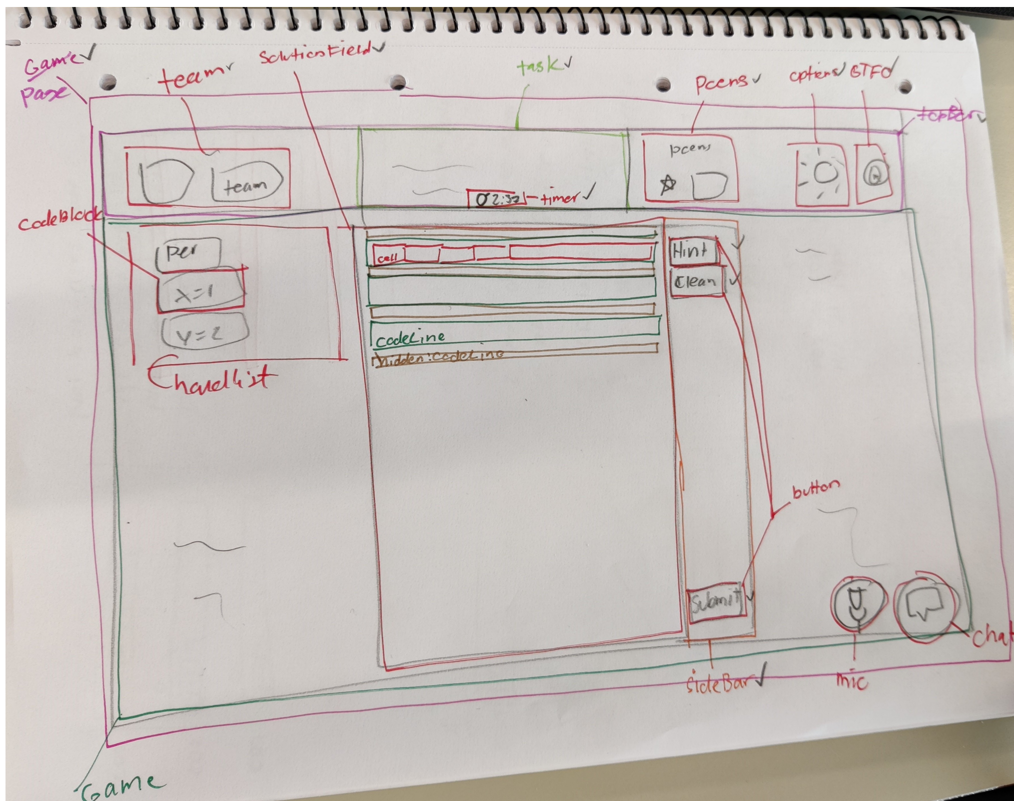


Figure 17: Initial sketch of the React components using the user interface design from iteration two.

Before the user test, we decided to perform a technical test to ensure TPP was working as intended. We invited two classmates from our study program to perform the test. During this test, we encountered a race condition bug. If two players moved a block into the same code line simultaneously, and overwrite could occur, resulting in a block either disappearing or being duplicated. This was a game-breaking bug as the players would not be able to solve the task. The bug had been discovered earlier during a stress test, but we did not anticipate it to occur during normal play, as we underestimated the number of simultaneous actions that could occur with more players. To relieve this bug, before the user test, we decided to limit the network traffic by not sending data as long as the player kept moving pieces. This created some delay, which resulted in less smooth gameplay. Additionally, we modified the 'clear' action to make it restart a task if the bug occurred during the user tests.

3.3.4 User testing

We created a test guide at the end of development phase three. Similar to the previous user test, we wanted to perform a usability test to identify design issues and better understand our target group's behaviors and preferences. Therefore, we asked two groups of four students to participate in the user test. In addition, each participant would receive a gift card for the school cafeteria. The test was performed in person on campus.

The user test used the following structure:

1. Welcome the participants.
2. Perform a pre-test questionnaire to obtain some basic information, such as their experience with programming and whether they preferred to work in groups.
3. Explain TPP and its controls.
4. Ask the participants to play a game using TPP.

5. Perform a post-test questionnaire to gather their thoughts on TPP.

6. Have a group discussion with all participants to obtain more feedback and ask questions relating to the communication and cooperation aspects of the game.

The feedback gave us the impression that they liked the cooperative and interactive elements. The results showed high scores when asked if the game was fun to use and if they liked to work together, see appendix B.3. When asked if the game was fun to use on a scale from 1 to 5, 75% answered 4 or higher. When asked if they enjoyed working in a team on a scale from 1 to 7, 87,5% answered 6 or higher. While we only tested two groups, this gave an impression that the game could be a fun and social activity. However, we noticed differences in group communication and their approaches for solving tasks. We had the impression that a participant in group A took on a leader role. The leader read the task description aloud and reviewed the solution, with the other team members, before submitting it. This group had a systematic approach where members would inform each other if they had a code block they thought could fit the solution. On the other hand, group B had no clear leader. We experienced this group as more silent, as the players would initially spend more time looking at their own blocks before trying to solve the task. As the test went on, they adopted an approach where everyone moved their code blocks into the solutionfield to make all the blocks visible. They then started eliminating the blocks they thought did not fit the solution.

In the post-test and group discussion, we learned that group A liked that they could not see each other's blocks. They argued that it benefited cooperation, similarly to our own argumentation for making them invisible. On the other hand, Group B disliked this design decision and would prefer if all the blocks were visible from the start. Moving all the blocks into the solutionfield would act as a workaround, with one participant arguing that: "getting an overview of all of our options helped solve the task". These groups had opposing opinions on this design choice, and we would have to decide how to handle this in the next iteration. Overall, both groups seemed to agree that the overall user interface design was good. According to the post-test questionnaire, the minimalistic design, layout, and colors made the interface "easy to understand" and sayintuitive to use. Besides a few issues like interface elements disappearing off-screen on smaller displays, and confusion about what the coloring of the code blocks was supposed to convey, we deemed the design mostly successful.

The race condition bug from the technical test only occurred with group B. We believe this happened since group A had a more systematic and a slower approach overall when solving the tasks. This resulted in less simultaneous interactions, decreasing the chance of the bug occurring. Since group B coordinated less when moving blocks, the chances of simultaneous interactions increased; resulting in the bug happening a few times during the user test. This reaffirmed that this bug was a high-priority issue for the next iteration. Otherwise, we got some feedback that the interface appeared a bit jittery when blocks were randomly moved around. We got some feedback that there could be more indication of what blocks are being interacted with; to add clarity for who is doing what. Both groups also agreed that everyone should get feedback when someone submits the solution. A few participants also mentioned the idea of a voting system where all players have to agree when submitting the task. Lastly, we got feedback that a few participants forgot that certain functionality existed during the test; including hints, double-clicking and tabbing. We argued that the quick introduction to the game was the main contributor to this as they did not have much time to get familiar with the interface and controls before play.

The questionnaires and feedback summary can be found in appendix B.3. Here are a few of the main takeaways from this user test that would be addressed in iteration four:

1. Address the race condition bug.

2. Decide on whether other players' blocks should be invisible or not.

3. Give all players feedback on task submission.

4. Consider having a voting system before submitting a task.

5. Provide more visual feedback on what blocks are being interacted with and by whom.

6. Address some interface issues like font size and screen size compatibility.

3.4 Fourth iteration

In the fourth iteration, we refined TPP based on feedback from the previous user test. We experienced a bug with how TPP handled game state updates when there were a lot of simultaneous code block interactions. Solving this bug would become a top priority for this iteration. We had also received some feedback that it was difficult to keep track of who was doing what during play. Therefore, we opted to implement *player-indication*, based on which blocks the player was interacting with; within the solutionfield. Additionally, as suggested by our participants, we decided to implement a new system for task submission. With this system, all players would have to *lock in* the solution in order to submit it.

3.4.1 Requirement specification

Following the user tests from iteration three, we made a summary of all the feedback we received, see chapter 3.3.4. Due to time constraints, we knew that the development phase had to be reduced to ensure we that had enough time to perform the final user test. The short time frame led us to prioritize game-breaking bugs as well as address some of the other received feedback. For example, one test group requested a voting system for submitting a solution. We liked this idea and deemed it a high priority; since this feature also would address feedback from the other group, which expressed that it was unclear when another team member submitted a task. Additionally, some participants requested more visual feedback on code block interaction by other team members. Several participants expressed that they were confused when several players simultaneously interacted with the code blocks. Another important issue we would address in this iteration was a bug where blocks would occasionally disappear or multiply. This bug resulted from a race condition; caused by multiple simultaneous code block interactions in the same position, which could overwrite the game state.

After reviewing the feedback, we decided to prioritize these main requirements for the final iteration:

1. Rework communication to address the race condition bug.
2. Add a voting system for submitting solutions.
3. Provide more visual feedback when team members interact with code blocks.
4. Address screen-size compatibility and other minor user interface problems.

3.4.2 Design

A design flaw from the previous iteration was that every peer updated the game state locally before broadcasting it to the network. We had intended to remove the need for an authoritarian server to reduce game lag. However, it introduced a race condition. If two messages were broadcasted in a short time interval, the second message could overwrite the first. If this were to happen, blocks could completely disappear from the game, and the task would be unsolvable. After researching how other peer-to-peer games handled communication, we found that a common solution was to set one of the peers as the *host*. See chapter 2.8.2 for more information on the topic.

To implement this, we had to re-route network traffic through a host, which in our case was decided to be the peer that started the game. If the host disconnects during a game, the first peer in the player order is set as the new host. Once the game has started, each peer who is not the host will send a *move request* to the host instead of performing each move locally. The host will check if the move is valid, update the game state locally, and finally broadcast the updated game state to all peers. See figure 18 for a simple illustration of how this works. Unfortunately, this solution creates some delay for peers who are not hosting. However, the host will experience little to no delay when interacting with the game.

We decided to implement a preview to address the delay introduced for non-host peers. The preview provides the players with instant feedback when moving blocks without affecting the game; see figure 19. The host overwrites the local previews to ensure synchronization. When implementing the previews, we faced a problem where players would receive their old move requests, resulting in code blocks skipping

back and forth. To solve this, we chose to block incoming move requests that appeared right after a local move request, see the alt fragment in figure 19. Alternatively, we could have blocked move requests originating from the receiving peer, but we had no way of knowing who sent the message in the current design.

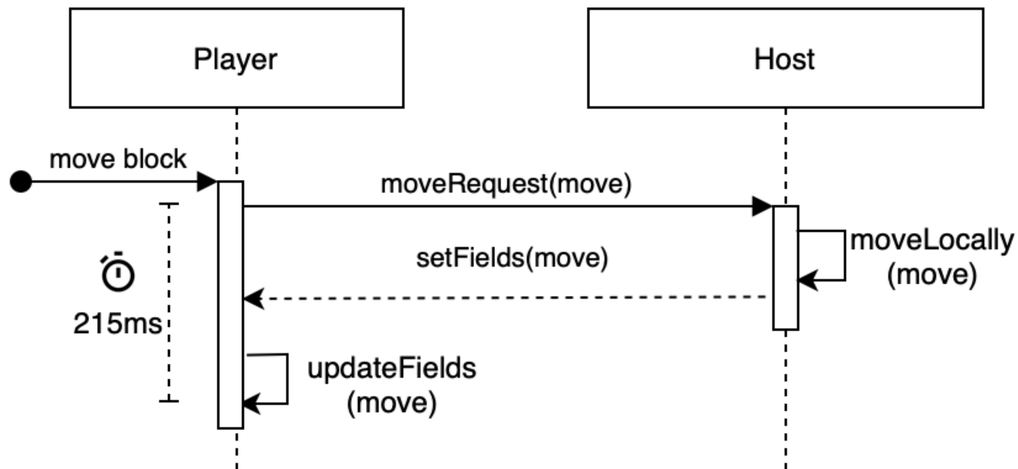


Figure 18: Sequence diagram of the communication using the player-as-host design. Peers request moves to the host, which performs it locally and finally broadcasts the updated game state to all peers.

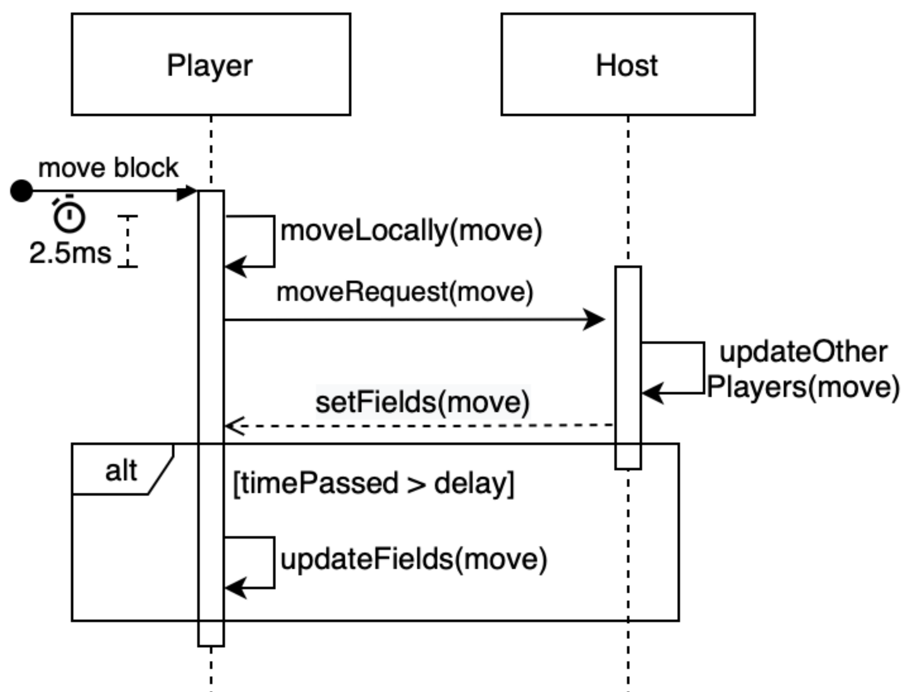


Figure 19: Sequence diagram of the communication using previews. Move request with preview to provide quicker feedback when moving blocks

Based on feedback from the user tests in the third iteration, we decided to add features to highlight actions performed by other team members. To address this, we decided to add tiny speech bubbles with player icons, see figure 21. These symbols are later referred to as *player indicators*. A player indicator pops up when another player moves a code block in the solutionfield.

We also decided to add a voting system for task submission, as suggested by the user testers. We implemented this by replacing the 'Submit' button with a 'Lock in' button. When a player *locks in* the board, they cannot perform additional moves. See figure 22 and 23 for screenshots of this implementation. The answer is submitted when all players are locked in. We decided to reuse the player indicators for the voting system.

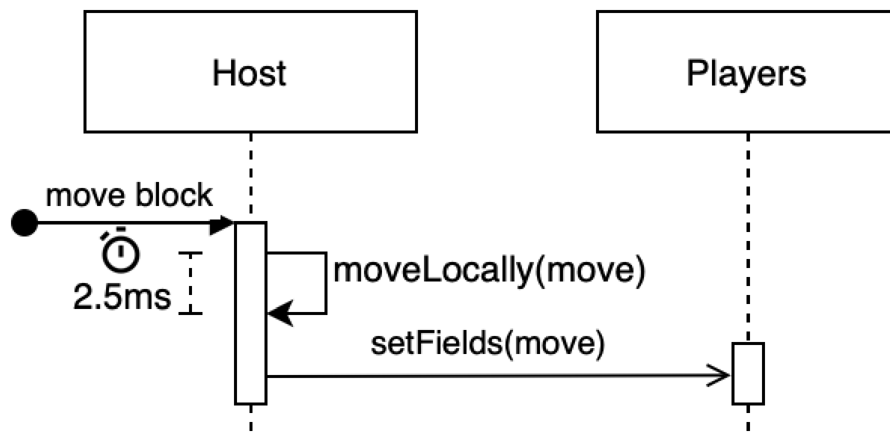


Figure 20: Sequence diagram of the communication from a host to a non-host peer. The host moves locally and updates the other peers game state

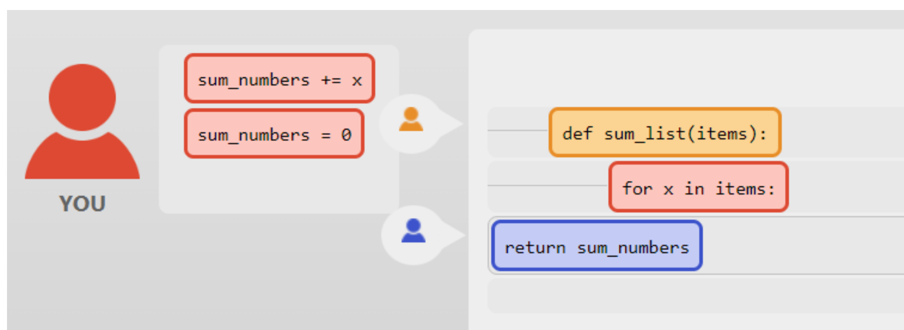


Figure 21: Code block interactions are indicated with speech bubbles and player icons.

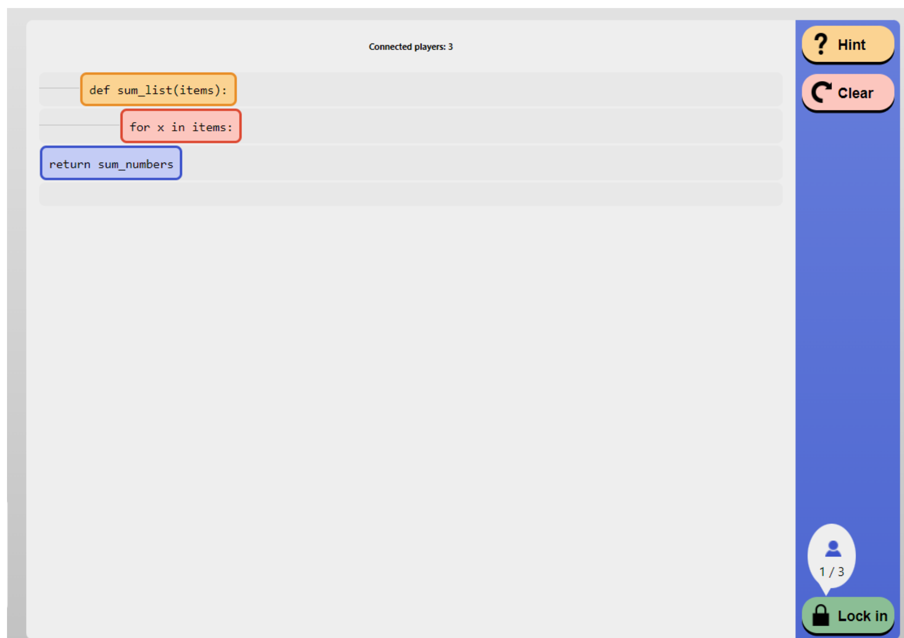


Figure 22: The lock-in indicator shows that the blue player is *locked in* and ready to submit the answer.

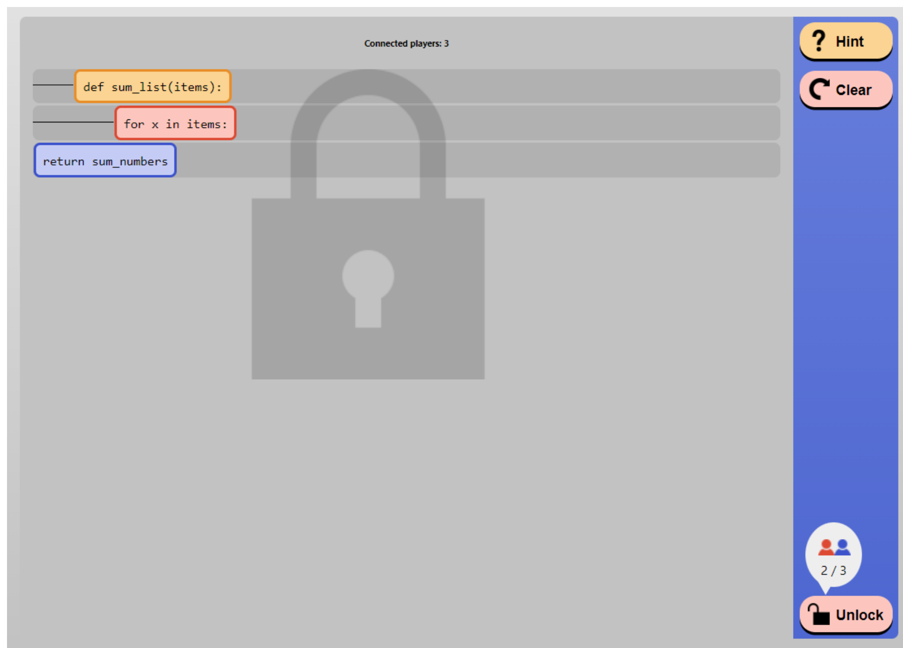


Figure 23: The game interface after a player is *locked in*. The player cannot perform additional moves.

3.4.3 Development

We started the fourth development phase by addressing the race condition bug from the previous iteration. After doing some research on peer-to-peer games, we learned that assigning one peer as the host, which acts as an authoritarian server during play, can solve similar synchronization issues, see chapter 2.8.2. With this in mind, we sketched some diagrams and proceeded to implement the *player as host* design. In our new implementation, all communication went through the host, see figure 18. Following the new communication flow we sketched out and implemented the player indicators and the voting system, see figures 21, 22, and 23. We also used the *react-responsive* library to address screen size compatibility [86].

By adding these new features, our performance worsened. We measured that a non-host player would experience a delay of approximately 800 ms between a move and an updated board. We considered this unacceptable, as this could negatively impact playability and cooperation. The codebase also grew in the fourth iteration, which impacted maintainability. With this in mind, we started to optimize and refactor our code. We used the React Profiler to assist us with this [87]. The Profiler calculated the time spent by each React component, which indicated where we should focus our efforts. Optimization did improve performance, but it was not enough to affect the player experience. To target player experience, we opted to implement a local preview, see figure 19.

3.4.4 User testing

In the fourth user test, we focused on gathering data for our research questions. Five groups of first-year programming students were recruited to participate in this user test. In order to get quantifiable data on TPP's usability, we opted for the System Usability Scale, see chapter 2.9. The 10 statements were included in the post-test questionnaire, and the score would be calculated after all the tests were done. We also decided to perform a thematic analysis, see chapter 2.10, to help us interpret our data. Two open-ended questions relating to research questions 1 and 2 were included in the post-test questionnaire. Chapter 3.6 explains how we performed the thematic analysis. See appendix B.4 to for the complete questionnaires and the group discussion questions.

The user test itself followed a similar structure as iteration three:

1. Welcome the participants and briefly explain our thesis.
2. Perform a pre-test questionnaire to gather basic information about their programming experience.
3. Ask participants to enter a TPP lobby. Then, have them watch a tutorial video to learn about the controls and the objective of TPP.
4. Ask participants to solve a task set. The test will stop if the participants solve the task set or 15 minutes have passed.
5. Perform a post-test questionnaire.
6. Have a group discussion to gather more data regarding learning, cooperation, and TPP.

The results from this user test can be found in chapter 4 and appendix B.4. Chapter 5.1 discusses our findings.

3.5 Task-creation tool

During iteration three, we created Parsons problems tasks for internal testing and the upcoming user test. With this in mind, we decided to develop a tool for creating tasks and task sets. The development of this tool followed a similar process as a single iteration. However, we did not test this tool on external users as it was developed for internal use only. The format for a task can be seen in table 6. Task sets combine multiple tasks into the same file. The following requirements were specified for the task-creation tool:

1. The interface shall feature a text editor to allow the user to write and paste code.
2. The user shall be able to include distractors.
3. The interface shall feature a task description field.
4. The interface shall feature a hint field where the user can add or remove up to five hints.
5. The interface shall feature an ’attempts’ field where the user can set the number of attempts the players have to solve the task.
6. The user shall be able to opt for setting an infinite amount of attempts.
7. The code blocks shall be categorized based on function and use, for example, variable declaration and loops.
8. The task shall be stored in a JSON file format when saved.
9. The task set creation page shall give feedback if the file format is incorrect when selecting tasks.

After creating a simple mockup on paper, we developed the tool in React. The interface is split into two down the middle. The left side features the text editor, whereas the right side features the fields that must be filled in. See figure 25 for the implemented design. By grouping the fields on the right side, the user first writes code on the left before specifying task settings. In order to not clutter up the interface, the user can add and remove hint text fields by clicking the ’+’ and ’-’ buttons. The ’save’ and ’cancel’ buttons received a green and grey color, respectively, to make it easier to distinguish the buttons.

We decided to have the user write directly into the code editor when adding a distractor. See chapter 2.2.1 for an explanation of distractors. We considered having fields for these, but writing them into the code editor would make it easier for the user to see where they could fit. For example, if the code has a line that says ’if a < b:,’ and the user wants to add a distractor that changes the ’<’ to ’>,’ they could add a distractor prior to the correct line. The user can mark a line as a distractor by prepending the characters ’#$’. We chose to use a special character to indicate the line being a distractor; in this case, ’$’. However,

```
"codeBlocks": [
  {
    "code": "def remove_chars(word, n):",
    "category": "function",
    "indent": 0,
    "id": "1"
  },
  {
    "code": "x = word[n:]",
    "category": "variable",
    "indent": 1,
    "id": "2"
  },
  {
    "code": "return x",
    "category": "function",
    "indent": 1,
    "id": "3"
  }
],
"distractors": [
  {
    "code": "function remove_chars(word, n)",
    "category": "function",
    "indent": 0,
    "id": "4"
  },
  {
    "code": "x.pop(n)",
    "category": "function",
    "indent": 0,
    "id": "5"
  }
],
"description": "Write a function to remove characters from a string starting from zero up to n and return a new string.",
"hints": [
  "remove_chars(\"pynative\", 2) should result in output native.",
  "Use string slicing to get the substring. For example, to remove the first four characters use s[4:].",
  ": is the delimiter of the slice syntax to 'slice out' sub-parts in sequences."
],
"attempts": "unlimited",
"field": []
```

Figure 24: Task format example.

if the user were to prefer writing and running the code in their preferred IDE or environment to test, this would throw a syntax error upon running. To solve this issue, we made it so that the line has to start with '#', which turns the line into a comment. See figure 26 for an example of what distractors look like in the interface.

From a technical perspective, the tool utilizes a library called 'react-simple-code-editor' [88]. This JavaScript library creates a code editor with basic syntax highlighting. When the user saves a task, the tool splits the code into code lines. These lines are filtered to remove blanks and comments. This process also separates legitimate code blocks and distractors. The indentation was calculated from the number of spaces used in the code editor. Additionally, the code blocks were categorized using regular expressions. These expressions were tested with unit tests to cover various ways of expressing a certain code category. The task data is downloaded to the user's computer as a JSON file after the filtering process. Tables 6 and 7 shows the format for a Parsons problem task and for a code block, respectively.

Property	Description	Type
codeBlocks	The code blocks that are part of the solution.	Array w/code block objects
distractors	The code blocks that are not part of the solution.	Array w/code block objects
description	The task description.	String
hints	The hints for the tasks.	Array w/Strings
attempts	The amount of submit attempts for the current task.	Integer or "unlimited"
fields	The code blocks that are part of the solutionfield.	Array w/code block objects

Table 6: JSON file format for a Parsons problem task.

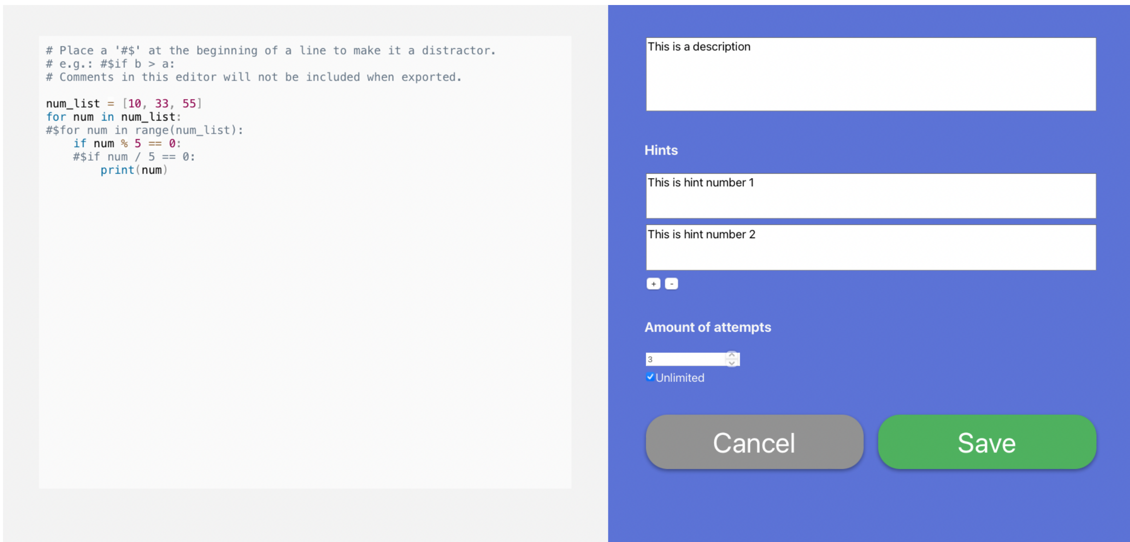


Figure 25: The create-task interface features a code editor and fields to set information and properties for the task.

```
# Place a '#$' at the beginning of a line to make it a distractor.
# e.g.: #$if b > a:
# Comments in this editor will not be included when exported.

num_list = [10, 33, 55]
for num in num_list:
#$for num in range(num_list):
    if num % 5 == 0:
    #$if num / 5 == 0:
        print(num)
```

Figure 26: Distractors are indicated in the create-task interface by using the prefix '#$'. A comment is added at the top of the editor to remind the user of how to include distractors.

Property	Description	Type
code	The content of the code block.	String
category	The category of the block (ex: operator or variable).	String
indent	The correct indentation for the code block.	Integer
id	The unique identifier for the code block.	String

Table 7: A code block's class properties.

The task set creation page can be accessed from the select menu, see figure 27. The page consists of an interface that allows users to upload tasks from their computer. The tool will check if the format is correct by reassuring that the files contain the correct properties, see table 6. Tasks with an invalid format will be discarded, and the user will get an alert popup. When one or more tasks are selected, the user can combine them into a task set by clicking the green button. This creates a new JSON file that is exported and downloaded to the user's computer.

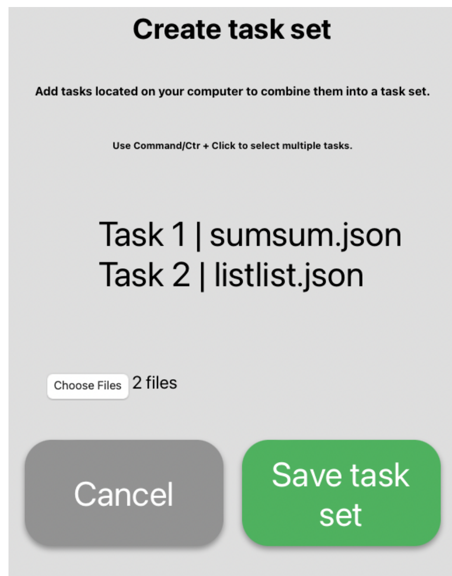


Figure 27: Interface for selecting tasks to be combined into a task set.

3.6 Thematic analysis

In order to answer our research questions, we looked at ways to analyze qualitative data from our user test surveys. After doing some research, we found thematic analysis (TA), see chapter 2.10, to be cited as an accessible form of analysis for qualitative research, particularly for those unfamiliar with the various analysis methods [70, p.81]. We followed TA using the step-by-step guide created by Braun and Clarke, see table 2 [70]. The data set consisted of answers from two open-ended questions in our post-test questionnaire during the final user test, see chapter 3.4.4. There were 15 participants in total, which resulted in 15 answers per question. We decided to perform TA on each question separately.

We decided to take a *deductive approach* as we already had our research questions defined prior to the analysis. We also had a theoretical framework from our literature review, see chapter 2. A theoretical framework would allow us to examine the data with some preconceived themes in mind relating to learning, socialization, cooperation, and gamification. Lastly, we chose to take a *semantic approach* when interpreting the data. We argued that the subtext of the participant's answers would not be relevant for answering our research questions. The semantic approach seemed fitting as we were only interested in their stated opinions.

Firstly, we familiarised ourselves with the entire data set. We read and re-read the answers to both questions during this phase while taking notes. Next, we started generating *codes* for question one. We systematically went through each answer by highlighting sections of text that we thought expressed some idea or feeling. We allowed text sections to be coded multiple times if we thought they expressed more than one idea or feeling. Once we had coded all the data, we went over the answers again to ensure that we did not miss anything of interest. We also looked for redundant codes that could be merged into one. When we were satisfied with our codes, we created a new table and extracted all the highlighted texts sorted by code. According to Braun and Clarke, context can easily be lost in the coding process [70, p. 89], and so we made sure to include some surrounding text in the extracts if we felt it added necessary context. We also noted how many times a code occurred in the data.

Next, we searched for potential *themes* by collating related codes. We started this process by writing the codes onto post-it notes and creating a mind-map. Related codes were placed close to each other on the map. If any groups of codes emerged, we would consider if we could create a theme that encapsulated them accurately. We also allowed codes to be present in multiple themes if we thought they fit. When we were satisfied with our initial themes, we went on to the next phase. We reviewed our themes by systematically checking if a theme worked in relation to the coded extracts. For example, if a theme named 'teamwork' included a code-named 'enforce cooperation,' we would look at all its extracts and see if 'teamwork' accurately represented the data. If it did not, we would consider the following: 1) could

the extract fit under a different code? 2) does the code fit within the theme? 3) is the theme too broad? If we thought there were issues with the theme, we would consider splitting it up. We also considered if the themes were distinct enough to avoid any overlap. When we were done reviewing the themes, we moved on to phase 5. This phase consisted of defining and naming the themes. We would refine the specifics of the theme by generating clear definitions and names for each one. When we were satisfied with the themes for question one, we repeated the process for question two. Finally, we completed the last phase by summarising the results in our thesis, see chapter 4.1.2 and 4.2.2. We discuss our findings in chapter 5.1.

4 Results

In this chapter, we present our findings; based on feedback and observations from our user tests. The first four subchapters are structured after our four research questions. The findings from the first two research questions (RQ) include a poll regarding whether the participants agree with a set of statements relating to fun, enjoyment, and learning outcome. In addition, we performed a thematic analysis (TA) for research questions 1 and 2. Next, we present our findings on how a team works together to solve cooperative Parsons problems in a digital space; as a set of characteristics based on our observations. To answer how a cooperate multi-player Parsons problems game can be designed, we present our final version of TPP. First, we present the user interfaces and features included in TPP. Secondly, we explain our architecture. Finally, we present an in-game performance test. At the end of this chapter, we present other findings, which we consider relevant for the discussion in chapter 5.

4.1 RQ1: Can cooperative Parsons problems be a fun and social activity for novice programming students?

This chapter presents our findings on whether cooperative Parsons problems can be a fun and social activity for novice programming students. We present the post-questionnaire polls and the results from our thematic analysis of the question: *How do you think that this game could impact socialization among students?*

4.1.1 Post-questionnaire polls

In the final user test, we asked the participants if they agreed with three statements; relating to enjoyment and socialization. These questions were asked as a part of the post-test questionnaire. The questions and their corresponding answers are shown in figure 28, 29 and 30. First, we asked whether the participants thought the game was fun. Secondly, we asked if the participants would have preferred to play the game alone; to compare cooperative and non-cooperative Parsons problems. Finally, we asked if the participants thought that the game could improve socialization among students.

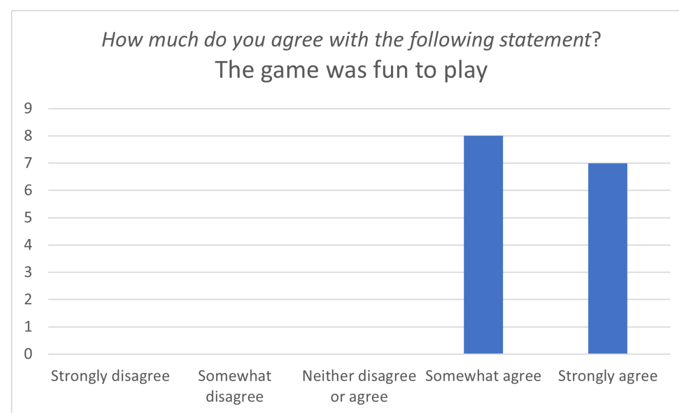


Figure 28: Bar chart of the responses to Q1

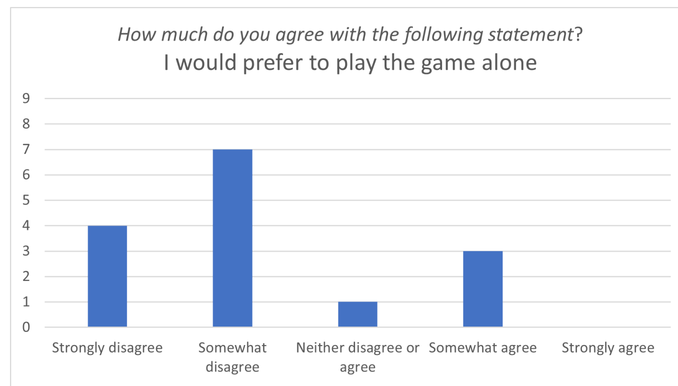


Figure 29: Bar chart of the responses to Q2

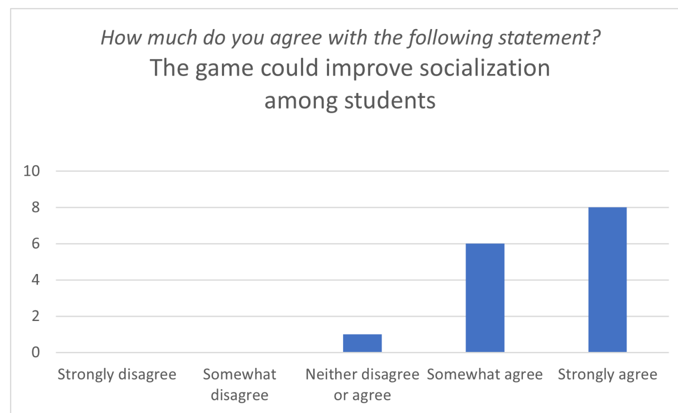


Figure 30: Bar chart of the responses to Q3

4.1.2 TA1: How do you think that this game could impact socialization among students?

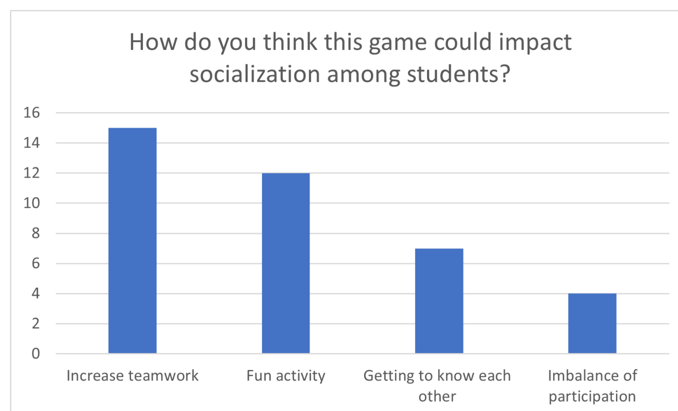


Figure 31: Bar chart of the responses to Q4

In the fourth user test, we asked 15 participants the following question in the post-test questionnaire: *How do you think that this game could impact socialization among students?* The participants answered the question in text. We then used these text-based answers to perform a thematic analysis. The identified themes from the analysis are shown in figure 31. We identified four major themes: *increased teamwork, fun activity, getting to know each other,* and *imbalance of participation.* The y-axis represents how frequently a theme was mentioned. Note that the same participant might have mentioned a theme multiple times.

We define *teamwork* as the process of working together to achieve a goal. The goal, in this case, is to solve Parsons problems. In contrast, we define *getting to know each other* to be socialization without an external goal. A majority of the participants mentioned that the game was fun; in a question about socialization. The *fun activity* and *getting to know each other* themes were often found together. Some participants also mentioned the risk of *an imbalance of participation*. We will have a closer look at this in chapter 4.3, where we answer the research question *how does a team work together to solve cooperative Parsons problems in a digital space?* Below, we included quotes from the user test and their related themes.

> 'Right now programming is something you do alone, and I think this is a great way of breaking that bubble.' — Participant, theme: *increase teamwork*.

> '[This game] forces students to talk to each other. Students has to work in a team setting to solve different tasks.' — Participant, theme: *increase teamwork*.

> '[This game] might increase team work between data/informatics students' — Participant, theme: *increase teamwork*.

> '[This game] It makes learning much more fun and social' — Participant, themes: *fun activity* and *getting to know each other*.

> '[This game] It is a conversation starter and a fun way to get students to engage socially' — Participant, themes: *fun activity* and *getting to know each other*.

> 'I can imagine an instance where one person commands/leads the group while the others play more of a secondary role. In this case, I believe socialization may be minimal since the non-leader students would not really be as engaged.' — Participant, theme: *imbalance of participation*.

4.2 RQ2: What are the players' perceptions of the potential learning outcome from solving cooperative Parsons problems?

This chapter presents our findings on the participants' perception of TPP's potential learning outcome. We present the post-questionnaire polls and the results from our thematic analysis of the question: *How do you think that this game could impact socialization among students?*

4.2.1 Post-questionnaire polls

In the final user test, we asked the participants if they agreed with four statements; relating to learning outcomes. These questions were asked as a part of the post-test questionnaire. The questions and their corresponding answers are shown in figure 32, 33, 34 and 35. We wanted to determine whether the participants thought that the game could be beneficial for learning. First, we asked whether the participants agreed that the game could test newly learned programming concepts. We wanted to ensure that the game could affirm or refute that a player understood a particular topic. Secondly, we asked if the game can be used to discover weak areas in their programming knowledge. We wanted to know if the game could clear up misunderstandings. Thirdly, we asked if the game can create discussion about programming concepts. We were interested in this since discussions can increase transactivity, see chapter 2.4.2, which is also tied to improved learning outcomes [26, p. 352]. Finally, we asked if the game can strengthen existing programming knowledge. We wanted to know if the game could function as a repetition activity.

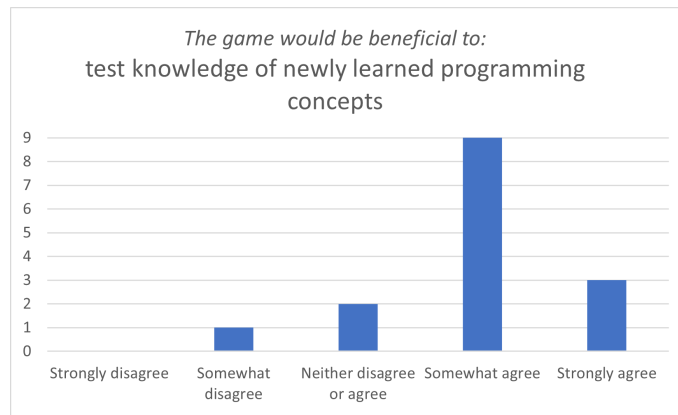


Figure 32: Bar chart of the responses to Q5

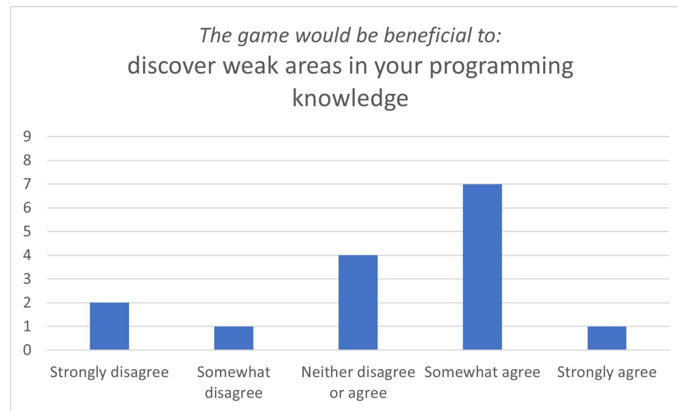


Figure 33: Bar chart of the responses to Q6

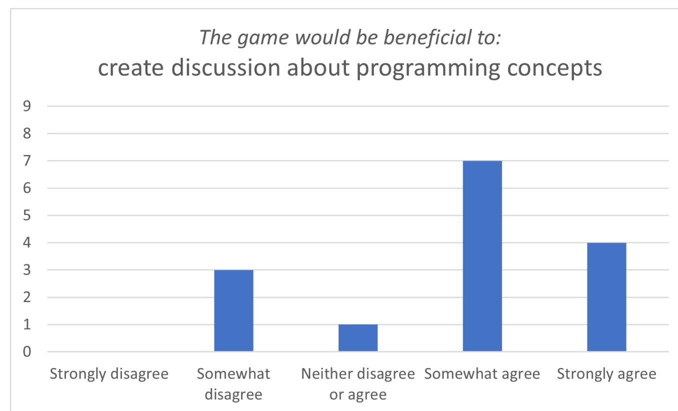


Figure 34: Bar chart of the responses to Q7

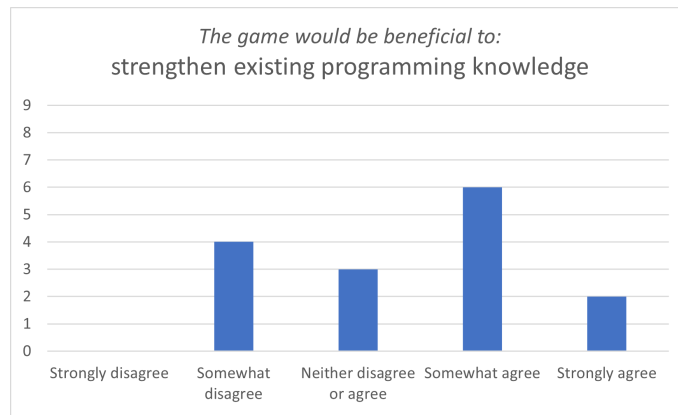


Figure 35: Bar chart of the responses to Q8

4.2.2 TA2 : What are your perceptions on potential learning outcomes from this game?

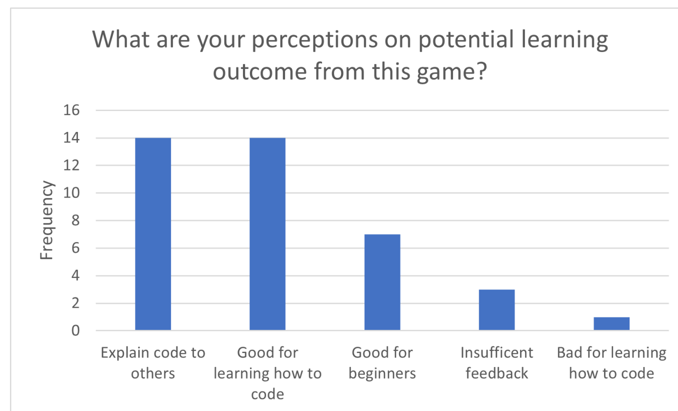


Figure 36: Bar chart of the responses to Q9

In the fourth user test, we asked 15 participants the following question in the post-test questionnaire: *What are your perceptions on potential learning outcomes from this game?* The participants answered the question in text. We then used these text-based answers to perform a thematic analysis. The identified themes from the analysis are shown in figure 36. We identified 4 major themes: *explain code to others*, *good for learning how to code*, *good for beginners*, and *insufficient feedback*. We also included the theme *bad for learning how to code* to show some nuance. The y-axis represents how frequently a theme was mentioned. Note that the same participant might have mentioned a theme multiple times.

Many participants mentioned that it would be beneficial if the entire team understood the problem before they started solving it. We placed these answers within the theme *explain code to others*. Most of the participants wrote that this game was good for learning how to code, especially for "beginners." However, one of the participants wrote that this game was *bad for learning how to code*, stating that: “I don’t think this is the best way to learn to program”. Some of the participants mentioned that we could improve learning outcomes by changing how the game provided feedback. Below, we included quotes from the user test and their related themes.

> ’I believe it [this game] may encourage discussion between inexperienced programmers. This may be a result of having to compound knowledge in order to completely understand the problem’ — Participant, theme: *explain code to others.*

> ’The [game offers the] opportunity to see how other students think.’ — Participant, theme: *explain code to others.*

'Another learning outcome could come from talking about code more thoroughly, in order to make sure everyone understands and agrees.' — Participant, theme: *explain code to others.*

'In this game, we were limited to the handout code, which means that we had to solve the exercise in a way we normally would not. I think that this game could expose players to different ways of coding.' — Participant, theme: *good for learning how to code* (this quote was translated from Norwegian to English).

'The learning outcomes are huge for beginner levels' — Participant, themes: *good for learning how to code* and *for beginners.*

'It could definitely be used to improve general programming ability, although more complex concepts could be more difficult to implement — Participant, themes: *good for learning how to code* and *for beginners.*

'I think it would be more educational if there were an explanation after each task detailing why things work and what every line does.' — Participant, theme: *insufficient feedback.*

4.3 RQ3: How does a team work together to solve cooperative Parsons problems in a digital space?

The findings relating to this research question are a combination of observations, group discussions, and post-test questionnaire answers. We included the data from 7 user tests, both from iterations 3 and 4. After reviewing our notes and questionnaire answers, we elicited 11 prevalent characteristics of how a team works together to solve cooperative Parsons problems using TPP. Table 8 presents our findings.

Characteristic		Description
C1	Leader emerging	In 5 out of 7 teams, a clear leader emerged. This participant often took the initiative to read the task description aloud, command where blocks should be placed, request blocks from others, ask if the team wanted to submit their solution, and review the code. The remaining teams had a more even contribution among their members.
C2	Passivity	There would typically be someone who did not participate as much as the others. They would contribute with their code blocks and occasionally participate in discussions, but otherwise, they would take on a more passive role. This was more prevalent in teams where a leader emerged.
C3	Moving all blocks into the solutionfield	In 4 out of 7 teams, a strategy emerged where everyone would move all of their code blocks into the solutionfield. Some teams followed this strategy from the start, whereas others would adopt it as the tasks got more challenging. Through a process of elimination, the teams would remove the blocks they deemed incorrect or irrelevant.
C4	Requesting blocks	Teams that did not take on the previous characteristic (C3) would request code blocks from each other. After they had read the task description, someone, usually the leader, would request various code blocks from their teammates. The requests that we frequently observed were function declarations, loops, variable declarations, and return statements. The others would then respond if they possessed the requested code blocks and subsequently move them into the solutionfield.

C5	Reviewing task description	In many teams, someone would read the task description aloud at the beginning of every task. The leader usually initiated this. Otherwise, the team members would read the task description by themselves until someone requested- or moved code blocks.
C6	Reflection	Most, if not all, teams had moments of reflection about the various programming concepts relevant to the current task they were engaged with. Some examples of reflections were: discussing whether *Python indexes* started at 1 or 0 if they needed to use *nested-for loops*, *reverse*, and *modulo*. Occasionally, this would result in someone explaining the concept to the other unsure members. This would also occur during discussions about how the tasks should be solved.
C7	Harder tasks created discussion	Harder tasks created more discussion compared to the easier tasks. In the final user test, the first three tasks were relatively easy. This resulted in minimal discussion as the teams were able to solve the tasks quickly. We observed that task 4 took a while longer to solve and created more discussion due to its complexity.
C8	Trial-and-error	During the more challenging tasks, some teams started to rely more on trial-and-error when submitting their solutions. One team, in particular, asked if they were punished with a lower score if they repeatedly submitted the task. When we assured them that their performance was not measured, they became less reluctant to submit. Especially during task 4, we observed that they would submit more frequently and use the feedback to assess whether they were improving their solution.
C9	Reviewing code collectively	A majority of the teams reviewed the code line-by-line when they were stuck on a task or before submitting. The leader would usually initiate this process. We observed that two teams, in particular, would review their solution by themselves and remain silent when they were stuck on a task.
C10	Lagging behind	During the team discussions, we were made aware that there were instances where some team members would start solving the task before everyone was done reviewing the task description. One participant would usually be reading or attempt to understand the task while the others were discussing or moving code blocks.
C11	Competitiveness	A majority of the teams expressed that they felt competitive during the user tests. They wanted to solve all the tasks within the time limit and beat the other teams' completion times. This was prevalent even after we made it clear that we were not keeping track of any of the teams' performances. They would also show excitement when they solved tasks.

Table 8: Characteristics for how teams work together to solve cooperative Parsons problems using TPP.

4.4 RQ4: How could a cooperative multi-player Parsons problems game be designed?

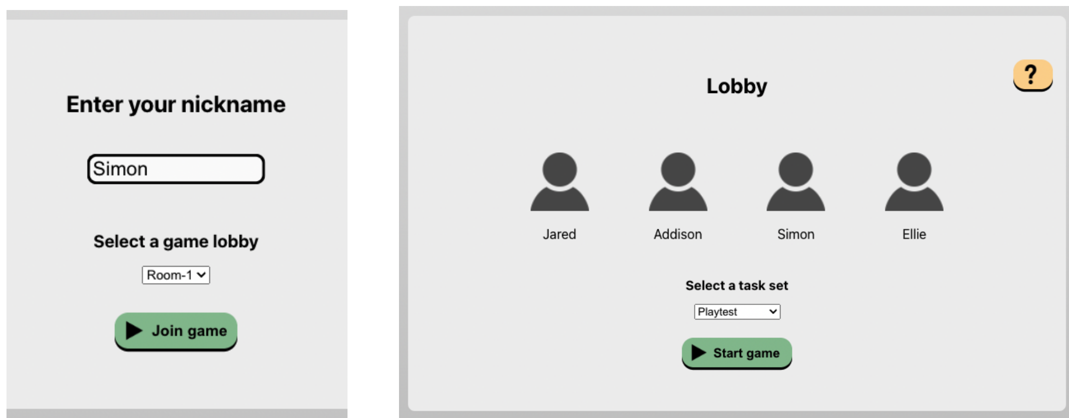
The final version of TPP serves as our suggestion for how a cooperative multi-player Parsons problems game could be designed. First, we will give a walkthrough of the user interface and the implemented features. Then we will present the architecture of TPP by presenting the different components, how TPP manages its states, and finally, how the communication is implemented. Hopefully, by showcasing these various aspects, we will provide some insight into how a cooperative Parsons problems game can be designed.

4.4.1 User interface and features

This chapter will give a walkthrough of TPP's user interface design and features. All player names featured in the following figures are fictive.

Lobby

A lobby was created to allow matchmaking in TPP. When accessing TPP on its hosted domain, the user will be prompted to fill in a nickname and select which game lobby to join from a list. If the nickname field is left empty, the user will receive a nickname based on its own peer id in the network. TPP will matchmake all players who select the same game lobby. However, TPP will alert the user if they try to access a full lobby of four players. See figure 37 (a) for the first screen. TPP will display an animated puzzle GIF while it is connecting. The second screen will display all the players who selected the same game lobby. Here, the user can click the '?' icon to access a tutorial video explaining the game's mechanics and the objective. The users can also select which task-set they want to solve. Any player can start the game when the team is ready by clicking the 'Start game' button. See figure 37 (b) for the lobby screen.



(a) Interface for entering a nickname and selecting what game lobby to join.

(b) The game lobby with four players.

Figure 37: The lobby of TPP.

Game interface

The final version of the game's user interface can be seen in figure 38. This interface consists of a topbar, four player fields, one solutionfield, and one sidebar. Each of these elements will be covered in the following chapters. Overall, this interface aims to direct the user's attention to the solutionfield, as this is where the players will work together to solve the task. The solutionfield and player fields feature a white background as a contrast to the grey backdrop. The sidebar and topbar also feature a blue background to create contrast for the buttons and task description.

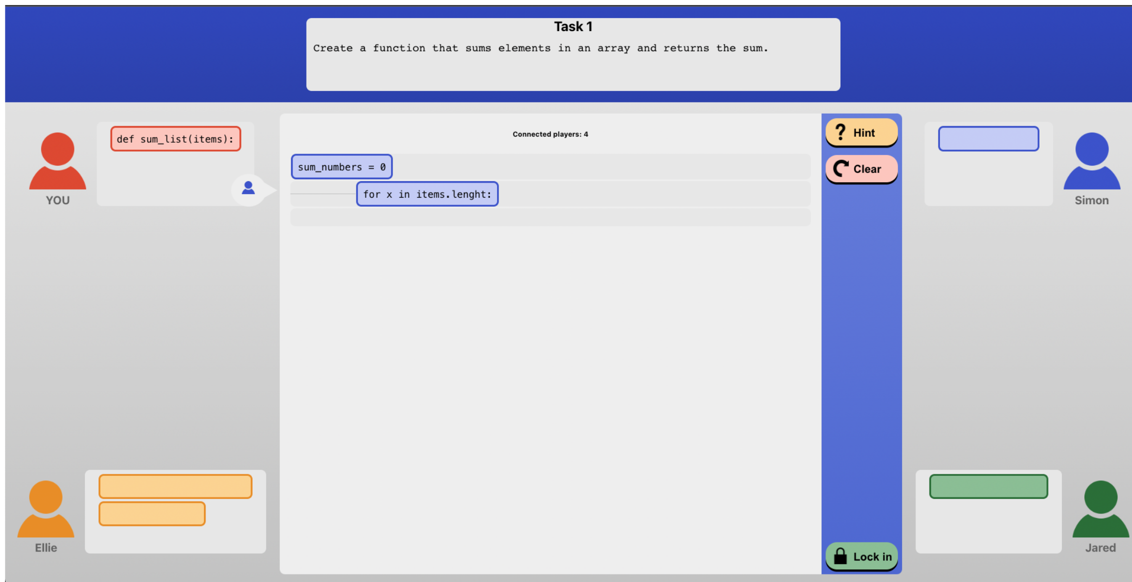


Figure 38: The game's user interface.

Player fields

Each player has their own field, containing their player icon, nickname, and distributed code blocks. The code-block list is also referred to as the 'handlist' to reference the idea that the players are playing a board game. TPP distributes code blocks to each player at the beginning of each task, which only they can interact with when positioned in the player field. Figure 39 shows an instance from the blue player's perspective where only their code blocks are visible. As stated earlier, this design decision was made to promote communication between players and avoid that one player solved the task alone. Finally, each code block features the color of the player who owns it. This was done to strengthen the feeling of contribution as they work together on the solution.

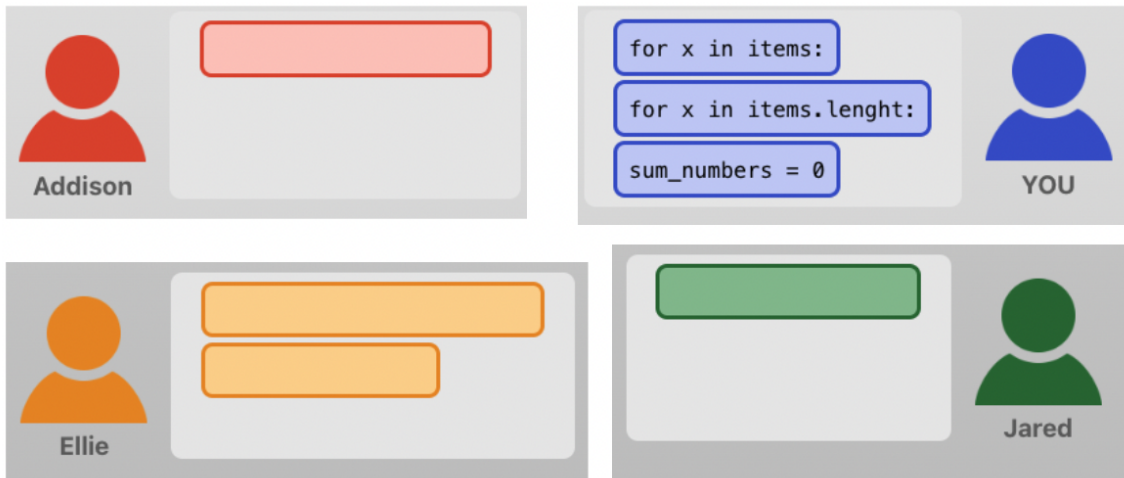


Figure 39: The player fields from the blue player's perspective.

Solutionfield

The solutionfield consists of horizontal lines where the code blocks can be placed. There is an empty line at the bottom of the field to indicate where new blocks can be placed. However, the user can choose to insert a code block anywhere by moving the code block over an occupied line. Figure 40 shows an

instance where the players have completed a task by moving their code blocks into the solutionfield. A pre-pended line to the left of the blocks further indicates indentation.

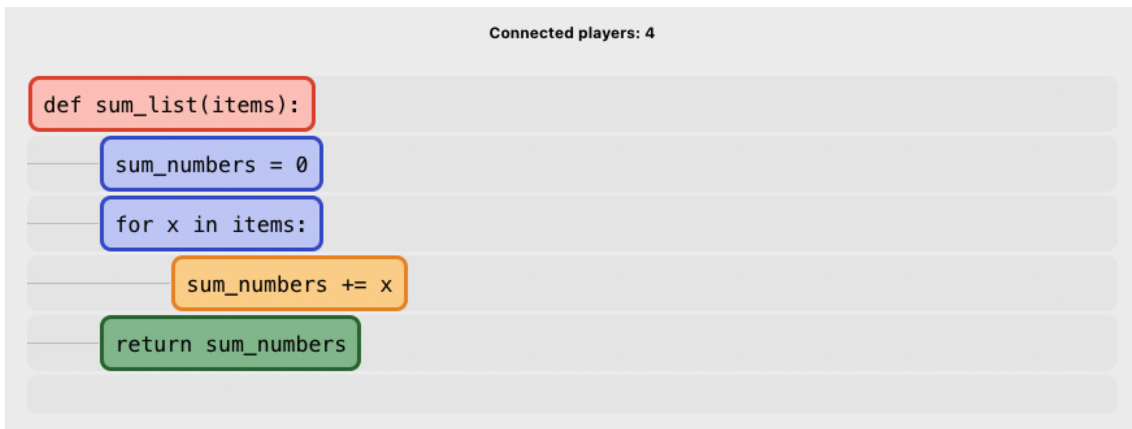


Figure 40: The solutionfield featuring a potential solution.

Code block interactions

The player can interact with code blocks in a few different ways. The player will primarily move blocks by using drag-and-drop, see figure 41. The block can be dragged from the player's handlist field into the solutionfield and vice versa. The player can also drag a block vertically to change the code block order. Another way to move a code block from one field to another is by double-clicking the block. This action will place the block at the bottom of the field it is moved to. A code block can be indented in the solutionfield by dragging it horizontally. Alternatively, the player can select the block by clicking on it and using the tab key. The block can be de-selected by clicking on it again. A player can only interact with code blocks in their own handlist or those placed in the solutionfield. Lastly, visual feedback is used when hovering over draggable and non-draggable code blocks. For example, hovering over a draggable block will change its background color and the cursor to a hand to indicate that it can be picked up. See figure 42 for this implementation.

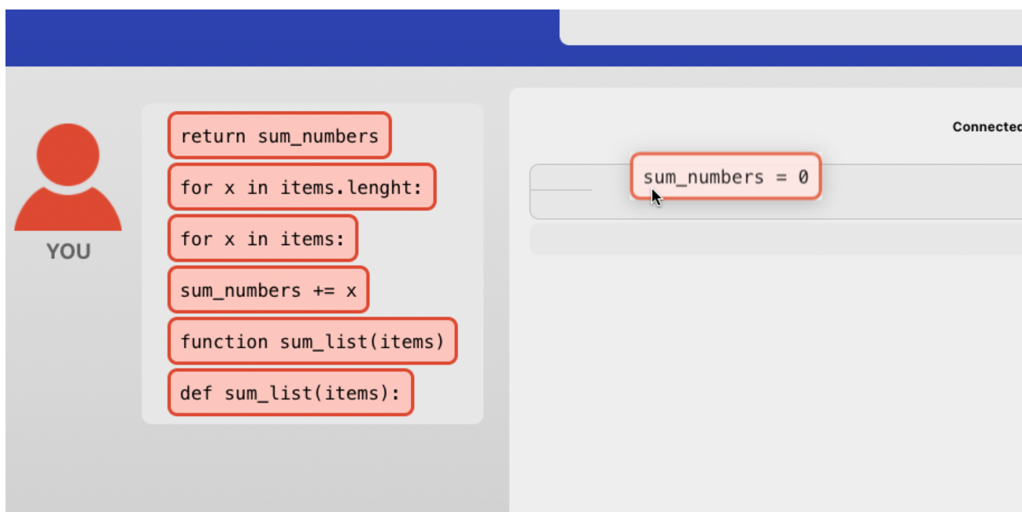
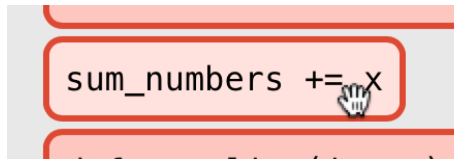
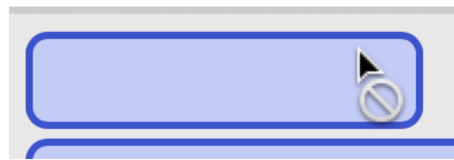


Figure 41: Player 1 is dragging a block into the solutionfield.



(a) Draggable code block.



(b) Non-draggable code block.

Figure 42: Visual feedback when hovering code blocks.

Topbar

The topbar contains the task description, which is updated as the players progress throughout the task set, see figure 43.

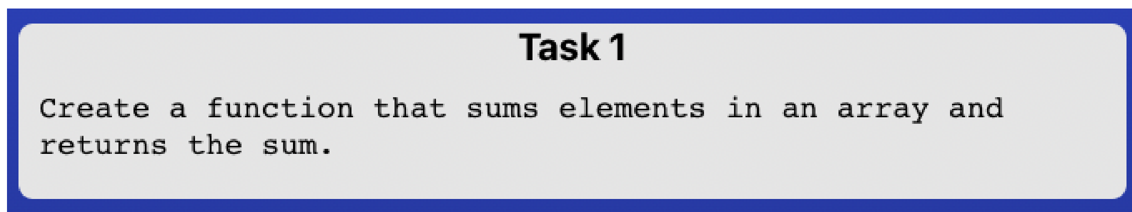


Figure 43: The topbar features the task description.

Sidebar

The sidebar is located to the right of the solutionfield, see figure 44. It contains the hint, clear, and lock-in buttons. The functionality for each of these is covered separately in the following chapters. Each button contains an icon that attempts to convey its function. For example, the 'Lock in' button is represented by a lock; to convey an action where the player 'locks in the answer.' Colors are also used to convey action. For example, 'Clear' features a red-like color to convey an irreversible action, and 'Lock in' is green to convey completion.

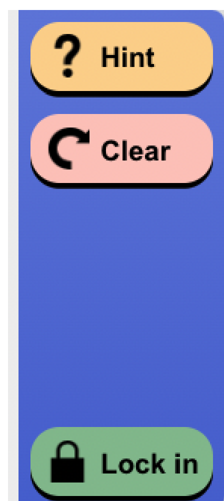


Figure 44: The sidebar.

Hints

The user can get hints for the current task by clicking the 'Hint' button in the sidebar. This brings up a modal window where the user can navigate through the provided hints by clicking the arrow buttons,

see figure 45. The background is tinted white to direct the focus towards the hint modal window. This modal window can then be closed by clicking the 'Back to task' button or outside the modal.

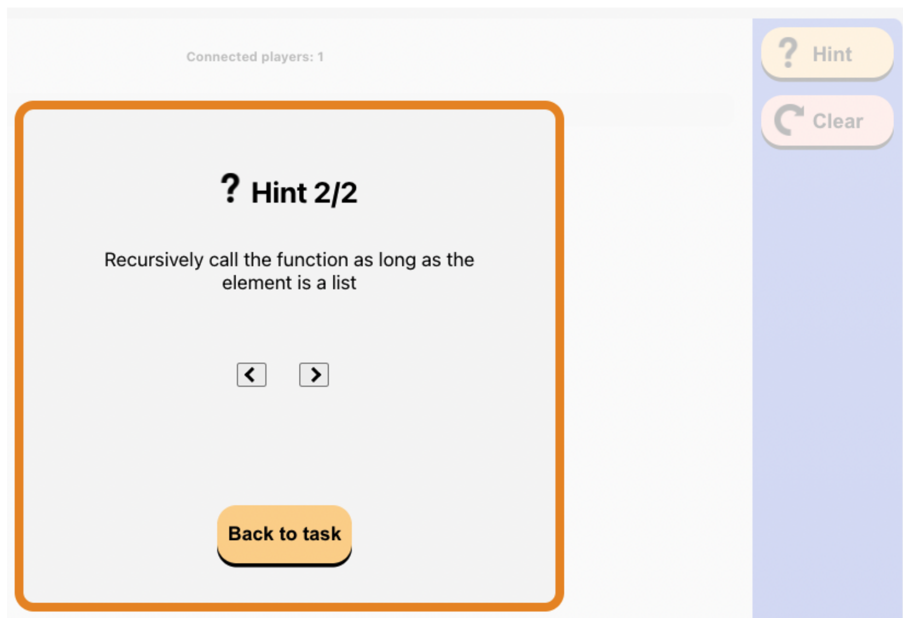


Figure 45: The hint modal window.

Clear board

The user can clear the solutionfield by clicking the 'Clear' button in the sidebar. This moves all code blocks in the solutionfield back to their owner, leaving the solutionfield empty. This action must be confirmed, as it is an irreversible action, see figure 46. Confirming this action also acts as a tolerance for error to account for misclicks. There is no voting system for clearing, which means that a player can perform this action without mutual agreement beforehand.

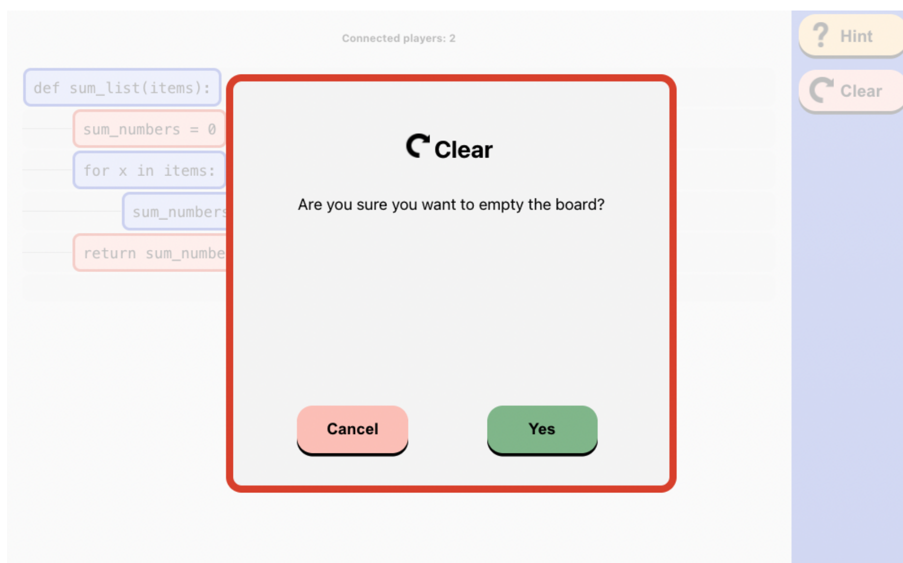


Figure 46: The clear modal window.

Lock in solution

When someone wishes to submit the current solution, they can *lock in* by clicking the 'Lock in' button in the sidebar. This action makes all code blocks immovable for that player while also signaling to the rest of the team that they locked in, see figure 47. Additionally, the solutionfield is overlayed with a lock icon to convey that the player cannot interact with the current solution. All fields featuring code blocks are also greyed out to convey this state further. The solution will be submitted when all players have locked in. A player can choose to *unlock* at any time. If someone *clears* the board or the submitted solution is incorrect, TPP will unlock automatically.

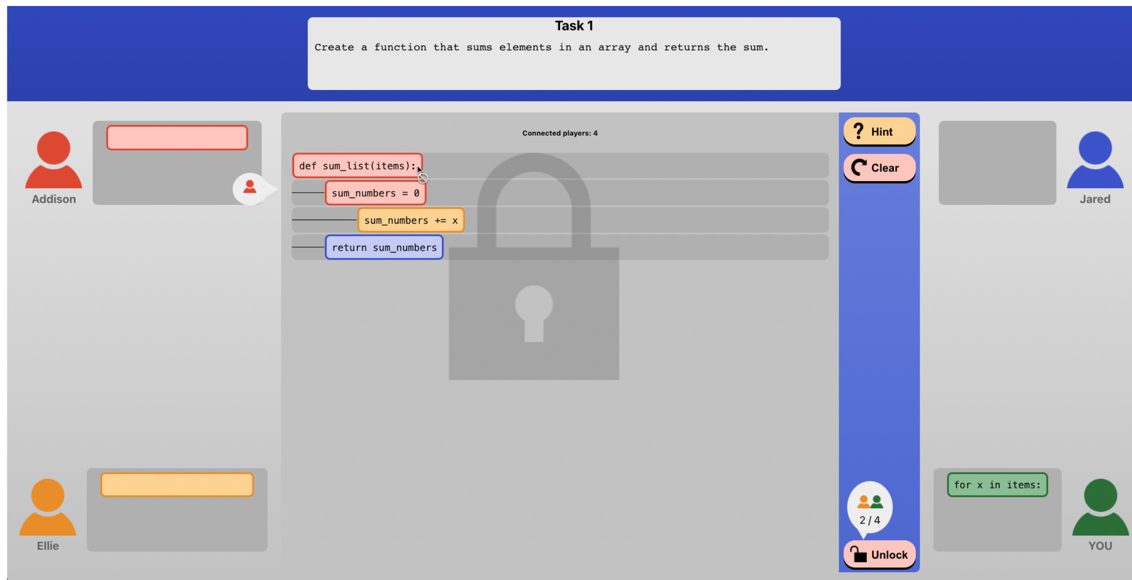


Figure 47: The interface when two players have locked in.

Line-based feedback

The solutionfield will be submitted for assessment when all players have locked in. If the solution is correct, TPP will initiate the next task and inform the players about this. However, if the solution is incorrect, TPP will open a modal window that provides *line-based feedback* to the players, see chapter 2.2. Each line is highlighted based on the code block's absolute position and indentation, see figure 48. Red and green are used for incorrect and correct lines, respectively.

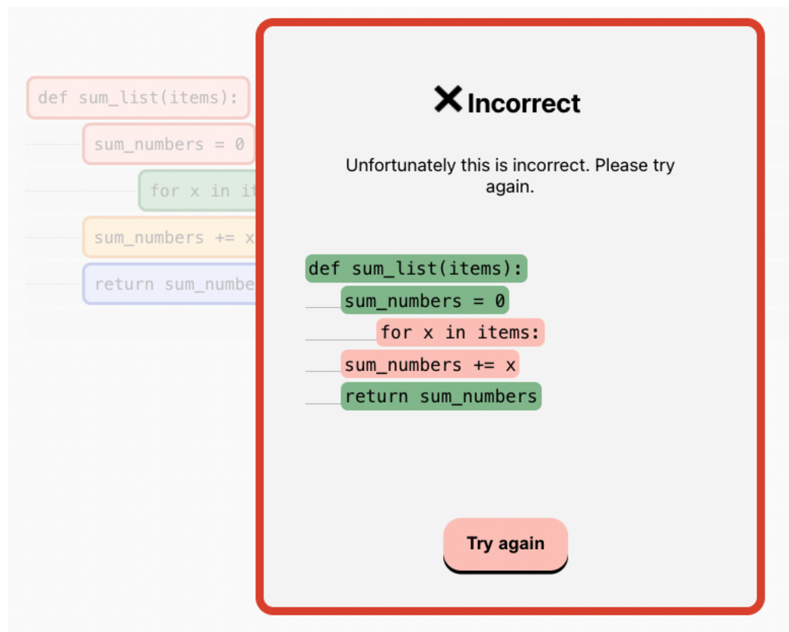


Figure 48: Line-based feedback for a submitted solution.

Player indicator

TPP will indicate which code blocks are being interacted with by prepending a code line with icons of the interacting players, see figure 49. Chapter 4.4.1 gives an overview of the different ways of interacting with code blocks. Player indication provides the players with an overview of who is doing what in real-time.

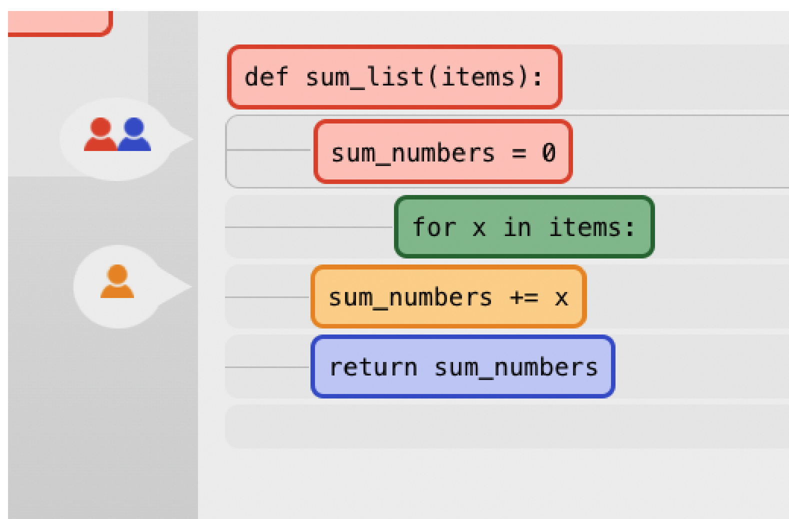


Figure 49: Player indicator for a code block interaction.

Adaptive layout

TPP will change its layout dynamically based on the screen size; see figure 15. The 'row' layout is set as default which places the players on the left and right sides of the solutionfield. If the screen size is too narrow, the players will be placed on the top and bottom instead.

Congratulations screen

When the players complete the task set, they are presented with a screen congratulating them. For now, the screen only contains a button that takes the player back to the lobby screen; see figure 50. This screen could display a summary of the performance, including completion time and game score, in a later iteration.

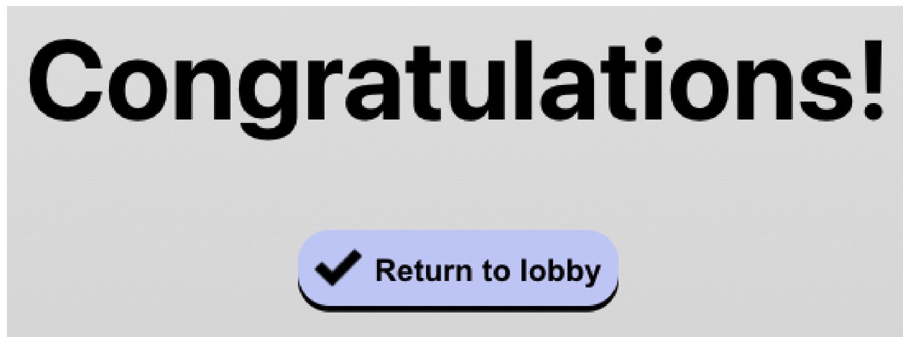


Figure 50: The screen presented after a finished task set.

4.4.2 Architecture

This chapter will walk through how TPP is implemented by looking at the React component tree, what *states* the Redux library manages, and how communication is handled.

React component tree

Figure 51 shows the final version of the React component tree for TPP. The following list will give a brief explanation of how each component functions:

- **Index**: The entry point that renders the game.
- **JoinGame**: Allows the user to enter a nickname and select which game lobby to join.
- **CommunicationHandler**: Handles all incoming traffic from other peers in the network. This includes updating the game state when other players interact with the game.
- **CommunicationListener**: Listens to changes in the Redux Store in order to broadcast them to the other peers in the network.
- **App**: Renders the correct state of TPP, including the congratulations screen, the game, or the lobby.
- **Finished**: A screen that congratulates the player for finishing a task set and allows the user to return to the lobby.
- **MainPage**: Contains the topbar and the game components.
- **Lobby**: The screen displaying a team and their selected task.
- **Topbar**: Renders the Task component.
- **Task**: Shows the current task number and description.
- **Game**: Contains the main elements of the game, including the sidebar, player fields, and the solutionfield.
- **Sidebar and SideBarButton**: Contains the hint, clear and lock-in buttons.
- **Player**: A field that contains the player icon, nickname, and its distributed code blocks.

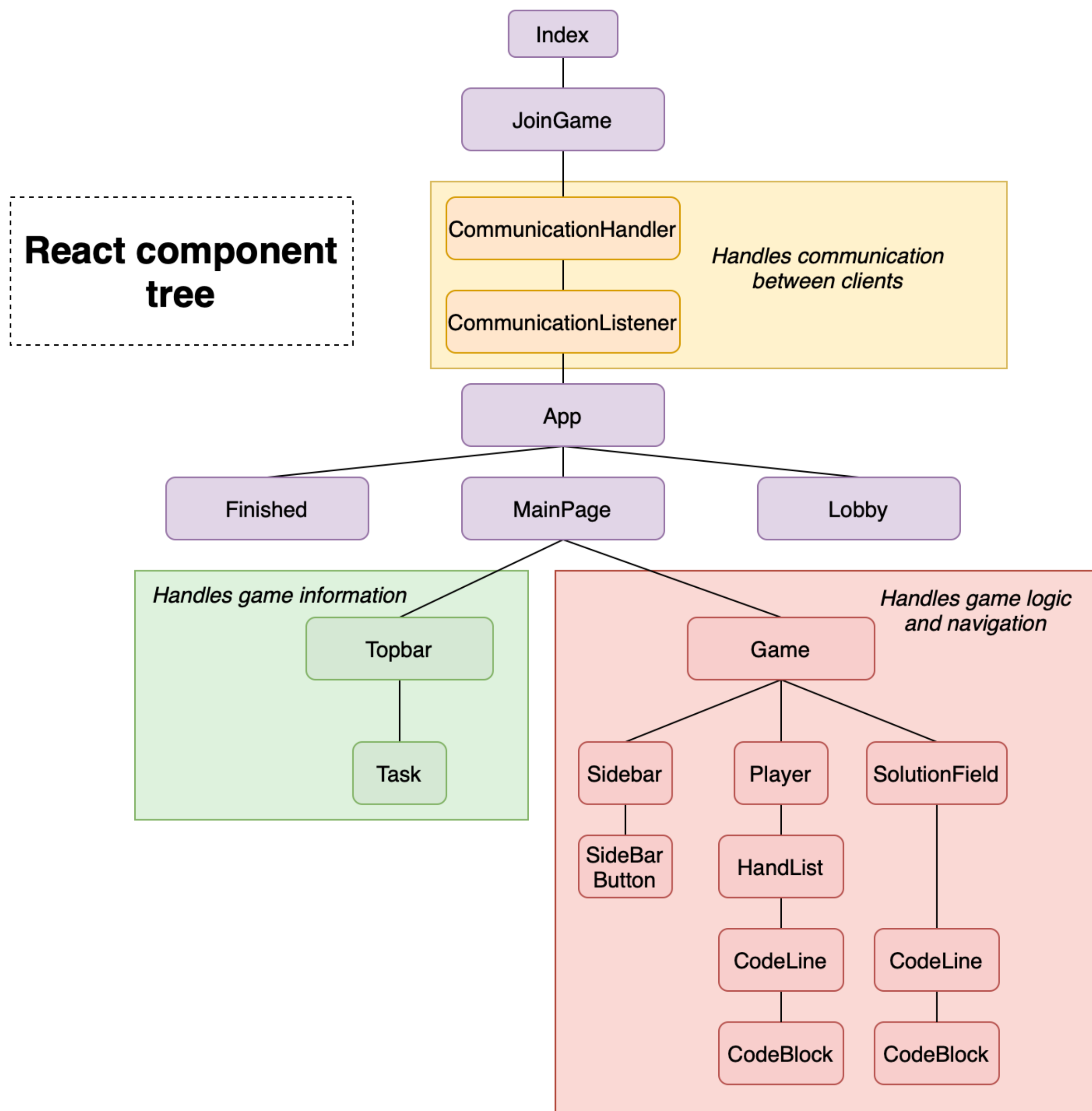


Figure 51: The final version of the React component tree.

- **HandList**: A field that contains code lines.
- **CodeLine**: A field that CodeBlock components can be dropped into. This component is reused in the SolutionField component.
- **CodeBlock**: A user interface object that players can move around.
- **SolutionField**: A field that contains code lines that allow indentation.

Redux Store

TPP utilizes Redux to manage the states of its components. This has two advantages: 1) Redux updates all components that rely on the same data across the component hierarchy, and 2) Redux allows data to move up. TPP requires data to be broadcast to other peers in the network, which is handled by the CommunicationHandler component, see chapter 4.4.2. Redux is used to signal game state updates to this component. An example of this is when a block is moved into the solutionfield, and this information must be broadcast to the other peers in the network to ensure everyone's game states are synchronized. Figure 52 illustrates this logic.

See table 9 for an overview of the states Redux manages in TPP.

State	Description
clearEvent	Inform other players that the board was cleared.
fieldEvent	Inform other players that the solutionfield was updated.
listEvent	Inform other players a handlist was updated.
taskEvent	Inform other players that the next task has been initiated.
finishEvent	Inform other players that the final task has been completed.
taskSetEvent	Inform other players that the selected task set has been updated.
selectEvent	Inform other players that a code line was selected.
lockEvent	Inform other players a player has locked in the solutionfield.
handList	The code blocks each player possesses.
solutionField	The code blocks placed in the game board.
currentTask	Information about the current task.
players	Information about all the players.
status	The status of the game (lobby, finished, game).
host	The player that is the host for the game session.
moveRequest	The last move request made to the host.
selectRequest	The last code line select request made to the host.
lockRequest	The last lock request made to the host.

Table 9: Redux store states.

Communication

The communication between peers utilizes the WebSocket protocol. This is a two-way communication protocol where message events are defined to trigger different actions when received. These messages are broadcast and received in the CommunicationListener and CommunicationHandler components, respectively. Broadcasts are triggered by dispatching events using the Redux state management library, see chapter 4.4.2. When the CommunicationListener component updates, it will check what event was dispatched by comparing the previous and current states in the Redux store. For example, selecting a code line will dispatch a "selectEvent" event, which will be caught in CommunicationListener. This will cause the component to broadcast a message to the other peers. This message contains a payload that may include any necessary data to update the game state; in this case, what code line is selected and by whom. This message is then received by the other peers in the CommunicationHandler component, which will display the player indicator for the selected code line. This logic of broadcasting messages upon state changes is how TPP ensures all peers are synchronized during play. Figure 52 shows a flow chart to help illustrate how this logic works. Lastly, an overview of the message events can be seen in table 10.

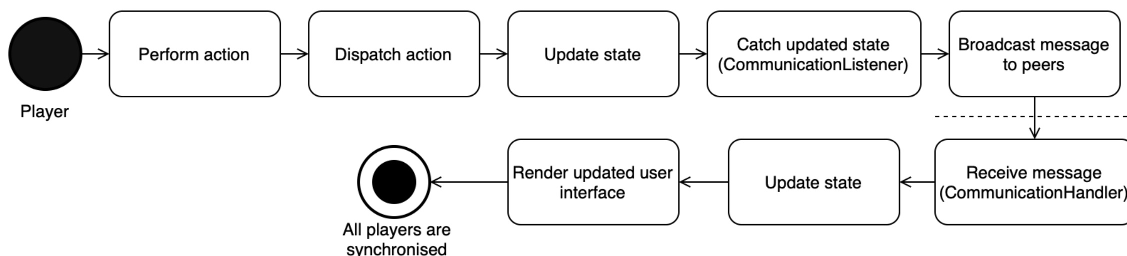


Figure 52: Flow chart illustrating how communication is implemented in TPP.

4.4.3 Performance

We chose to time the movement of code blocks since these functions are called the most often, see table 11. The start time was measured when the ReactDnD library or the Redux state registered a player input

Event	Description	Payload
SET_LISTS_AND_FIELD	Updates the handlists and the solutionfield.	Two arrays w/code blocks
MOVE_REQUEST	Request to move a code block - sent to host.	{id, index, indent, field, timestamp}
SELECT_REQUEST	Request to select a code line - sent to host.	{index, pid}
SELECT_EVENT	Update player indication for a selected code line.	{pid, index}
LOCK_REQUEST	Request to lock in the solution - sent to host.	{forWho}
LOCK_EVENT	Toggle the lock for a player.	{pids , lock (true/false)}
START_GAME	Initiate the game with a task set.	{taskSetNumber}
CLEAR_TASK	Clear the solutionfield.	-
NEXT_TASK	Initiate the next task.	{task}
FINISHED	Inform that the task set is finished.	-
SET_TASKSET	Update the selected task set in the lobby.	{taskSetNumber}

Table 10: Communication protocol in iteration four.

[80]. The end time was measured after the Redux Store had dispatched an update [77]. Subtracting the start time from the end time gave us the time spent to perform a task. Note that a lot of traffic in-game will lead to a worse performance than what this benchmark indicates. Internet speed will affect the time it takes for messages to be sent between peers, thus affecting performance.

Description	Time
Time measured between a non-host moving a block and the preview updating the board.	2.5 ms
Time measured between the host moving a block and the host receiving an updated board.	4 ms
Time measured between a non-host moving a block and the host receiving an updated board.	215 ms
Time measured between the host moving a block and a non-host receiving an updated board.	230 ms
Time measured between a non-host moving a block and another non-host receiving an updated board.	435 ms

Table 11: Some performance benchmarks for moving code blocks. The times are calculated as approximate average values.

4.5 SUS score

In the final user test, we asked our participants to answer a questionnaire; after they had used TPP for about 15 minutes. The participants answered ten statements in order to measure the System Usability Scale (SUS) score, see chapter 2.9. Table 12 shows the calculated SUS score for all participants and the mean score. The mean score was 77.5, which according to table 1 gives an adjective rating of *GOOD*. As the score is well above 68, it is also considered *acceptable*.

P1	P2	P3	P4	P5	P6	P7	P8	P9	P10	P11	P12	P13	P14	P15
72.5	77.5	67.5	92.5	52.5	87.5	75	87.5	77.5	77.5	92.5	65	72.5	75	90
MEAN SCORE								**77.5 - GOOD**						

Table 12: System Usability Scale scores from final user test.

4.6 Other findings

This chapter highlights findings relevant to the discussion but not directly tied to any of the research questions. These questions were asked as a part of pre-test questionnaires. *How do you prefer to work?* was asked in the final user test; see figure 53. The other questions, see figures 54, 55 and 56, were asked in both user test 3 and 4.

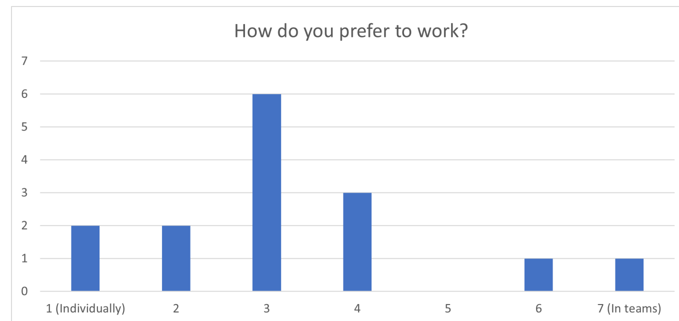


Figure 53: Bar chart of the responses to Q10

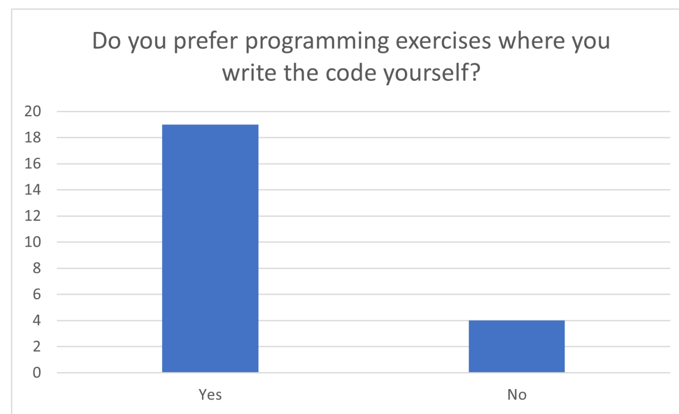


Figure 54: Bar chart of the responses to Q11

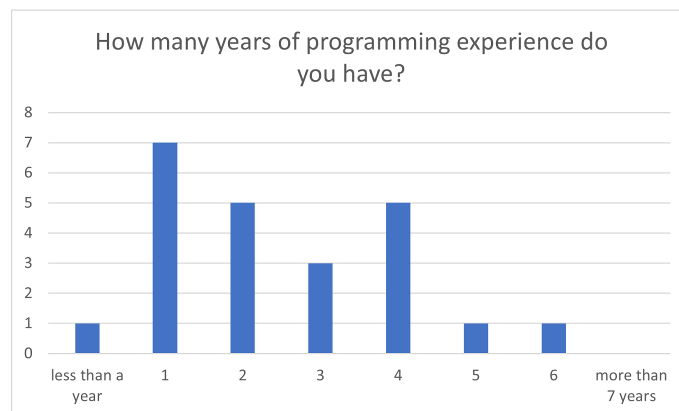


Figure 55: Bar chart of the responses to Q12

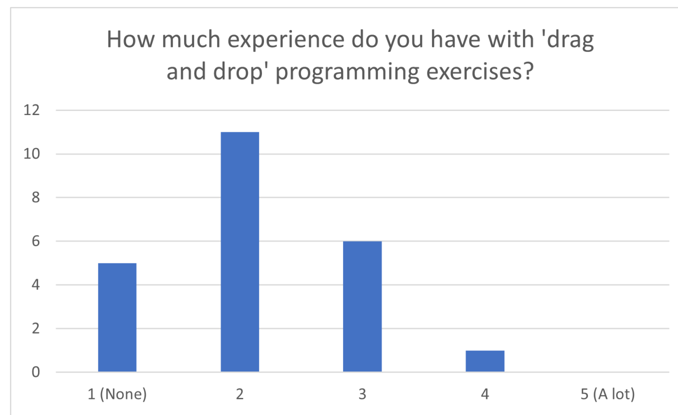


Figure 56: Bar chart of the responses to Q13

5 Discussion

In this chapter, we discuss our findings and evaluate our project. We start this chapter by evaluating our findings relating to our research questions. Next, we evaluate our methodology, our potential solution for a cooperative Parsons problems game, and our fulfillment of the project goals. We also address some of the limitations of this project. Lastly, we present our recommendations for further work.

5.1 Evaluation of findings

In this chapter, we will discuss our results relating to the research questions defined in chapter 1.3. For each research question, we will summarize our findings and discuss how our results relate to the motivation from chapter 1 and literature review from chapter 2.

5.1.1 RQ1: Can cooperative Parsons problems be a fun and social activity for novice programming students?

Studies report that students in higher education are feeling lonely [31, 32, 33, 34, 35]. Loneliness, as a mental distress, is claimed to negatively affect a student's academic study and overall well-being [36, 37]. We theorized, in chapter 1.1, that the introduction of cooperative learning and gamification could promote socialization through cooperative structures and positive goal interdependence. We asked our user testers if they thought TPP could fit that role. Additionally, we wanted to know if they found the game to be fun, as we considered it an important motivational factor for learning.

Figure 30 shows that 14 out of 15 the participants agreed that *the game could improve socialization* among students. However, 4 out of 15 would rather have played this game alone. We asked whether the participants preferred to work alone or in teams in the pre-test questionnaire. Interestingly, the participants seemed to prefer to work alone in the pre-test questionnaire, see figure 53. The participants seemed to be more positively inclined to play the game cooperatively than generally working together in teams. They also claimed that the game could *increase teamwork*, as evident by the thematic analysis, see figure 31. In the group discussion, several participants also mentioned that they would rather play this game with friends instead of it being a "get to know each other" activity. This is also reflected in the thematic analysis, see figure 31, where there were more mentions of working together to solve a task compared to mentions of *getting to know each other*.

Our data suggest that the participants found the game to be fun. A common theme that emerged in our thematic analysis was *fun activity* in regards to how TPP could impact socialization among students, see chapter 4.1.2. All participants agreed with the statement: *the game was fun to play*; in the final user test. For example, a participant wrote that "[This game] It is a conversation starter and a fun way to get students to engage socially". Another participant mentioned, during the group discussion, that the fun came from the chaos that emerged; when everyone moved their code blocks simultaneously.

Several participants mentioned that it was the cooperation that made the game fun, both in the group discussion and in the thematic analysis, see figure 31. This statement aligns with Zhang's principles for social relatedness, which state that software should be designed to facilitate human-to-human interaction; see table 4. The *people fun* key also aligns with the statement that fun and cooperation go hand in hand. The *people fun* key suggests that people that play together express more feelings. See chapter 2.11.1 for more information on the *people fun* theory.

However, several participants mentioned that they thought that the game would become boring quickly. For example, a participant wrote in the post-test questionnaire that: "I think it [The game] could quickly become boring after 1 or 2 days". We also got requests to add competitive elements to TPP; in the form of a leaderboard. Several participants told us that they were motivated by competition. This trait can also be seen in the *competitiveness* (C11) characteristic in table 8. This characteristic also aligns with the *people fun* theory in chapter 2.11.1. Other participants pointed out that the introduction of competitiveness could demotivate struggling students.

5.1.2 RQ2: What are the players' perceptions of the potential learning outcome from solving cooperative Parsons problems?

In chapter 1.1 we presented Parsons problems and how they can be used to help struggling programming students; by reducing cognitive load, see chapter 2.4.1. We theorized that we could increase the learning outcome even further by introducing cooperative gamification. We based this assumption on previous studies on transactivity and cooperative gamification. Transactivity is a measurement of the depth of social interaction in cooperative activities, and transactivity has been empirically tied to improved learning outcomes [26, p. 352]. Cooperative gamification can encourage participants to share and interact with each other to solve a problem [54, 55, 30]. See chapter 2.4.2 and 2.4.3 for more information on transactivity and cooperative gamification. We chose to focus on the perception of learning since we deemed that we did not have the time to test actual learning outcomes, see chapter 5.4. Perceived learning outcome can also be linked to a player's motivation, referring to *serious fun* in chapter 2.11.1.

80% of the participants thought that the game could be useful for testing newly learned programming concepts; see figure 32. An instructor could, for example, introduce a new topic such as loops or functions. The students would then play the game in teams to ensure that everyone understood the concept. Figure 34 shows that 70% of the participants thought that the game could create discussion about programming concepts. We also observed that the players discussed problems; if they were to disagree on a task. According to the thematic analysis and our observations, the discussion would often involve players trying to explain their thought processes. A participant even wrote that: "The [game offers the] opportunity to see how other students think". There were 15 mentions of the theme *explain code to others* in the thematic analysis, see figure 36. Transactivity has been empirically tied to improved learning outcomes, see chapter 2.4.2. Discussion and *explain code to others* relate to both *dialectic* and *dialogic transactivity*, respectively. However, we observed that an increase in transactivity could result in misunderstandings, especially if a team was exposed to the *leader emerging* characteristic (C1), see table 8. Players tended to agree with the leader, even when the leader was wrong.

Figure 33 shows that just over 50% of the participants agreed with the statement: the game could discover weak areas in their programming knowledge, whereas 20% disagreed. We argue that we might have phrased the question incorrectly in the questionnaire. The other questions consider general benefits, while this question asks for their personal gain. The answers might be affected by the fact that the participants did not consider themselves the target group. The participants were also split on whether or not this game could be used as a repetition exercise, see figure 35. Just over 50% of the participants agreed with the statement, whereas 20% disagreed. We observed that the players showed more engagement in tasks that they considered to be more difficult.

The thematic analysis also suggests that the game could be more educational if the game provided better feedback. A participant wrote that: "I think it would be more educational if there were an explanation after each task detailing why things work and what every line does". Some teams essentially submitted the correct solution but with a wrong index or indent, resulting in feedback with only incorrect lines. In the group discussions, several participants mentioned that they found this to be frustrating. We noticed in our observations that complex tasks encouraged trial-and-error behavior, see table C8 8. We theorize that the trial-and-error behavior could be reduced if we were to include more educational feedback. Several participants also requested execution-based feedback or debugging functionality. A participant in a group discussion pointed out the importance of learning how to locate errors through debugging.

We also got feedback on the use of Parsons problems in general. A participant wrote that: "In this game we were limited to the handout code, which means that we had to solve the exercise in a way we normally would not. I think that this game could expose players to different ways of coding". In the group discussions, we found out that most participants were not as positive about using Parsons problems as they thought they would learn more from writing code themselves. Several participants also requested that we should replace drag-and-drop with code writing in TPP. A poll from the pre-test questionnaire also reflects this; see figure 54. It would be interesting to explore whether it would be possible to replace drag-and-drop with code writing; in a similar game. However, we suspect that such a game would still face problems related to intra-group coordination, such as leader emerging (C1) and passivity (C2) in table 8.

The results from the thematic analysis suggest that TPP can be *good for learning how to code*, as suggested by the themes; see figure 36. The thematic analysis also suggests that TPP would be more suitable *for beginners*, as it was another prevalent theme. Therefore, we claim that if TPP were to be used in higher education it would be mostly beneficial in the early phases of a programming course.

5.1.3 RQ3: How does a team work together to solve cooperative Parsons problems in a digital space?

This research question aims to explore how teams work together when solving cooperative Parsons problems. In chapter 1.1 we explain that our motivation for this project was to develop software that could promote socialization, motivation, and learning for novice programming students. In order to gather more insight on how to promote these themes, we defined this research question. After reviewing the user tests, we developed a list of characteristics of teamwork when using a game such as TPP. This knowledge can hopefully be beneficial for design decisions; in future iterations of TPP or for creating similar cooperative Parsons problems games. Chapter 4.3 presents the results derived from 7 user tests across two iterations. Keep in mind that these characteristics are our own interpretations from observations, group discussions, and questionnaire answers.

Firstly, we observed that a leader would emerge in a majority of the teams (C1). The leader would often take the initiative to request blocks (C4), review the task description (C5), review the code (C9), and in general, contribute heavily to the overall discussions and interactions within the game. It is not entirely clear why this happened. It is possible a student's confidence in their programming abilities can contribute to this, as they might believe that they can guide the team towards the correct solution. Additionally, the pre-existing group dynamics could also be a contribution. As we only tested TPP on students who already knew each other prior to the tests, we cannot speculate too much on this. It would be interesting for future research to observe how teams of strangers would work together in such a game. During a group discussion on the topic of leadership, one participant said that it is unrealistic to expect everyone to understand what is going on at any given time. According to the participant, it is therefore beneficial to have a leader who can coordinate the teamwork.

We also observed that there would typically be one person in a team that took on a more passive role (C2). This observation is also reflected in the theme *imbalance of participation* that emerged in our thematic analysis, see chapter 4.1.2. We prompted the participants on this during the group discussions by asking how the teamwork was. A few participants said they would sometimes fall behind the others (C10). The other members would start solving the task before everyone had finished reading the task description and understood the objective. Falling behind could lead to passivity for some, as they struggled to keep up with what the team was doing. Another factor that contributed to passivity was task difficulty. A participant mentioned during the group discussions that: "if it was a difficult task, I just put in everything [their code blocks] and let the others solve it". Another participant said: "I just moved in all of my blocks and let the other do the work." According to the theory, see chapter 2.1, while *pair and cooperative programming exercises* such as cooperative Parsons problems can be good for creating discussion, team composition can be challenging. Assuring an appropriate task difficulty, and that the team members are on a somewhat equal level in terms of programming skills, could help circumvent this issue. To address this, TPP could measure individual participation in a team. We theorize that some players would step up if we were to group together individuals with low participation. However, it might also be the case that participants with low participation require easier tasks.

On the topic of difficulty, we observed that harder tasks created more discussion compared to easier tasks (C7). There would not be as much discussion during the easier tasks, as the teams were able to solve them rather quickly. Harder tasks usually led to more reflection as well (C6). Reflection would usually be followed by 'eureka'-moments when a programming concept or an aspect of the task became apparent. Another side effect of increased task difficulty was the introduction of trial-and-error behavior (C8). This aligns with the theory that *line-based feedback* encourages trial-and-error behavior, see chapter 2.2.3. Since TPP did not punish this behavior, they might have felt inclined to submit more frequently when the tasks got harder. We believe this can be circumvented by implementing scores and a limited amount of attempts per task. According to the theory, see chapter 2.11.2, scores can discourage trial-and-error behavior while also creating motivation.

Trial-and-error was one of the strategies we observed during play. Another strategy was moving all the

code blocks into the solutionfield at the start of a task (C3). Some participants said they did this to circumvent the invisible code blocks. Another participant claimed that it became easier to get an overview of how to solve the task when they could see all of their options. This strategy defeated the purpose of having the code blocks be invisible, as the feature was indented to promote communication between the players. On the other hand, the teams that did not follow this strategy relied on requesting blocks from each other (C4). As previously mentioned, the leader would usually initiate this by asking questions such as: 'does anyone have a *def*?' or 'can someone place the return statement?'. This strategy became less prevalent during the harder tasks, usually because the players would simply place all of their blocks in the solutionfield from the start (C3). Lastly, another strategy that we observed was reviewing the code collectively (C9). Once again, the leader would usually initiate this by reviewing the code line by line, explaining how it works, and asking if the others agreed. This approach often led the team to discover where the mistakes were located. On the contrary, we observed that teams that did not follow this strategy would review the code in silence. The latter was less prevalent in teams that featured leaders.

A majority of teams showed competitive behavior (C11). They would try to solve the tasks quickly and showed excitement when solving tasks. This evidently resulted from the teams wanting to beat each other's times and scores. Occasionally, a few teams would ask if they performed better than some of the other teams, even though we assured them that we did not measure their performances. It is unclear if this characteristic would occur naturally in a more realistic classroom setting. Because the groups knew they were a part of a user test among other participants, they might have felt some inherent desire to beat their classmates. Implementing scores and leader boards might help create a competitive environment in a classroom setting. However, this might not be a desirable outcome. While competition might motivate some, we argue that it could also create unnecessary stress for others. It is also possible that competition promotes an approach of trying to solve the tasks as quickly as possible, neglecting the learning aspect of the game, as there might be less discussion and reflection around the relevant programming topics. This last point is more speculative and should be explored further in future iterations.

5.1.4 RQ4: How could a cooperative multi-player Parsons problems game be designed?

This research question aims to explore how a cooperative multi-player Parsons problems game can be designed. As stated in chapter 1.2, we did not find any existing solutions for this type of game. Therefore, we decided to develop our own game as part of this project. This game, referred to as *TPP*, was tested across four iterations, including seven user tests, to gather feedback on how such a game could be designed. We hope to answer this research question by reviewing what we believe worked and did not work with TPP; by discussing its features, design, and usability. Chapter 4.4 gives a thorough overview of the final version of TPP's user interface, features, architecture, and performance.

With a System Usability Scale score of 77.5, see chapter 4.5, the data suggests that TPP is a *GOOD* and usable product. TPP also got positive feedback on its design during the user tests as it was quoted to be 'intuitive,' 'simple,' and 'easy to use.' One participant wrote: "I like the layout with the code in the middle and the players surrounding it. It is intuitive mapping and simulates players sitting around a table in real life. The intuition leads the eyes towards the middle, and you understand quickly how to interact with the game". This quote suggests that TPP follows principles 3 and 4 of Universal Design, see chapter 2.7. We argue that the tutorial our participants were asked to watch before play contributed to this. It would have been interesting to perform user tests without the tutorial to see if the participants could independently figure out the controls and the objective.

We also believe TPP fulfills principle 2, *Flexibility in Use*, by providing the players with multiple options for moving code blocks. In addition, confirmation modal windows provided *tolerance for error*, which follows principle 5 of Universal Design. However, the final design is not without its flaws. A few participants expressed that the design looked 'childish' due to its color scheme. Since TPP was indented to be used by students in higher education, we think the color scheme could be improved. Lastly, TPP also has some issues with its layout across various screen sizes and resolutions. During the final user test, we observed that smaller screen sizes reduced the usability to some extent. Some participants also had to scroll through the task description field to see the full content. In this regard, TPP fails to some extent to follow principle 7 of Universal Design: *Size and Space for Approach and Use.*

We believe distractors were a good addition to TPP, as they require players to reflect on why the content of a code block is correct or incorrect. However, we cannot confirm whether distractors made the tasks harder, as stated in theory; see chapter 2.2. Whether distractors added difficulty could have been tested by performing a test where the performance was measured based on the inclusion or exclusion of distractors. In regards to scaffolding, we opted to use *student-scaffolding*, see chapter 2.2. We deemed it to be appropriate for the Python language as it requires the players to set correct indentation. One participant mentioned that indentation is an important aspect of learning, specifically for languages such as Python. Since we did not implement *pre-scaffolded* problems, we cannot draw any conclusions on its effect. However, participants seemed to handle indentation well. Lastly, we did not test *part-complete problems* as a feature since we did not include it in our user tests. However, some participants did suggest that some blocks could already be placed in the solutionfield, for example, function declarations and return statements. This suggestion would classify as a *part-complete problem* and could be interesting to test further.

Both drag-and-drop and double-clicking worked well for moving code blocks. We also observed that both dragging, and using the tab or backspace keys, were used for indenting blocks. Overall, we received few complaints regarding code block interaction. However, there were some issues regarding synchronization when there were a lot of simultaneous code block interactions. In those scenarios, we observed that blocks would temporarily skip back and forth, which caused the players to exclaim in confusion. This was a result of the implementation of previews which attempted to fix the delay caused by the *player as host* solution, see chapter 3.4. Overall, we believe this issue was not too impeding but could be mediated by slight adjustments to the preview solution. We evaluate our choice of architecture in chapter 5.3.2, such as how the *peer-to-peer* networking architecture worked for TPP.

The user testers had varying opinions on the decision to hide other players code blocks in the handlists. One participant wrote: "[I think] it is good that you cannot see each other's code. Instead, you have to communicate what you need [code blocks], and then you find out what everyone has afterward. Therefore you have to start with a common understanding and structure of the code, and then cooperate on how to advance with the code blocks you have received". However, other players felt it to be cumbersome as they would rather have all of their options visible to them from the start.

We followed guideline 4 regarding *resources*, see chapter 2.11.2, where the distributed code blocks acted as resources. Our intention with this was to promote communication and cooperation between the players, but as evident by characteristic 3, see table 8, it ended up being counterproductive. We believe this behavior could be prevented by implementing some restrictions, for example, restricting how many blocks a player can move into the solutionfield within a certain time frame. One participant also mentioned that this strategy would not work as well if there were more distractors. The participant argued that this behaviour would be discouraged as the distractors would not be useful for the solution. More research should be done on this aspect in order to decide on the most beneficial design to promote cooperation.

We received some feedback that it was difficult to maintain a good overview of the code when there were a lot of simultaneous interactions. *Player-indication* improved on this from iteration three, but this feature does not apply when moving a code block from a handlist to the solutionfield. Initially, we had planned to display all the players' cursors, something akin to Figma [75]. We believe this feature could improve clarity, while also providing the players with a tool for communicating within the game itself by, for example, referring to- or highlighting code with their cursors.

The *lock-in* system for submitting tasks seemed to be intuitive during user test four. We observed that when one player locked in, the others would follow. This suggests that the system worked well for coordinating task submission. However, we got feedback that it was cumbersome that all players had to lock in if they needed to re-submit after making adjustments to the code. Some participants wanted a quicker submission process, potentially akin to how submission worked in iteration three. We argue that this could lead to more trial-and-error behavior as the lock-in system inherently adds some resistance that prevents a player from simply submitting multiple times in rapid succession.

The opinions on the code solution feedback itself were mixed. While some participants thought the feedback to be intuitive, others were confused as it was not clear that it was *line-based*, see chapter 2.2. Since TPP assessed the code based on the absolute position of code blocks, it led to some confusing scenarios. For example, if the order was correct, but slightly offset by one line, the game would mark

the entire solution proposal as incorrect. The game does not separate its feedback on whether a code block is on a wrong line or if the indentation is incorrect. Js-parsons, see chapter 2.3, features this separation, and we argue it would be a good addition to TPP. We also had multiple participants express that *execution-based feedback* would be preferred compared to our *line-based* implementation, see chapter 2.2. This statement is also reflected in the theme *insufficient feedback* from our thematic analysis, see figure 36. One participant, in particular, claimed that it would be beneficial for novice programming students to see the stack trace, as debugging is an important aspect of being a programmer. The literature, see chapter 2.2.3, states that students requested less feedback when it was *execution-based*. This suggests that this form of feedback was more helpful than *line-based*, as they were able to solve tasks with less errors.

The hints did not seem particularly helpful during the user tests based on our observations. When the participants were stuck on harder tasks, some would resort to opening the hint modal window. However, it did not seem to help them figure out the solution. We believe this was more of an issue of our execution rather than the hint functionality being inherently ineffective. Writing helpful hints is therefore crucial. The *clear* functionality was mostly neglected by all groups. If the participants had to make major adjustments to the code, they would do it manually. However, we acknowledge that 15 minutes is a relatively short time to fully explore all the features of any program. It is also possible that the participants would have requested other hotkeys or features if they had more time to explore and play the game.

We mentioned in chapter 5.1.4 that player passivity could be caused by the struggle to keep up with the team during play. Buchinger and Hounsell's guideline 2, regarding synchronization, states that turn-based games allow for more time to think about actions, which could help mitigate this issue. However, the guideline also states that the long waiting times could lead to boredom and slow down the game dramatically. Ultimately, we believe synchronization was the better alternative for TPP, as we believe the playability would have suffered if the game was turn-based. We also believe a maximum of 4 players was appropriate, as bigger teams could have potentially led to more passivity. We also followed guideline 3 regarding *roles*, see chapter 2.11.2, which states that all players should have the same role if the content is important. However, as we discussed in chapter 5.1.4, having one player as the leader could be beneficial for coordination. Other potential roles could be a player that ensures that everyone is heard, and another player that ensures that the team is reflecting on the various topics of the task. Furthermore, assigning roles that have different actions in-game is also an interesting idea.

5.2 Evaluation of methodology

This chapter will present the evaluation of our methodology. We will cover the iterative process, development process, the user tests and our thematic analysis. In our evaluation of the iterative process, we discuss whether we followed and benefited from the approach. We then discuss two of the phases in the iterative process: the development phase and the user test phase. Lastly, we evaluate our thematic analysis, which we performed after the fourth user test.

5.2.1 Evaluation of the iterative process

We followed an iterative process to fulfill the design thinking and design science methodologies, see chapter 2.5 and 2.6. We believe that we were able to utilize these methodologies successfully. At first, we defined what kind of game TPP would be and what problems it would address, see chapter 1.4. Next, we defined a set of research questions, see chapter 1.3, which would become a primary focus for the final user test. We then performed the last four phases of design thinking through the iterative process, including requirement specification, design, development, and user testing, see chapter 3. This process was performed four times resulting in four complete iterations. These phases also include activities 2, 3 and 4 in design science. Lastly, we evaluated TPP based on the final user test and documented our findings, see chapter 5.1, which relate to activities 5 and 6 in design science.

A phase of the iterative process involved feedback from user tests. This feedback would occasionally provide ideas for potential functionality that we later defined as new requirements. However, we did not manage to implement much of the functionality we got as feedback from the user tests, especially in

the first two iterations. This somewhat diminished the strength of the iterative process. In addition, the priority of requirements required us to focus on many of the pre-existing requirements we had elicited in iteration two. This meant that new feedback regarding potential functionality was moved to the backlog. If we had more time, or a bigger team, we probably would have been able to benefit more from this feedback cycle.

We also believe iteration three could have been split into two. Iteration three lasted over three months, compared to iterations 1, 2 and 4, which lasted between two to six weeks each. The reason for this was that we needed to implement a substantial amount of functionality to make TPP fully playable in iteration three. We argue that we could have focused primarily on the moving of code blocks and task submission in the third iteration, and pushed the lobby and task-creation tool to the following iteration. This would have allowed us to get feedback on the 'feel' of the game before developing matchmaking and task creation.

Overall, we believe the iterative process worked well. It allowed us to set short-term goals, which made the workload easier to digest in the long term. If we had followed a more linear software development process, such as the waterfall model, it would have been difficult to make certain design decisions, especially early on. We argue that a tight feedback cycle during the first two iterations allowed us to create a well-accepted design, which stayed mostly consistent throughout development. It also allowed us to gain experience on how to carry out user tests with groups, which benefited us when conducting our last user test. For example, we were able to set more appropriate difficulty levels for our tasks, and improve our surveys and group discussions. Surveys and group discussion were especially altered from iteration three to account for our research questions.

5.2.2 Evaluation of the development process

The development process took elements from the Scrum development methodology; including Sprint Planning, Sprints, Sprint Review, the Product Backlog and the Sprint Backlog. Each Sprint lasted for two weeks, including a meeting with our supervisor every other week. Each Sprint started with a Sprint Review where we summarised what we had accomplished in the previous Sprint, and what needed to be done for the next one. We considered what activities to prioritize, and placed them in the Sprint Backlog, which would become the focus for that Sprint. However, we were not strict with our deadlines, resulting in us not always reaching our biweekly goals. Elements from the previous Sprint would often carry over to the next. This could have been mediated by spending more time estimating the time necessary to complete each task. In addition, we could have used planning methods, such as Planning Poker, at every Sprint Planning meeting to set more accurate goals. Nonetheless, we believe this format worked well overall for breaking down long-term goals into short-term ones.

We also utilized Trello to help organize our project. Activities were first placed as cards in the Product Backlog, which would be transferred over to the Sprint Backlog during each planning meeting. We would then assign ourselves onto an activity card, and move it to the 'In progress' list. Next, we moved the activity card to the 'Code review' list when the activity was complete. During development, we would review each other's work, and this list would indicate that something had to be reviewed. We would then move the activity card to the 'Address feedback' list if an activity needed revision. Finally, the activity card would be moved to the 'Done' list. This system was useful for reviewing work, providing an overview of what needed to be done, and what activities were in progress. If any of us had any suggestions, we would also put them on the board. The other person could then add a comment to that card to create a discussion. Trello thus became the primary way for us to administer work to be done, and to keep an overview of the development process.

When building the software, we followed test-driven development. We experienced that this process worked well early on when developing the interface. However, this process became more difficult to follow when we started developing the more game-like aspects. For example, the drag-and-drop functionality was complicated to test. It is possible that we could have utilized automation tools such as Selenium [89] for these purposes. The test-driven approach would be abandoned for most of the user-driven actions. Ultimately, we argue that we did not utilize some of the benefits of test-driven development. Being somewhat unfamiliar with React and the libraries used also contributed to this. The test-driven development was primarily beneficial for the task-creation tool, as the form-like interface

lent itself better to the testing framework we had chosen.

5.2.3 Evaluation of the user tests

We were primarily concerned with reliability and validity when evaluating the user tests. We use Lawrence Leung's paper to define *reliability* and *validity* [90]; as we conducted qualitative research. Reliability will refer to the replicability of the processes and the results. Therefore, *consistency* is an important factor we have to evaluate in the following chapter. Validity is concerned with the 'appropriateness' of the tools, processes, and data. Therefore, we will have to evaluate whether our choice of methodology is *appropriate* for answering any questions we might have had going into each user test.

User test 1

For the first user test, our primary goal was to get feedback on two potential designs for TPP. Based on this feedback, we would choose what design to move forward with for the second iteration. The participants were recruited from personal connections, and had various degrees of experience with programming and design. Due to their various backgrounds, we were able to get feedback from different perspectives. This included experienced developers as well as novice programming students. We believe this gave us varied and valuable feedback.

The test was performed using a low fidelity prototype on paper over Zoom [23]. Due to the Covid-19 situation, we had to perform the tests digitally and have the participants direct us to move the paper pieces around. Even though this was less than ideal, we believe it did not impact the quality of the feedback, as we were only interested in the design aspect of the prototypes.

- **Reliability**: Each participant went through the same procedure, see appendix B.1. The designs were presented in the same order, and the same questions were asked for each participant. Naturally, some follow-up questions would differ for each test. The most prominent affecting factor, in terms of reliability, was the participants themselves. Due to the variance in programming experience, they would inherently have different perspectives and opinions when evaluating a design. Overall, we believe there is a strong consistency.
- **Validity**: We believe the A/B testing method worked well for choosing a design for TPP. Only qualitative questions were utilized in an interview-style question round after they had experienced both prototypes. Whether or not the data is valid would depend on whether we could interpret their answers and opinions correctly. This will ultimately be a subjective matter, and is also affected by our own interpretation abilities. Additionally, we would have to stay neutral, and not let our own opinions and preferences of the designs dictate how we perceived the feedback. Ultimately, we believe our methods and interpretations were appropriate for choosing which design to iterate further.

User test 2

The goal for the second user test was to get more explicit feedback on the design, and get opinions on some of the functionality. We performed the test on five participants, consisting of first and second-year programming students. Our intended target group for this project was novice programming students, which most of our participants were. However, it could be argued that the second-year students would be considered too advanced. We deemed them adjacent enough, and decided to include them regardless.

The tests themselves were performed in-person, using a digital low-fidelity prototype. This prototype was developed in Figma, where we had to pre-determine the actions that were possible to perform. This is a limitation of the tool, which created a primarily linear experience for the participants. For example, they would have to move blocks into the solutionfield in a set order, but some of the buttons like 'hint,' 'clear,' and 'submit' could be interacted with at any time.

- **Reliability**: Each participant went through the same procedure, see appendix B.2. In terms of consistency, Figma as a prototype lent itself well for reliability purposes, as the participants were forced to interact in a set order. We argue that this would have been more of an issue if we were testing a more traditional user application, as navigation is an important aspect of those interfaces. In this user test, we were more so interested in whether the game felt intuitive in regards to button actions and code block interactivity. We believe there is a strong consistency with the prototype and the post-test interview.

- **Validity**: Using a prototype tool such as Figma was a natural progression from the low-fidelity paper prototype in iteration one. We were able to gather more feedback on the chosen design and the interactivity of the code blocks. An interview was performed afterward, where we gathered more explicit feedback on specific aspects of the design and functionality. Similarly to the previous user test, our interpretation abilities would affect the validity of the data. We would discuss the feedback afterwards, create a summary of changes to be made, and what to prioritize going forward. Ultimately we believe our testing and feedback gathering methods were appropriate.

User test 3

The user test in iteration three was ambitious in scope. TPP was developed as a fully working game, including a lobby system, code block interactivity, solution feedback, task navigation, and network communication between players. We wanted to get feedback on usability, teamwork, and how fun the participants perceived the game to be. We recruited two teams of four first year-programming students. Each team would work together to solve five tasks using TPP. We met at our school campus and performed the tests in person. Each test consisted of a pre-test questionnaire, playing the game, a post-test questionnaire, and finally, a group discussion.

When the game started, we would observe how they interacted with TPP, and take note of how they communicated and cooperated to solve the tasks. We had little to no experience performing user tests with groups, so getting a complete overview of the whole session proved to be challenging. Especially with the current iteration, there was little visual feedback on the screen when other players interacted with code blocks. We would attempt to interpret their body language, spoken words, and watch their screens to perceive if they at any point felt confused or overwhelmed; when there were a lot of simultaneous code block interactions. The post-test questionnaire also featured a question that prompted the participants on this issue. Similar concerns would also be addressed in the group discussion to elicit more feedback from the participants.

- **Reliability**: The same procedure was followed for both teams, see appendix B.3. The participants were presented with the same tasks, and the teams consisted of four first-year programming students. We observed that the first team had a more structured approach for solving the tasks. They also had a clear leader role emerging that coordinated the teamwork. This contrasted more or less with the second team, which had less communication and a more 'trial-and-error' approach in regards to submitting solutions. Unfortunately, the second team experienced an impeding bug in the software that the first team did not; see chapter 3.3.4. This forced the team to 'play around' the bug, which we argue caused some inconsistency between the two tests. We cannot exclude that this bug gave the second team a worse user experience, which might have affected their feedback.

- **Validity**: Performing the user tests with groups was appropriate, as we wanted to observe how the participants cooperated and how TPP would perform in a more realistic setting. As for the feedback, it could have been better to move some of the questions from the post-questionnaire to the group discussion segment. We suspect it could have been difficult for the participants to express all of their thoughts accurately in text form, especially with the amount of prompted questions. In addition, it might have been more appropriate to discuss specific topics with the whole group, considering TPP is intended to be a cooperative game. We also would have had more opportunities to ask follow-up questions, which might have helped clarify any concerns they might have had. Lastly, the post-test questionnaires lasted longer than anticipated, resulting in little time for the group discussions. Ultimately, we were able to answer most of our questions regarding design and functionality, even though our feedback elicitation approach could have been improved.

User test 4

The goal for the fourth and final user test was to gather data for the research questions. We solved a race condition bug in this iteration; that was present during the last user tests. We also recruited five groups, consisting of 3 to 4 first-year programming students, to participate in this user test. Similar to the previous user test, each team would work together to solve five tasks using TPP. We met at our school campus and performed the tests in person. Each test consisted of a pre-test questionnaire, playing the game, a post-test questionnaire, and a group discussion.

- **Reliability**: The same procedure was followed for all groups, see appendix B.4. All teams were asked to solve the same task set with a 15-minute time limit. This time limit was implemented because we wanted each team to have the same amount of time to solve the tasks, while also ensuring we had enough time afterward to perform the post-test questionnaires and group discussions. We experienced one bug that occurred during play for one team. We believe that this minor inconvenience did not affect the test negatively, as it only lasted about 2 minutes before we restarted the game. However, there was some inconsistency with the team sizes. We argue that it was important for the teams to be of similar size, as communication and cooperation was a major focus for these tests. One team had a few participants missing, resulting in only two members. Ultimately, while there were some inconsistencies with the group sizes, we believe it did not negatively affect the data in any meaningful way.
- **Validity**: Our approach for this user test stayed mostly the same as the previous iteration. After evaluating the validity of the previous user tests, we decided to spend more time on the group discussions. Therefore, we only included two open-ended questions, among a few close-ended questions, in the post-test questionnaire. In addition, more emphasis was put on the team discussion by asking more questions in order to create a dialogue between us and the participants. We argue that this worked well as we were able to gather data for our thematic analysis while also creating a more fluid and open discussion about the relevant topics. However, we believe that the format for this user test lacks some validity compared to the previous user test. The participants only interacted with TPP for about 15 minutes; in both iterations three and four. We argue that 15 minutes was appropriate for iteration three, as we primarily wanted to gather feedback on the usability and the design. However, we believe this time limit was a limitation in user test four, as it created an artificial environment for TPP as a game. As one of our participants mentioned, it is difficult to accurately perceive the effectiveness of the game as a learning and cooperation game in a user test environment. The participant argued that the time limit does not necessarily reflect how the game would be used in a real setting. It is possible that the 15-minute time limit contributed to an artificial setting by unintentionally pressuring the participants to solve the tasks as quickly as possible. Even though we stated that there was no pressure to solve all the tasks, we cannot disregard the possibility that this affected the performance. We argue that not mentioning the time limit could have removed this factor.

5.2.4 Evaluation of the thematic analysis

In order to evaluate our thematic analysis, we will use Braun, and Clarke's 15-point checklist of criteria for good thematic analysis [70, p.96]. Firstly, all of our data was gathered from an open-ended questionnaire, meaning no transcription was involved. During the coding process, we made sure to give equal attention to each answer by highlighting sections of text with different colors to separate them. Next, we generated themes from all of our codes. A mind map was created to group together related codes, which we then used to generate themes. We avoided themes from only a few vivid examples, which is point number 3 in the checklist. We also checked all of our themes against the related data extracts and the entire data set in order to ensure the themes accurately represented them. Next, we ensured that the themes were internally coherent, consistent, and distinctive. Lastly, we analyzed the data by interpreting them using our themes in relation to our research questions, see chapter 5.1. Ultimately, we believe the thematic analysis was successful as we were able to analyze and interpret our data systematically and methodologically in order to discuss and answer our research questions.

5.3 Evaluation of TPP

This chapter will first evaluate the functionality by reviewing our requirement specification. Next, we will evaluate the architecture and performance. Finally we will discuss our choice of technology.

5.3.1 Evaluation of functionality

Table 13 gives an overview of the requirements we completed during the four iterations. We were able to complete the requirements from priority 1 and 2, and some from priority 3. See appendix A for the requirement list.

Status	Requirements
Complete	1, 2, 3, 4, 6, 8, 9, 20, 21, 22, 25, 26, 27, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 45, 46, 47, 48, 49, 50, 57, 58, 59, 60, 61, 62, 63, 64.
Incomplete	5, 7, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 23, 24, 28, 42, 43, 44, 51, 52, 53, 54, 55, 56, 65, 66, 67, 68.

Table 13: Complete- and incomplete requirements.

Completed requirements

Starting with requirements 1-4, creating a new task in the task-creation tool is possible. The tasks are generated from the correct solutions and task descriptions. The task-creation tool also includes the option to add distractors and hints. The resulting output can be downloaded as a JSON file. Continuing to requirement six, multiple solutions for a task are possible, even though it is cumbersome that the solutions have to be added manually. To address this, we could have modified the task-creation-tool to calculate other possible solutions. This could be achieved by checking whether or not the last line was dependent on any previous lines. Fortunately, we did not have to create many tasks for the user tests. If we were to perform a more extensive user test, it could have been beneficial to automate the process of finding alternative solutions to a task.

It is also possible to create task sets within the task-creation tool. Initially, we planned to create an instructor version of TPP. However, as we deemed it more useful to focus on the student's view, rather than splitting our efforts, we chose to develop a task-creation tool for internal purposes. The user must create a number of tasks before the tasks can be added to a task set. This functionality is relating to requirements 8 and 9. In order to play the task set, manually add the generated JSON file to the source code of TPP. Ideally, it should have been possible to upload a task set to a database that could be accessed from TPP. Since the task-creation tool is only used by us, the developers, we argue that such an implementation would fall under the category 'nice to have', and thus be a part of priority 4.

When it comes to starting new games, we managed to implement functionality relating to joining and starting a game. To join a game, a player needs an invite link from us; see requirement 21. A player is then asked to fill in a nickname and select what game lobby they want to join; see requirements 25 and 26. If the player submits an empty nickname field, they are granted a random id, which relates to their peer id from the LioWebRTC library [81]. The player is then presented with a lobby where they can see their connected team members and select a task set, see requirements 20 and 27. Any player in the lobby can start the game on behalf of their team; see requirement 22. However, some user testers expressed that they would prefer if all players had to explicitly confirm that they were ready to start the game. We could probably have reused the lock-in functionality from solving tasks for this purpose.

We completed 13 out of 16 requirements in the 'solving task' category. Players can move code blocks within their own handlist and within the solutionfield according to requirement 30. It is also possible to select and use tab-keys to change solutionfield indentation according to requirements 37 and 39. Players can move code blocks between their handlist and the solutionfield by using drag-and-drop according

to requirement 29. Players can also move code blocks between the handlist and the solutionfield by double-clicking the code blocks; according to requirements 34 and 35. One could argue such extensive code block interactivity was unnecessary. However, according to the user tests, all of these different methods of moving code blocks were requested in earlier iterations and was used intuitively in later iterations.

We decided that each player should be distributed their own code blocks and that the content of the other player's handlists should be hidden and not draggable, see requirements 31, 32, and 33. In the first two user tests, this functionality was highly requested. However, we got mixed responses on this functionality in the third user test. It could be interesting to explore whether the handlists resulted in a better learning outcome or if they facilitated better teamwork. We added hints and clear functionality according to requirements 40 and 41. 'Hint' provides tips on how to solve the given task. 'Clear' moves code blocks from the solutionfield back to the handlists. This functionality was primarily added based on existing solutions. We argue that we could have waited for the user testers to explicitly request these features. However, we did get some feedback on what kind of hints the user testers expected to get. We also added a tutorial within the game according to requirement 36. The tutorial briefly explains the game and how to play it. We argue that this tutorial prevented us from observing whether or not our game concepts were intuitive through use alone.

Based on feedback from iteration three, we decided to implement the player indicator functionality to add more clarity, see requirement 38. This functionality was reused to show the players who are ready to submit a task; see requirements 46-49. A player that is ready to submit needs to lock in their answer, which will prevent them from making additional moves. When all players are locked in, the solution is submitted. The lock-in functionality seemed to work as intended in the final user test. We observed that a discussion emerged when a player locked in, which often led to the remaining players to lock in as well. We also implemented line-based feedback, according to requirements 45 and 50, that showed whether or not a block was placed correctly.

Regarding the non-functional requirements, based on our results, we argue that we created an intuitive and synchronous game according to requirements 57 and 58. Since we are not collecting any data as of now, it is easy to fulfill the privacy requirements, see requirement 59. Furthermore, we created a web-based game for 1-4 players that allows multiple simultaneous games, see requirements 60-63. We also made some efforts to create a layout that can support multiple screen sizes according to requirement 64.

Incompleted requirements

Starting with requirement 5, we did not prioritize optional rules or the number of attempts for a task. We argue that this could confuse a player playing the game for the first time, which would be the case for the user tests. However, it would be interesting to see how optional rules and the number of attempts could influence performance. The same applies to requirement 10, which considers time limits for tasks. Requirement 7 states that TPP should be able to estimate the difficulty of a task. To estimate the difficulty, a function could be implemented to take parameters, such as the number of code blocks, distractors, and programming concepts, to calculate some sort of score. Some established metrics could also be used for code complexity, such as cyclomatic complexity, cognitive complexity, or Halstead volume [91, 92, 93]. Since students tend to struggle with different topics, we argue that the difficulty rating might not be that helpful. For the same reasons, we also dropped requirement 13, which states that a time limit should be calculated from the difficulty of a task.

Requirements 11-12, 14-16, and 23-24 were dropped since we decided not to focus on the instructor's view of the game. Internally, it was enough to edit the JSON files manually, and thus we also dropped requirements 11 and 12. Also, we did not have enough tasks to justify the implementation of requirement 16, which states that TPP should be able to filter tasks based on parameters. We did not include any authentication functionality or databases, as we had no incentive to collect data on student performance for our prototype, see requirements 17-19. We argue that it could be interesting to see what students tend to struggle with if such a game were to be used in a course. We did not find the time to add the option of a team name; see requirement 28. However, a user tester in the second iteration mentioned that a team name could be a good addition to show that a player has to cooperate and not compete with the

other players. In later user tests, we explicitly informed the user testers that they were to cooperate. In a final solution, it might be good idea to add more indication that the game is intended to be cooperative and not competitive.

When it comes to solving tasks, we did not prioritize a built-in text chat or voice chat, see requirement 42, as we did not find it relevant for the user tests. It is worth mentioning that we added the chat requirement based on suggestions from the first and second user tests. We decided that whether or not the game should include or use third-party chat functionality is outside the scope of this project. Additionally, we believe it could be beneficial to be able to highlight and move multiple code blocks at once; see requirements 43 and 44. However, we think that this functionality is less important than the other requirements that fall under the same category.

There is more exploration to be done regarding feedback on solutions. For example, we did not consider consequential errors. We also did not differentiate between the wrong indentation or the wrong index. Additionally, we did not include execution-based feedback or high scores; see requirements 51-56. It would be interesting to observe whether execution-based feedback would change how a team works together, or how competitive aspects could affect motivation and learning outcomes. There is progress to be made with the non-functional requirements as well, including support for various screen sizes and resolutions, see requirement 65. We did not prioritize mobile devices, nor did we add functionality to support touch controls, see requirements 66 and 67. Lastly, we did not include third-party authentication according to requirement 68.

5.3.2 Evaluation of architecture

After evaluating our architecture, we believe we could have spent more time on the design phases before development. There were a few instances where the data flow and communication had to be changed to account for different solutions. While this is to be expected in an iterative process, we feel that the readability and cohesion suffered because of this. Due to the long development cycles, especially in iteration three, we did not take enough time to step back to consider our design choices. Instead, we would make impromptu decisions that we believe decreased the overall code quality over time. We had planned to do a major code refactoring near the end of iteration 4, but ultimately could not due to time constraints. In the end, we believe the component hierarchy is a bit cumbersome to work with. When implementing functionality that includes communication between players, changes must be done across various components. The flow chart in figure 52 shows how the communication was implemented with this design. This became difficult to keep track of when TPP grew in size. Future designs should re-consider how the data flows, in regards to player communication, to avoid the high coupling issue.

We also chose to use a peer-to-peer solution for our game. This choice was based on our initial belief that a peer-to-peer solution would increase performance. Since our initial implementation turned out to be buggy, see chapter 3.4, it could have been better to use a client-server solution. On the topic of performance, it is also possible that we could have benefited from using a game engine. Some game engines include inbuilt support for multiplayer games. However, to what degree a client-server architecture or a game engine would affect performance remains uncertain. It is also possible that there are other solutions for synchronizing the game state across peers that does not require a *player-as-host* solution. We recommend looking into potential alternative solutions for this in future designs.

We intended for TPP to be used in a classroom setting, which requires it to be able to scale up tremendously. For example, in a scenario where close to two hundred students are present, roughly 50 teams would need be created to play together. This would result in 50 simultaneous games, which TPP would have to support. Even though we have only tested on a small scale, with two teams playing simultaneously, we are confident that it should be possible to scale TPP to support as many simultaneous games as needed. Since TPP utilizes peer-to-peer networking, there are no server-bottlenecks that could hinder performance. As stated, our signaling server only serves to match-make peers together, but from there, all communication occurs between the peers directly. Thus we believe scaling TPP to support a classroom exercise is possible. However, further implementation and bug-testing will be necessary before it can be used in a real classroom environment.

A potential drawback of peer-to-peer is that the players must have a good internet connection to pre-

vent the game performance from suffering. Additionally, using peer-to-peer can allow cheating due to manipulation of data, see chapter 2.8.2. We believe this is not as critical because the players would know that cheating only affected them and their own learning potential. However, if leader-boards and scores were to be implemented, it could become an issue, if there were to be some incentive to perform well in the game. These drawbacks must also be considered if peer-to-peer were to be used as the networking communication architecture for future versions.

5.3.3 Evaluation of performance

In chapter 4.4.3, we timed how long it would take to update the board state after moving a code block. Local moves, such as host moves and non-host previews, appear almost instantly. We benchmarked previews and host moves to be 2.5 ms and 4.5 ms, respectively. We believe this performance is acceptable. Two articles claim that a delay of less than 20 ms is considered exceptional [94, 95]. Although these articles probably consider latency as part of a server-client architecture, and not with a local preview, they can give us an idea of whether our delay would appear noticeable to the players. Note that we do not know whether the opinions from these articles align with the average gamer.

In iteration three, before previews were implemented, it would take roughly 435 ms to update the game state after performing a move as a non-host. We deemed this performance to be unacceptable. By implementing previews the performance improved drastically. It is worth noting that TPP still faces the same 435 ms delay when it comes to traffic between two non-hosts. However, since a player cannot distinguish whether a move from another player happened 435 ms in the past or, for example, 100 ms in the past, this issue is not as severe.

We argue that it is more important that the player's own moves appear "smooth". Even though we deem the performance between non-hosts as less critical, we still think half a second of delay is too much. Unfortunately, this delay is mostly caused by network latency. Whether or not some of this delay is caused by the LioWebRTC library itself remains uncertain [81]. We benchmarked the game with a fairly good internet speed; 300 MB upload and 300 MB download. If this game were to be played in an environment with a worse internet connection, the performance would suffer.

Hewlett-Packard's (HP) article deems a latency above 150 ms as “less desirable” [94]. The article from Gamers Guide (GG) deems delay above 300 ms as “unplayable”. GG describes the state as unplayable when “There is a noticeable lag, long delays in response, inconsistent movements, and lost shots. Ping in this range will likely be more frustrating than rewarding.” [95]. Although we agree that the latency is too high, we do not deem the game unplayable, as this delay is relating to updates from other peers and not the player's own moves. Additionally, we argue that latency is less important in problem-solving games, such as TPP, compared to games requiring quick reactions, such as shooters and car racing games. However, we agree with HP that the current delay is less than desirable.

The latency between host and non-host is not as severe, as the traffic only has to go in one direction, whereas non-host to non-host communication is bidirectional. According to the GG article, the game is, in this case, not unplayable, but rather just “poor”. GG describes this state as where “Lags and delays are common but do not completely disrupt gameplay. You're likely to encounter dips in performance and in-game responsiveness.” [95]. We do agree that this description fits our game. For example, if two players are discussing a solution, and one of them requests a block from the other, the other could already have made a move prior to the request. This lag is by no means game-breaking, but it can be frustrating, at least from our own experience.

All in all, TPP's performance was satisfactory to test the research questions, but we argue that the current performance is not good enough to be used as part of a programming course. However, we believe that if the latency was to be reduced to at least 100 ms the game could pass as a real-time cooperation game. According to the HP article, a 100 ms delay would serve as *average* latency, and according to GG, it would serve as a *fair* latency [94, 95].

5.3.4 Choice of technology

In our research, we found single-player Parsons problems software implemented in JavaScript, see chapter 2.3. At the time, we argued that we could save development time by reusing some of the functionality from Js-parsons in our multiplayer game [50]. One could argue that it would have been better to write the code in pure JavaScript. The choice of React was only based on our own preference. However, we believe React worked well for us, as we were able to use various libraries, such as *react-dnd* and *LioWebRTC*, to implement the drag-and-drop functionality and peer-to-peer communication.

A weakness of the game is that it utilizes libraries with known vulnerabilities. These libraries are no longer supported, meaning the vulnerabilities are likely to stay. In our initial requirement specification, we included several security requirements. However, with time constraints in mind, we deemed it unlikely to create a game before the final deadline without using the vulnerable libraries. We argue that this results in a game that only works as a prototype. Therefore, we cannot recommend TPP to be used as a mandatory exercise. However, we argue it could be used as a voluntary exercise in an introductory programming course as the risk of a game crash or other issues would not be as severe.

The *react-dnd* [80] library worked well for implementing drag-and-drop due to its flexibility and ease of use. We can therefore recommend it for future designs that will utilize drag-and-drop functionality. Similarly, *LioWebRTC* [81] worked well for facilitating peer-to-peer communication in TPP. However, as previously stated, a signaling server has to be set up to allow NAT traversal. Fortunately, the developer of LioWebRTC has a solution for this called *SignalBuddy* [82]. Amazon Web Services worked for server hosting, and we can recommend it for any external servers that might be needed.

5.4 Evaluation of project goals

Initially, we planned to test learning outcomes from playing TPP. The plan was to use the same programming exercises to test both cooperative Parsons problems and traditional code writing. We argued that we did not have the time to perform such tests as we would need a substantial amount of data to prove statistical significance. Learning outcome would rely not only on the quality of TPP itself but also on the quality of the provided tasks, whether the tasks are suitable for the student level, and the in-game team composition. Ultimately, we were able to complete our result goals through the creation of TPP and the task-creation tool. However, whether or not we completed our effect goals is up to discussion. Extensive testing would be required to evaluate whether or not we accomplished these goals.

5.5 Limitations

This chapter will discuss some potential limitations of this project, including; the user tests, the results, our execution of the questionnaires, and our literature search.

5.5.1 User tests on the wrong target group

We argue that TPP was not tested by the intended target group; novice programming students. In the first tests in autumn, we had to rely on acquaintances due to the Covid-19 situation. The spring semester presented us with a new challenge: no students were in their first semester of study. We considered the second-semester, first-year, students to be the next best alternative; for user tests 3 and 4. The participants from the second semester turned out to be experienced programmers. 65% of the participants had more than a year of programming experience, according to figure 55. One could argue that high school students in an elective course in programming, currently learning Python, could have been a better alternative. However, our lack of contacts within a high-school organization made it difficult to recruit user testers. Furthermore, testing TPP on high-school students would also have required us, or the high school students, to travel.

In addition, we only tested CPP on students within computer science (CS) and computer informatics systems (CIS). However, Dawson et al. found that non-CS students tend to perform worse than non-

programming students [96]. Most non-CS/CIS students who take introductory courses at our university, NTNU, do so because it is compulsory for their study program. We argue that TPP could benefit these students as well. It could be interesting to test non-CS/CIS students' attitudes towards TPP or similar games.

5.5.2 Risk of misinterpreted data

It might be the case that we recalled a group discussion or an in-game observation incorrectly; since we did not have the time to write down every word and action during the user tests. We could have mitigated these sources of errors with the help of recordings. A recording could also have been helpful for a larger-scale thematic analysis. Unfortunately, we could not collect recordings since they classify as sensitive personal data; and we were too late to request the necessary permissions.

5.5.3 Risk of misinterpreted questionnaires

We base our results on questionnaires, which we consider a weakness. It is not certain that the questionnaires accurately conveyed what the participants thought. Factors such as misinterpretation or fatigue might have affected the answers. Contexts represent another weakness of our questionnaires. For example, we asked a close-ended question regarding *fun*, see figure 28, prior to an open-ended question about socialization, see figure 31. In the question about socialization, we had 12 mentions of the word *fun*. It is possible this prompt about *fun* influenced the participants prior to the open-ended question, which caused an influx of references to *fun*.

5.5.4 Risk of dishonest feedback

We got primarily positive feedback on our game. However, it is possible that the participants wanted to be 'nice' by telling us what they thought we wanted to hear. Our experience is that the participants provided more critique in the anonymous questionnaires compared to the group discussions.

5.5.5 Poor basis for comparison between cooperative and non-cooperative Parsons problems

Figure 56 suggests that the participants were unfamiliar with 'drag and drop' programming exercises. The phrase 'a lot' is quite subjective, in our opinion. However, some of the participants answered that they had no experience at all with drag-and-drop exercises. This might have resulted in a poor basis for comparison between cooperative and non-cooperative Parsons problems. A source of error could be that they had experience with drag and drop exercises but were unfamiliar with the term 'drag and drop exercises.'

5.5.6 Unorganized literature search

As we reviewed other papers during the project, we learned that it is good practice to include a description of how the literature was reviewed. By including our searches, we could have improved reliability. We know for certain that we used the search engine Google Scholar [97]. Unfortunately, as far as we know, Google Scholar does not keep track of search history. However, we argue that this is not a severe issue, as our project did not require a heavy literature search to write a synthesis. Instead, our literature review summarises the concepts and theories that helped us develop TPP and analyze our results.

5.6 Further work

This thesis attempts to provide insight into how novice programming students can benefit socially and academically from a cooperative Parsons problems game; such as TPP. This chapter will present our recommendations for future research paths, as well as what features TPP could benefit from; based on our findings, see chapter 5.1.

Firstly, we recommend testing the learning effectiveness of cooperative Parsons problems games, such as TPP. While the user testers' perception of the learning outcome is optimistic, we recommend that further research empirically tests the learning outcome; to ensure that cooperative Parsons problems have value in education. This could be done by having a group of students take the same pre- and post-test. The learning outcome could then be measured by comparing the performance of the pre- and post test after engaging with different programming exercises. One group could engage with cooperative Parsons problems, while the others engage with other programming exercises, such as single-player Parsons problems and code writing; both individually and cooperatively. It could also be interesting to observe whether students continues to use TPP after being introduced to the game. Long-term use could indicate that the game is valuable, whether it is for socialization, learning or fun.

Next, we will recommend how TPP could be improved further by exploring what features and design aspects of a cooperative multi-user Parsons problems game we believe are important. While there are many options, we believe these points are the most crucial and beneficial at the moment:

1. Improve clarity by displaying all the players' cursors. Many participants expressed that it was difficult to keep track of other players' actions; when there were a lot of simultaneous code block interactions. We believe it would be easier for the players to coordinate if the game displayed their cursors.
2. Implement competitive elements such as scores and leader boards. This could motivate students and discourage player error and behavior such as trial-and-error. The score could be calculated from factors such as correctly submitted tasks, the number of attempts, and the time spent on a task. However, as competition can be demotivating for some students, it should be optional to share the results.
3. Improve the feedback by providing more information on why the code is incorrect and what can be done to fix it. We recommend implementing *execution-based feedback*, see chapter 2.2; as several participants requested it during our user tests. This could also benefit learning, as user testers deemed debugging as an important aspect of programming.
4. Improve performance of the game by decreasing the latency of code block interactions to at least 100 ms. As stated in chapter 5.3.3, a 100 ms delay would serve as a fair latency. While further optimization of the peer-to-peer solution could work, we also recommend looking into a client-server architecture as TPP currently mimics this through the *player as host* design.
5. Ensure that TPP's layout is adaptable to various screen sizes and resolutions. We observed that the usability decreased with smaller screens, as the layout would be set to the vertical layout, which we believe is inferior to our original horizontal design.
6. Expand TPP to make it more suitable for an educational environment. We have a list of requirements, see appendix A, that cover this. This includes: a mode for initiating games automatically for all students in a classroom, implementing a server to access task sets created in the task-creation tool, and providing statistics on student performance. Statistics on student performance can provide instructors with insight about what programming concepts their students tend to struggle with.
7. Consider adding role assignment to TPP; in order to engage all team members. TPP was designed without different roles, as we wanted the game's content to be the focus. However, based on our findings, we theorize that assigning roles to each team member could provide certain benefits, such as a reduction in passivity.

We believe TPP is a good starting point for further research and development. However, as stated in chapter 5.3.4, TPP utilized libraries with known vulnerabilities. Therefore we cannot recommend using

TPP as a mandatory exercise. However, we argue it could be used as a voluntary exercise in an introductory programming course as the risk of a game crash or other issues would not be as severe. We do, however, recommend using the current implementation, and this thesis, as a guideline for how future cooperative Parsons problems games could be designed. Chapter 2.3 also provides some examples of existing single-player Parsons problems solutions that can act as inspiration. Lastly, we can recommend using our cooperative Parsons problems solution for testing the learning outcomes of cooperative Parsons problems games. In fact, we recommend empirically testing the learning outcome prior to investing time and resources into developing a new product.

6 Conclusion

Lack of motivation and difficulties in learning how to program has been linked to dropout and failure rates in higher education. Additionally, loneliness is reported to be a prevalent issue among students in higher education and is linked to decreased learning, motivation, and health. Studies indicate positive learning effects of a code completion exercise called *Parsons problems*. Furthermore, cooperative learning activities have been shown to increase achievements, as well as social relatedness. In this project, we set out to create a cooperative Parsons problem game that can promote learning and socialization in higher education. We called this game *Team-based Parsons problems (TPP)*. The goal of cooperative Parsons problems is to combine the pedagogical advantages of Parsons problems with the social benefits of cooperation. By utilizing cooperative learning and gamification concepts, we developed a game that was tested on novice programming students across four iterations to gather feedback from the target group. Through qualitative and quantitative research, we sought to answer our research questions relating to learning, socialization, how teams work together, and how a cooperative Parsons problems game could be designed.

Our results suggest that students perceived cooperative Parsons problems as an educational, fun, and social activity. The results also indicate that cooperative Parsons problems can facilitate discussion and reflection about programming concepts. However, as we only gathered the students' perception of learning and not measures of actual learning outcome, we cannot make any conclusions on the direct effects of the game in the short- or long term. We are optimistic about our results, but future research should focus on testing the learning effectiveness of a cooperative Parsons problems game; by comparing it to single-player Parsons problems and traditional code writing. We also elicited 11 characteristics of how teams work together to solve Parsons problems digitally; by observing seven teams of students cooperate to solve tasks using TPP. We believe this information can provide insight into how similar cooperative software products can be designed to emphasize cooperative learning, fun, and social aspects. Lastly, we gathered feedback on TPP's design and received a usability rating of *GOOD* according to the System Usability Scale. While the overall feedback was positive regarding the game itself and its design, we also received suggestions on how it could be improved further. We summarised the most important points in chapter 5.6.

Currently, we cannot recommend our cooperative Parsons problems solution to be used in mandatory exercises, as TPP utilizes libraries with known vulnerabilities. Instead, we recommend using the current implementation, and this thesis, as a guideline for how future cooperative Parsons problems games could be designed. Since our user testers showed an overall positive response to using TPP, we can also recommend using it for testing the learning outcome of cooperative Parsons problems. We are optimistic about the potential of cooperative learning through gamification. Hopefully, our findings can contribute to future research within these fields that benefit struggling programming students in higher education.

References

[1] *fun*. URL: https://www.britannica.com/dictionary/fun (visited on 13/05/2021).

[2] *learning*. URL: https://www.britannica.com/science/learning (visited on 13/05/2021).

[3] Dale Parsons and Patricia Haden. 'Parson's programming puzzles: A fun and effective learning tool for first programming courses'. In: vol. 52. Jan. 2006, pp. 157–163.

[4] Yuemeng Du, Andrew Luxton-Reilly and Paul Denny. 'A Review of Research on Parsons Problems'. In: Feb. 2020, pp. 195–202. DOI: 10.1145/3373165.3373187.

[5] *socialization*. URL: https://www.britannica.com/science/socialization (visited on 13/05/2021).

[6] Päivi Kinnunen and Lauri Malmi. 'Why Students Drop out CS1 Course?' In: ICER '06. Canterbury, United Kingdom: Association for Computing Machinery, 2006, pp. 97–108. ISBN: 1595934944. DOI: 10.1145/1151588.1151604. URL: https://doi.org/10.1145/1151588.1151604.

[7] Andrew Petersen et al. 'Revisiting Why Students Drop CS1'. In: *Proceedings of the 16th Koli Calling International Conference on Computing Education Research*. Koli Calling '16. Koli, Finland: Association for Computing Machinery, 2016, pp. 71–80. ISBN: 9781450347709. DOI: 10.1145/2999541.2999552. URL: https://doi.org/10.1145/2999541.2999552.

[8] Aharon Yadin. 'Reducing the Dropout Rate in an Introductory Programming Course'. In: 2.4 (Dec. 2011), pp. 71–76. ISSN: 2153-2184. DOI: 10.1145/2038876.2038894. URL: https://doi.org/10.1145/2038876.2038894.

[9] Ulrich Heublein. 'Student Drop-out from German Higher Education Institutions'. In: *European Journal of Education* 49 (Nov. 2014). DOI: 10.1111/ejed.12097.

[10] Christopher Watson and Frederick W.B. Li. 'Failure Rates in Introductory Programming Revisited'. In: *Proceedings of the 2014 Conference on Innovation & Technology in Computer Science Education*. ITiCSE '14. Uppsala, Sweden: Association for Computing Machinery, 2014, pp. 39–44. ISBN: 9781450328333. DOI: 10.1145/2591708.2591749. URL: https://doi.org/10.1145/2591708.2591749.

[11] Jens Bennedsen and Michael Caspersen. 'Failure rates in introductory programming'. In: *SIGCSE Bulletin* 39 (June 2007), pp. 32–36. DOI: 10.1145/1272848.1272879.

[12] Sara Hooshangi, Margaret Ellis and Stephen H. Edwards. 'Factors Influencing Student Performance and Persistence in CS2'. In: SIGCSE 2022. Providence, RI, USA: Association for Computing Machinery, 2022, pp. 286–292. ISBN: 9781450390705. DOI: 10.1145/3478431.3499272. URL: https://doi.org/10.1145/3478431.3499272.

[13] Henrik Rudi Næss, Sindre Solheim. *Team-based code puzzles, TDT4501 Specialization project*. Student Project Report, NTNU, December 2021. 2021.

[14] Tony Jenkins. 'On the difficulty of learning to program'. In: *Proceedings of the 3rd Annual Conference of the LTSN Centre for Information and Computer Sciences*. Vol. 4. 2002. Citeseer. 2002, pp. 53–58.

[15] Anthony Robins, Janet Rountree and Nathan Rountree. 'Learning and teaching programming: A review and discussion'. In: *Computer science education* 13.2 (2003), pp. 137–172.

[16] Yorah Bosse and Marco Aurélio Gerosa. 'Why is programming so difficult to learn? Patterns of Difficulties Related to Programming Learning Mid-Stage'. In: *ACM SIGSOFT Software Engineering Notes* 41.6 (2017), pp. 1–6.

[17] Jinwoo Kim and F Javier Lerch. 'Why is programming (sometimes) so difficult? Programming as scientific discovery in multiple problem spaces'. In: *Information Systems Research* 8.1 (1997), pp. 25–50.

[18] Dr. Krista M. Kecskemety Brooke Morin. 'Collaborative Parsons Problems in a Remote-learning First-year Engineering Classroom'. In: *ASEE* (2021), pp. 1–10.

[19] Barbara Ericson, James Foley and Jochen Rick.
'Evaluating the Efficiency and Effectiveness of Adaptive Parsons Problems'. In: Aug. 2018, pp. 60–68. DOI: 10.1145/3230977.3231000.

[20] Barbara Ericson, Lauren Margulieux and Jochen Rick.
'Solving parsons problems versus fixing and writing code'. In: Nov. 2017, pp. 20–29.
DOI: 10.1145/3141880.3141895.

[21] Rui Zhi et al. 'Evaluating the effectiveness of parsons problems for block-based programming'. In: *Proceedings of the 2019 ACM Conference on International Computing Education Research*. 2019, pp. 51–59.

[22] Paul Denny, Andrew Luxton-Reilly and Beth Simon.
'Evaluating a new exam question: Parsons problems'.
In: *Proceedings of the fourth international workshop on computing education research*. 2008, pp. 113–124.

[23] *Zoom: Video Conferencing, Cloud Phone, Webinars, Chat ...*
URL: https://zoom.us/ (visited on 10/12/2021).

[24] Marjan Laal and Seyed Ghodsi. 'Benefits of collaborative learning'.
In: *Procedia - Social and Behavioral Sciences 31 (2012) 486 – 490* (Dec. 2012).
DOI: 10.1016/j.sbspro.2011.12.091.

[25] Robert Slavin. 'Cooperative Learning'.
In: *Review of Educational Research* 50 (June 1980), pp. 315–342.
DOI: 10.3102/00346543050002315.

[26] Susan A. Yoon et al. 'Encouraging collaboration an building Community in Online Asynchronous Professional Development: designing for social capital'. In: *International Journal of Computer-Supported Collaborative Learning* (Aug. 2020), pp. 351–362.
DOI: doi.org/10.1007/s11412-020-09326-2.

[27] Patrick Buckley and Elaine Doyle. 'Gamification and Student Motivation'.
In: *Interactive Learning Environments* (Oct. 2014). DOI: 10.1080/10494820.2014.964263.

[28] Raed Alsawaier. 'The Effect of Gamification on Motivation and Engagement'.
In: *International Journal of Information and Learning Technology* 35 (Nov. 2017), pp. 00–00.
DOI: 10.1108/IJILT-02-2017-0009.

[29] Juho Hamari, Jonna Koivisto and Harri Sarsa.
'Does Gamification Work? — A Literature Review of Empirical Studies on Gamification'. In: Jan. 2014. DOI: 10.1109/HICSS.2014.377.

[30] Benedikt Morschheuser, Alexander Maedche and Dominic Walter.
'Designing Cooperative Gamification: Conceptualization and Prototypical Implementation'. In: Feb. 2017. DOI: 10.1145/2998181.2998272.

[31] Erlenawati Sawir et al. 'Loneliness and International Students: An Australian Study'.
In: *Journal of Studies in International Education* 12.2 (2008), pp. 148–180.
DOI: 10.1177/1028315307299699. eprint: https://doi.org/10.1177/1028315307299699.
URL: https://doi.org/10.1177/1028315307299699.

[32] University of Michigan. *Results of the National College Health Assessment*.
https://uhs.umich.edu/files/uhs/NCHA-2018-web.pdf.. Accessed: 2022–26-05. 2018.

[33] Ugur Ozdemir and Tarik Tuncay. 'Correlates of loneliness among university students'.
In: *Child and adolescent psychiatry and mental health* 2 (Nov. 2008), p. 29.
DOI: 10.1186/1753-2000-2-29.

[34] Joyce Pijpers. 'Loneliness among students in higher education: influencing factors'.
In: *A quantitative cross-sectional survey research. Student Health Service, UvA* (2017).

[35] Børge Sivertsen. 'Studentenes helse-og trivselsundersøkelse - Hovedrapport'.
In: *Studentenes helse- og trivselsundersøkelse* (2021), pp. 43–44.

[36] Universities UK. *Student mental wellbeing in higher education: Good practice guide*.
https://www.m25lib.ac.uk/wp-content/uploads/2021/02/student-mental-wellbeing-in-he.pdf.
Accessed: 2022–26-05. 2015.

[37] Renee Kaufmann and Jessalyn I Vallade. 'Exploring connections in the online learning environment: student perceptions of rapport, climate, and loneliness'. In: *Interactive Learning Environments* (2020), pp. 1–15.

[38] Azad Ali and David Smith. 'Comparing social isolation effects on students attrition in online versus face-to-face courses in computer literacy'. In: *Issues in Informing Science and Information Technology* 12.1 (2015), pp. 11–20.

[39] MR Palloff and Keith Pratt. 'Learning together in community: Collaboration online'. In: *20th Annual Conference on Distance Teaching and Learning*. 2004, pp. 4–6.

[40] Konstantina Vasileiou et al. 'Coping with loneliness at University: A qualitative interview study with students in the UK'. In: *Mental Health & Prevention* 13 (2019), pp. 21–30.

[41] Jason C McIntyre et al. 'Academic and non-academic predictors of student psychological distress: The role of social identity and loneliness'. In: *Journal of Mental Health* 27.3 (2018), pp. 230–239.

[42] Kanika Kanika, Shampa Chakraverty and Pinaki Chakraborty. 'Tools and Techniques for Teaching Computer Programming: A Review'. In: *Journal of Educational Technology Systems* 49 (May 2020), pp. 170–198. DOI: 10.1177/0047239520926971.

[43] Scott Leutenegger and Jeffrey Edgington. 'A Games First Approach to Teaching Introductory Programming'. In: *SIGCSE Bull.* 39.1 (Mar. 2007), pp. 115–118. ISSN: 0097-8418. DOI: 10.1145/1227504.1227352. URL: https://doi.org/10.1145/1227504.1227352.

[44] Aparna Chirumamilla and Guttorm Sindre. 'E-Assessment in Programming Courses: Towards a Digital Ecosystem Supporting Diverse Needs?' In: *Digital Transformation for a Sustainable Society in the 21st Century*. Ed. by Ilias O. Pappas et al. Cham: Springer International Publishing, 2019, pp. 585–596. ISBN: 978-3-030-29374-1.

[45] Ville Karavirta, Juha Helminen and Ihantola Petri. 'A Mobile Learning Application for Parsons Problems with Automatic Feedback'. In: Nov. 2012, pp. 11–18. DOI: 10.1145/2401796.2401798.

[46] Andrew Luxton-Reilly Denny Paul and Beth Simon. 'Evaluating a New Exam Question'. In: Jan. 2008, pp. 20–29. DOI: 10.1145/1404520.1404532.

[47] Juha Helminen et al. 'How Do Students Solve Parsons Programming Problems? - An Analysis of Interaction Traces'. In: Sept. 2012, pp. 119–126. ISBN: 9781450316040. DOI: 10.1145/2361276.2361300.

[48] Juha Helminen et al. 'How Do Students Solve Parsons Programming Problems? – Execution-Based vs. Line-Based Feedback'. In: *2013 Learning and Teaching in Computing and Engineering*. 2013, pp. 55–61. DOI: 10.1109/LaTiCE.2013.26.

[49] Stuart Garner. 'An Exploration of How a Technology-Facilitated Part-Complete Solution Method Supports the Learning of Computer Programming'. In: *ECU Publications* 4 (Jan. 2007). DOI: 10.28945/966.

[50] *js-parsons - a JavaScript library for Parsons Problems*. URL: https://js-parsons.github.io (visited on 11/04/2022).

[51] *Parsons Puzzle*. URL: http://parsons.problemsolving.io (visited on 11/04/2022).

[52] *jupyter*. URL: https://jupyter.org/ (visited on 30/05/2022).

[53] American Psychological Association. *APA Dictionary of Psychology*. https://dictionary.apa.org/cognitive-load. Accessed: 2021-09-03.

[54] Zachary Fitz Walter. *What is Gamification?* URL: https://www.gamify.com/what-is-gamification (visited on 13/10/2021).

[55] *What is Gamification?* URL: https://www.biworldwide.com/gamification/what-is-gamification/ (visited on 15/10/2021).

[56] Simone Mora, Francesco Gianni and Monica Divitini. 'Tiles: A Card-based Ideation Toolkit for the Internet of Things'. In: June 2017, pp. 587–598. DOI: 10.1145/3064663.3064699.

[57] Rikke Friis Dam and Teo Yu Siang. *The Design Thinking Methodology*. URL: https://www.impactweek.net/how-we-work/method/?gclid=Cj0KCQjww4OMBhCUARIsAILndv76S%5C%5C2o6fRJhGjFlEHGnoGWVCzFZ2SMy3VMInZ7gkNTyJ6oB6vkQMB8aAiacEALw%5C_wcB (visited on 02/11/2021).

[58] *What is Design Thinking and Why Is It So Popular?* URL: https://www.interaction-design.org/literature/article/what-is-design-thinking-and-why-is-it-so-popular (visited on 02/11/2021).

[59] Rauno Pello. *Design science research — a short summary*. URL: https://medium.com/@pello/design-science-research-a-summary-bb538a40f669 (visited on 02/11/2021).

[60] National Disability Authority. *What is Universal Design*. URL: https://www.universaldesign.ie/what-is-universal-design/ (visited on 22/03/2022).

[61] James Cope. *What's a Peer-to-Peer (P2P) Network?* URL: https://www.computerworld.com/article/2588287/networking-peer-to-peer-network.html (visited on 21/03/2022).

[62] Codrut Neagu. *What are P2P (peer-to-peer) networks and what are they used for?* URL: https://www.digitalcitizen.life/what-is-p2p-peer-to-peer/ (visited on 21/03/2022).

[63] Google Developers. *Real-time communication for the web*. URL: https://webrtc.org (visited on 26/04/2022).

[64] Traci Ruether. *What Is WebRTC? (Update)*. URL: https://www.wowza.com/blog/what-is-webrtc (visited on 21/03/2022).

[65] Tsahi Levent-Levi. *WebRTC Signaling Servers: Everything You Need to Know*. URL: https://www.wowza.com/blog/webrtc-signaling-servers (visited on 21/03/2022).

[66] *Introduction to Network Address Translation (NAT) and NAT Traversal*. URL: https://www.pjsip.org/pjnath/docs/html/group__nat__intro.htm (visited on 21/03/2022).

[67] *STUN*. URL: https://webrtcglossary.com/stun/ (visited on 21/03/2022).

[68] Rhett Roxl. *What is Peer-to-Peer Gaming, and How Does it Work?* URL: https://vgkami.com/what-is-peer-to-peer-gaming-and-how-does-it-work/ (visited on 21/03/2022).

[69] *Measuring and Interpreting System Usability Scale (SUS)*. URL: https://uiuxtrend.com/measuring-system-usability-scale-sus/ (visited on 18/04/2022).

[70] Virginia Braun and Victoria Clarke. 'Using thematic analysis in psychology'. In: *Qualitative Research in Psychology* 3.2 (2006), pp. 77–101. DOI: 10.1191/1478088706qp063oa. eprint: https://www.tandfonline.com/doi/pdf/10.1191/1478088706qp063oa. URL: https://www.tandfonline.com/doi/abs/10.1191/1478088706qp063oa.

[71] Nicole Lazzaro. 'The Four Fun Keys'. In: *Advice from the Experts for Advancing the Player Experience* (2008). DOI: 10.1016/b978-0-12-374447-0.00020-2.

[72] Hounsell Buchinger. 'Guidelines for designing and using collaborative-competitive serious games'. In: *Computers & Education* 118 (Nov. 2017). DOI: 10.1016/j.compedu.2017.11.007.

[73] Ping Zhang. 'Motivational Affordances: Fundamental Reasons for ICT Design and Use'. In: *Communications of the ACM* 51 (Oct. 2008), pp. 145–147. DOI: 10.1145/1400214.1400244.

[74] *affordance*. URL: https://www.merriam-webster.com/dictionary/affordance (visited on 02/11/2021).

[75] *Figma: the collaborative interface design tool*. URL: https://www.figma.com/ (visited on 05/10/2021).

[76] Brian de Haaff. *Minimum Viable Product vs. Minimum Lovable Product.* URL: https://www.linkedin.com/pulse/minimum-viable-product-vs-lovable-brian-de-haaff/ (visited on 21/09/2021).

[77] *React Redux - Official React bindings for Redux.* URL: https://react-redux.js.org (visited on 11/11/2021).

[78] *GitHub: react-beautiful-dnd.* URL: https://github.com/atlassian/react-beautiful-dnd (visited on 28/03/2022).

[79] *GitHub: react-grid-layout.* URL: https://github.com/react-grid-layout/react-grid-layout (visited on 28/03/2022).

[80] *GitHub: react-dnd.* URL: https://github.com/react-dnd/react-dnd/ (visited on 28/03/2022).

[81] *LioWebRTC.* URL: https://github.com/lazorfuzz/liowebrtc/ (visited on 03/11/2021).

[82] *GitHub: signalbuddy.* URL: https://github.com/lazorfuzz/signalbuddy (visited on 28/03/2022).

[83] *Bring it all together.* URL: https://www.netlify.com/ (visited on 26/05/2021).

[84] *Start building aws today.* URL: https://aws.amazon.com/ (visited on 26/05/2021).

[85] *Heroku.* URL: https://www.heroku.com/ (visited on 26/05/2021).

[86] *react-responsive.* URL: https://www.npmjs.com/package/react-responsive (visited on 14/04/2022).

[87] *Profiler API.* URL: https://reactjs.org/docs/profiler.html (visited on 14/04/2022).

[88] *Github: react-simple-code-editor.* URL: https://github.com/satya164/react-simple-code-editor (visited on 21/05/2021).

[89] *Selenium automates browsers. That's it! What you do with that power is entirely up to you.* URL: https://www.selenium.dev (visited on 27/04/2022).

[90] Lawrence Leung. 'Validity, reliability, and generalizability in qualitative research.' In: *Journal of family medicine and primary care* 4 (3 2015), pp. 324–327. DOI: 10.4103/2249-4863.161306. (Visited on 27/04/2022).

[91] *Cyclomatic Complexity.* URL: https://www.geeksforgeeks.org/cyclomatic-complexity/ (visited on 20/05/2021).

[92] *Cognitive Complexity.* URL: https://docs.codeclimate.com/docs/cognitive-complexity (visited on 20/05/2021).

[93] *Halstead Volume.* URL: https://dartcodemetrics.dev/docs/metrics/halstead-volume (visited on 20/05/2021).

[94] Jolene Dobbin. *Lag! Top 5 Reasons your Ping is so High.* URL: https://www.hp.com/us-en/shop/tech-takes/5-reasons-your-ping-is-so-high (visited on 05/01/2022).

[95] *What's a good ping speed for gaming and how to test it.* URL: https://www.cox.com/residential/internet/guides/gaming-performance/ping-testing.html (visited on 05/01/2022).

[96] Jessica Q. Dawson et al. 'Designing an Introductory Programming Course to Improve Non-Majors' Experiences'. In: *Proceedings of the 49th ACM Technical Symposium on Computer Science Education.* SIGCSE '18. Baltimore, Maryland, USA: Association for Computing Machinery, 2018, pp. 26–31. ISBN: 9781450351034. DOI: 10.1145/3159450.3159548. URL: https://doi.org/10.1145/3159450.3159548.

[97] *Google Scholar.* URL: https://scholar.google.com/ (visited on 05/02/2022).

[98] Aharon Yadin. 'Reducing the Dropout Rate in an Introductory Programming Course'. In: 2.4 (2011), pp. 71–76. ISSN: 2153-2184. DOI: 10.1145/2038876.2038894. URL: https://doi.org/10.1145/2038876.2038894.

[99] Linda Isbell and Nicole Gilbert Cote. 'Connecting With Struggling Students to Improve Performance in Large Classes'. In: *Teaching of Psychology* 36 (July 2009), pp. 185–188. DOI: 10.1080/00986280902959960.

[100] Scott Leutenegger and Jeffrey Edgington.
'A Games First Approach to Teaching Introductory Programming'.
In: *Proceedings of the 38th SIGCSE Technical Symposium on Computer Science Education*.
SIGCSE '07. Covington, Kentucky, USA: Association for Computing Machinery, 2007,
pp. 115–118. ISBN: 1595933611. DOI: 10.1145/1227310.1227352.
URL: https://doi.org/10.1145/1227310.1227352.

[101] Andrew Petersen et al. 'Revisiting why students drop CS1'. In: *Proceedings of the 16th Koli Calling International Conference on Computing Education Research*. 2016, pp. 71–80.

[102] Katharina Diehl et al.
'Loneliness at universities: determinants of emotional and social loneliness among students'.
In: *International journal of environmental research and public health* 15.9 (2018), p. 1865.

Appendix

A Requirement specification

This appendix presents the final version of our requirement specification after the four iterations. These requirements are updated to include the new requirements added after iterations three and four based on feedback from our test participants. The requirement specification table has also been updated to reflect the priority changes during development. Initially, we planned that TPP should allow instructors to create new tasks for students to solve. However, we changed this to be an internal task-creation tool to reduce our scope, as we did not have the time to perform user tests on instructors. Therefore, requirements targeting instructors can be seen as obsolete. The requirements are categorized as functional and non-functional. These requirements were then placed in a prioritization table in order of importance.

A.1 Functional requirements

This chapter presents the functional requirements gathered in the requirement elicitation process. They are categorized as 'Creating tasks,' 'Creating task sets,' 'Statistics,' 'Starting new games', 'Solving tasks,' and 'Feedback and submitting solutions.'

Creating tasks

TPP should facilitate the creation of tasks. This functionality was later moved to the task-creation tool. The tasks should have both required and optional fields and settings. A user can choose to either select the content of each code block or automatically let TPP do it for them. The instructor will also have to set a difficulty for the task, but TPP can also suggest this. TPP shall also allow for multiple solutions for the task.

1. TPP shall allow the creation of new Parsons Problem tasks.
2. Each task requires a description, code blocks, and a solution.
3. TPP shall feature a code editor that will split each line into code blocks.
4. Each task can also include distractors, hints, and amount of attempts.
5. TPP shall clarify if there are any optional rules set, such as the number of attempts during the game.
6. TPP shall allow for multiple solutions for a task.
7. TPP shall be able to estimate the difficulty of the task.

Creating task sets

TPP needs to facilitate the creation of task sets for students to solve. This functionality was later moved to the task-creation tool. Task sets should include a time limit. It should be possible to modify a task set either by editing or deleting them. The task creation tool should also support a filtering of task sets based on different parameters.

8. When creating a task set, TPP shall allow choosing from already existing tasks in the system.
9. When creating a task set, TPP shall allow creating new tasks from scratch.
10. Each task in a task set has a required time limit field that has to be filled in by the instructor.
11. TPP shall allow an instructor to edit a task set after it's created using the tool. This includes both tasks and settings.

12. TPP shall allow an instructor to delete a task set using the tool.
13. TPP shall give an estimated time limit for each task based on the difficulty.
14. TPP shall have authentication to make sure only instructors can create tasks and task sets.
15. An instructor shall not be able to edit or delete other instructor's task sets or tasks.
16. TPP shall allow filtering tasks and task sets based on difficulty, amount of tasks, and content within the task.

Statistics

Initially, we planned to collect data on student performance that instructors could use to adjust courses. However, we dropped this as we wanted to focus on the students' views.

17. TPP shall collect statistics on student performance.
18. TPP shall allow instructors to review collected statistics.
19. TPP shall allow an instructor to download statistics locally.

Starting new games

TPP needs to facilitate a lobby system so that instructors and students can create teams to play with. TPP shall not require authentication for play. In order to join games, a player needs to get an invite link. If a player does not have a pre-defined group, they can choose to join a random game without the need for an invite link. An instructor can also host a classroom exercise where all the students present can join the lobby. This will allow the instructor to use TPP in a classroom setting to create teams automatically.

20. TPP shall allow students and instructors to join a game lobby without being authenticated.
21. Players shall join a specific game using an invite link.
22. A player shall be able to start the game on behalf of the whole team.
23. TPP shall allow instructors to host a game lobby without being authenticated.
24. An instructor shall be able to host a classroom exercise where students who join are put into random groups automatically when the instructor initiates the exercise.
25. TPP shall allow players to set a nickname before joining.
26. Players shall be able to select a game lobby to join from a list if not joining from an invite link.
27. A task set must be selected before the game can begin.
28. TPP shall generate a default team name. Any player can change the team name.

Solving tasks

TPP shall have the functionality necessary to solve tasks. This includes moving code blocks and communication tools to facilitate cooperation. The teams can choose to use the in-game voice chat or highlight code blocks to communicate. A player shall only be able to move their own code blocks from their codeblock field to the solutionfield. This is to facilitate cooperation as each player has to be active. When the players are satisfied with their solution, they will submit using a dedicated button. Players can also get hints, as well as clear the board.

29. Players shall be able to move code blocks using drag-and-drop between a handlist field and the solutionfield.
30. Players shall be able to reorder code blocks within the handlist and solutionfield.
31. Each player shall have their own code blocks at the start of the game.
32. A code block's content shall only be visible within a player's own hand list and in the solutionfield.
33. A player can only move their own code blocks from their handlist to the solutionfield.
34. Players shall be able to move a code block from a handlist by double-clicking. This moves the code block to the bottom of the solutionfield.
35. Players shall be able to double-click on code blocks in the solutionfield to move them back to the owner's handlist.
36. TPP shall include a tutorial that can be accessed in the lobby before the game starts.
37. Players shall be able to select and focus code blocks by clicking on them.
38. A selected block shall have an indicator of who has it selected.
39. TPP shall allow indenting in the solutionfield; either through drag-and-drop or by using the tab key on a selected block.
40. Players shall be able to request a hint by clicking the hint button.
41. Players shall be able to clear the solution board by clicking the clear button. This moves each player's code blocks back into their hands.
42. TPP shall have built-in voice chat.
43. Players shall be able to highlight code blocks.
44. Players shall be able to drag-select multiple blocks to be moved in the solutionfield.

Feedback and submitting solutions

TPP will give feedback based on submitted solutions. If a task is not correctly solved within a certain margin of error, they have to redo the task. TPP will also keep track of a score based on performance and the number of attempts to solve each task. The players shall have the option to skip a task if they are stuck. If the time runs out before submitting, the task will be marked as not completed, and the next task in the set will be presented.

45. TPP shall give line-based feedback to all players on the submitted solution by highlighting the correct and incorrect code blocks.
46. All players have to lock in before the solution is submitted.
47. TPP shall indicate which players have locked in.
48. Locking in shall make all code blocks immovable for that player.
49. TPP shall unlock the solutionfield when someone clears the board or when a wrong task is submitted.
50. The team has to redo a task if the team's proposed solution is not approved within a certain margin of error.
51. TPP shall support execution-based feedback for the submitted solution.
52. A team shall be able to receive a score based on their performance, such as attempts, remaining time, and completed tasks.

53. A team should be able to skip a task.

54. Each task-set or class exercise shall have support for a high score calculated by the score from each task.

55. If the time limit runs out before the team has submitted, the current task is marked as 'not completed' and will move on to the next one in the set.

56. TPP shall have a leaderboard with the scores of finished games.

A.2 Non-functional requirements

As TPP is a cooperative game, it has to update all players for each action in real-time. TPP shall be available in the browser and should also be accessible on mobile devices. For authentication, TPP shall use a third-party authentication system in compliance with the university. TPP also has to be scalable in order to facilitate more teams, and more lobbies are necessary. TPP also needs to be secure and comply with GDPR and Norwegian law as it will handle performance data on the students.

57. TPP shall update every player's board when an action is performed, such as moving code blocks, selecting code blocks, and locking in.

58. Students and instructors shall be able to use TPP without any training.

59. Account information and user statistics shall comply with GDPR and Norwegian law.

60. A game shall have 1-4 players.

61. TPP shall be accessible in the browser.

62. TPP shall be scalable to support as many simultaneous games as needed.

63. A classroom exercise lobby shall be scalable to support as many players as needed for any given course.

64. TPP shall update the layout based on screen size.

65. TPP shall support various screen sizes and resolutions.

66. TPP shall be accessible on mobile devices.

67. TPP shall support touch controls.

68. TPP shall support third-party authentication systems used by the university.

A.3 Requirements priority

Requirements are categorized into four priority groups where 1 is the highest priority and 4 is the lowest priority. Functionality linked to how the student used the game was considered the highest priority. Functionality that is not directly linked to the students was considered to be of the lowest priority.

1. **Prototype** : Core functionality necessary to interact with blocks.

2. **Minimum Viable Product (MVP)**: Functionality for students to join games and solve tasks together. MVP is a term used for describing the minimal functionality a product can have and still be used. [76]

3. **Minimum Lovable Product (MLP)**: Functionality that improves the user experience, but are not crucial for solving tasks. MLP is a term used for describing the minimal functionality a product can have and still be loved by the users. [76]

4. **Nice to have**: Functionality that further improves the user experience.

Category	Requirements
Prototype	29, 30, 57, 58, 59, 60
MVP	1, 2, 3, 8, 20, 21, 22, 31, 32, 33, 34, 35, 45, 61, 62
MLP	4, 5, 6, 9, 10, 11, 12, 23, 24, 25, 26, 27, 36, 37, 38, 39, 40, 41, 46, 47, 48, 49, 50, 63, 64
Nice to have	7, 13, 14, 15, 16, 17, 18, 19, 28, 42, 43, 44, 51, 52, 53, 54, 55, 56, 65, 66, 67, 68

Table 14: Requirement prioritization table.

B User tests

This chapter presents the interview guides we followed for each user test in our iterations and the summarised feedback we received. Each user test followed the same procedure and a set of questions. There were also impromptu questions asked in-between the set questions that may not be included in this overview. The summaries were created at the end of each iteration following the user tests. These summaries were used as a basis for the requirement specification phase in the next iteration.

B.1 Iteration 1

Procedure

1. Introduce ourselves and what the user test is for.
2. Ask the participant to think aloud while they are performing the test.
3. Ask the participant to perform the user test for both prototypes.
4. Ask follow-up questions for each prototype - including which prototype they preferred and why.

Questions

- Were you able to interpret what the use case for this program is?
- Are you familiar with Parsons problems?
- Explain: how did you move the code blocks?
- If you wanted to move the code blocks back from the field, how would you go about doing so?
- What do you think [button or icon] means? What would you expect it to do when clicking it?
- How would you want to communicate with your teammates?
- Which design did you prefer, and why?
- Which design was the most intuitive to use, and why?
- Any functionality you thought was missing from these designs?
- Do you have any other input on how the program could be more intuitive, pedagogical, or fun?

Feedback

- Interpretations of design A: the use case was not entirely clear, but they thought it was used to solve some sort of task. A participant thought it was a calculator.
- Interpretations of design B: turn-based, multiple players (compared to design A, which looked like a single-player game), competition between players? (the inclusion of points in the interface), looks like a board game.
- The participants would move the code blocks using drag-and-drop as well as double-clicking to automatically move them.
- Reset button should be renamed to "Clear" as some people thought "Reset" meant to restart the entire game.
- The participants wanted the hint button to give more context and information about the task without giving the answer away.
- It should be clearer that it is a multi-player game (design B emphasized this more than design A).
- Design A seemed more intuitive and simple. Design B looked more like a game and seemed more fun.
- Voice chat seemed like the preferred way for the participants to communicate. One participant suggested having emotes and a pinging system to highlight code blocks with messages.
- Other suggestions from the participants: leaderboard, achievements, and more sound- and visual effects.

B.2 Iteration 2

Procedure

1. Introduce ourselves and what the user test is for.
2. Ask the participant to think aloud while they are performing the test.
3. Ask the participant to solve a task using a low fidelity prototype.
4. Ask follow-up questions once the participant has finished the user test.
5. Provide the participant with a reward for participating.

Questions

- What did you think of the program? Anything that was confusing or not clear?
- How do you think playing with other players would affect how you used the program?
- How would you want to communicate with your teammates?
- Did all the buttons act as you expected?
- What do you think would happen if the time ran out?
- Was it easy to perceive which player you were?
- How should points (score) be handed out? Would a point score motivate you to perform better?
- Would you use this program (in a complete state) with your classmates as a supplementary to learning how to program?
- How would you want to invite or be invited to initiate games?
- Any other input on ways to make the game more fun?

Feedback

- The program was not entirely intuitive for most participants. Some thought it was a competitive game and not a cooperative one.
- Voice chat is preferred over text chat. They preferred if the voice chat was integrated over having to use existing solutions like Discord or Zoom.
- The color scheme seemed child-like. We concluded to update the design to address this.
- Multiple participants tried to double click the code blocks to move them. This reaffirmed that we should allow both dragging and clicking for moving blocks.
- Multiple participants thought the game was turn-based.
- Most participants would prefer an invite link over a game code they would have to type in to join games.
- Points should be handed out based on if the players can solve the task or not.
- The participants agreed that every player should possess their own code blocks.
- Some thought the timer was meant to indicate the time left for the entire task set and not the current task.
- The participants did not want to get all the hints at once when pressing the hint button.
- The participants seemed to be content with the number of features currently featured in the prototype.
- No one got stuck solving the task. Everyone was able to solve the task fairly quickly. Granted, this was a low-fidelity prototype with limited actions available.

B.3 Iteration 3

Procedure

1. Introduce ourselves and what the user test is for.
2. Ask participants to perform a pre-test questionnaire.
3. Explain the application.
4. Ask participants to start the game and cooperatively solve the task set.
5. Ask participants to perform a post-test questionnaire.
6. Have a group discussion to get more feedback from the participants.
7. Provide participants with a reward for participating.

Questions

Pre-test:

- Age and gender
- How well do you know your teammates (context for the communication/cooperation aspect of the user test)(not well - very well) [1-7]
- How do you prefer to work: 1 is invidually and 7 is in a team ? [1-7]
- Do you use a trackpad or mouse for this user test?

- How many years of programming experience do you have? [1-7]
- How much experience do you have with drag-and-drop programming exercises (little experience - a lot of experience)? [1-7]

Post-test:

- How was your overall experience using the program (not good - very good)? [1-7]
- What did you like the best about the program?
- What did you like the least about the program?
- Any frustrating moments while using the program?
- What did you think of the design overall (could use improvement - I liked it)? [1-5]. Elaborate.
- Was the program fun to use (not fun - very fun) [1-5]. Elaborate.
- Was the program easy to understand (hard - easy)?
- Did you find the hints useful (not useful - useful) [1-5]. Elaborate.
- Did you find the feedback on task submission useful (not useful - useful)? [1-5]. Elaborate.
- How was your experience moving the code blocks (not good - very good)? [1-5]. Elaborate.
- How was your experience indenting the code (not good - very good)? [1-5]. Elaborate.
- What did you think of working in a team during this test (did not like - liked it)? [1-7]. Elaborate.
- If you could change one thing about the program, what would it be?
- Any other feedback?

Group discussion:

- What did you think the communication was between you as a team?
- Anything that came in the way of the cooperation?
- Anything that promoted cooperation?
- Were you able to keep track of who did what?

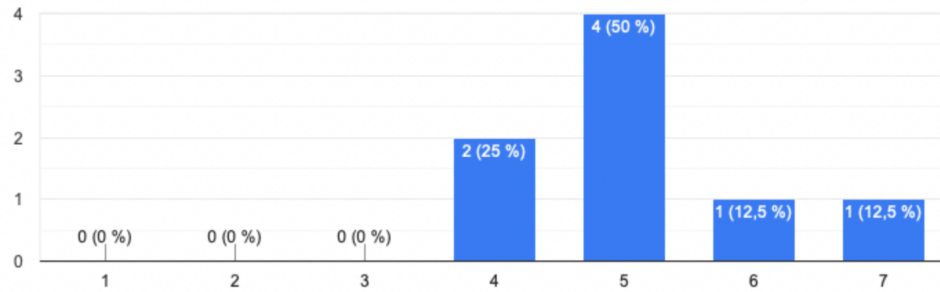
Feedback

- All participants seemed to enjoy playing the game from both observation and from the post-test questionnaire.
- No complaints about the design except for a few screen size compatibility issues.
- The two groups had conflicting opinions on whether they preferred to not see the other player's code blocks or not.
- It was not apparent that the colors of the code blocks indicated their assigned category. It was also not clear that the border color indicated which player owned the block.
- The participants requested that the solution feedback modal should be visible for all when someone submits.
- Some participants requested a system where all players have to lock in the solution before submitting.

- one group pointed out a race condition bug that occurred during the test.
- Some participants thought it was disorienting when multiple players moved blocks simultaneously.
- Some participants forgot certain functionality such as double-clicking, tabbing, and using hints.

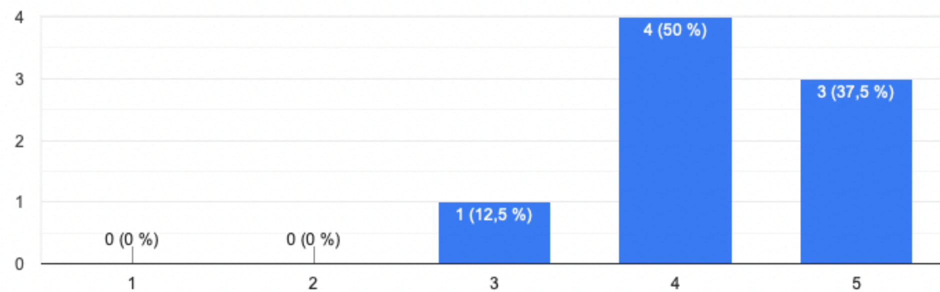
How was your overall experience using the program?

8 svar



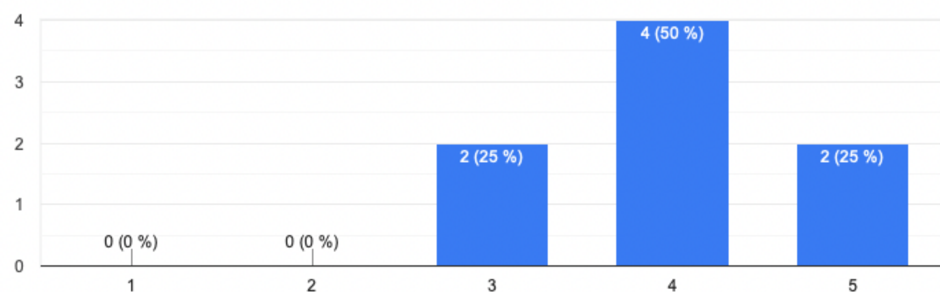
What did you think of the design of the program?

8 svar



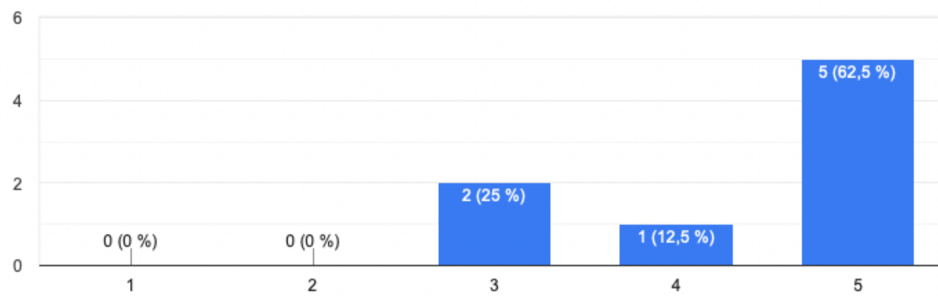
Was the program fun to use?

8 svar



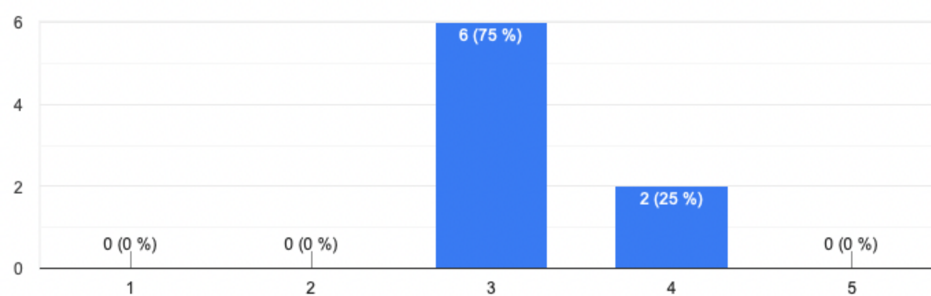
Was the program easy to understand?

8 svar



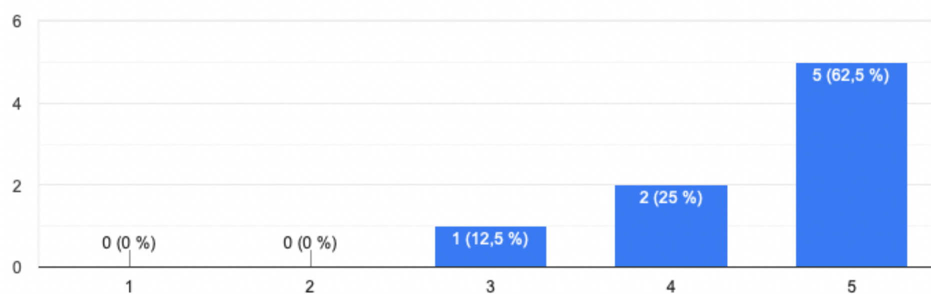
Did you find the hints useful?

8 svar



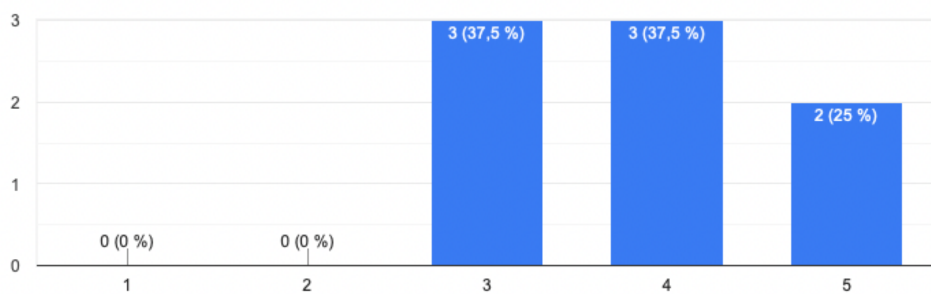
Did you find the feedback on submission useful?

8 svar



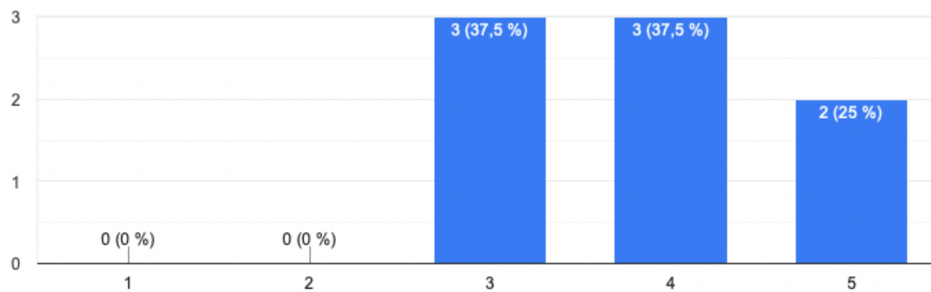
How was your experience moving the code blocks?

8 svar



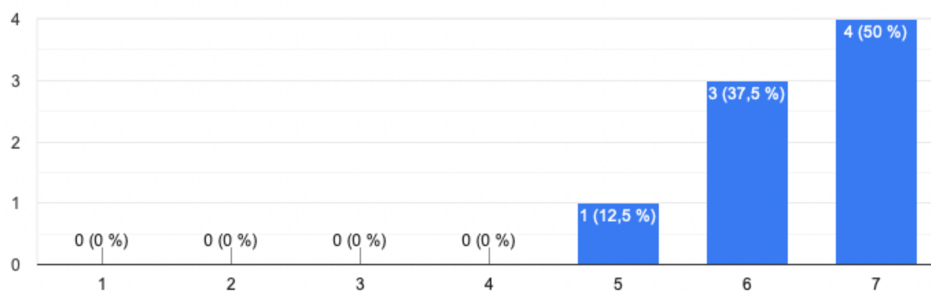
How was your experience indenting the code blocks?

8 svar



What did you think of working in a team during this test?

8 svar



B.4 Iteration 4

Procedure

1. Introduce ourselves and explain the test.
2. Ask participants to perform a pre-test questionnaire.
3. Ask participants to enter TPP's lobby and watch the tutorial.
4. Ask participants to start the game and cooperatively solve the task set.
5. End the game when all tasks have been solved or 15 minutes have passed.
6. Ask participants to perform a post-test questionnaire.
7. Have a group discussion to get more feedback from the participants as well as discuss relevant topics such as learning and cooperation.
8. Provide participants with a reward for participating.

Questions

Pre-test:

- How do you prefer to work: 1 is invidually and 7 is in a team ? [1-7]
- How many years of programming experience do you have? [1-7]
- How much experience do you have with drag-and-drop programming exercises (little experience - a lot of experience)? [1-7]

- Do you prefer programming exercises where you write the code yourself? [yes/no]

Post-test:

- The game was fun to play [strongly disagree - strongly agree]
- I would prefer playing the game alone (single-player) [strongly disagree - strongly agree]
- The game could improve socialization among students [strongly disagree - strongly agree]
- How do you think that this game could impact socialization among students?
- What are your perceptions of potential learning outcomes from this game?
- We also asked the participants to answer the statements from the System Usability Scale (SUS). We present the SUS scale in chapter 2.9.

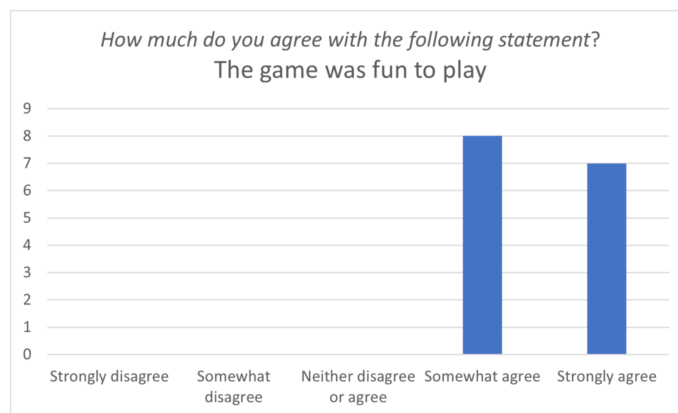
Group discussion:

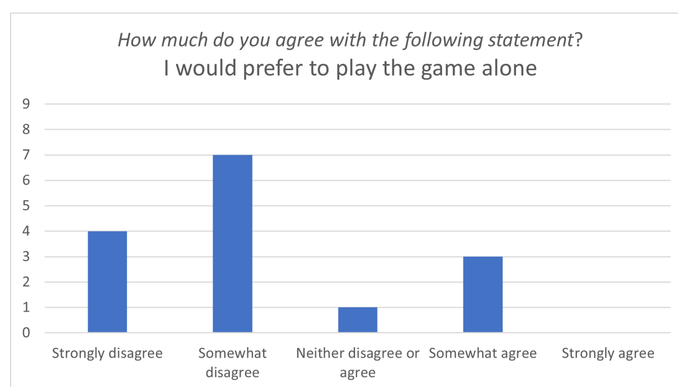
- How do you think this method of learning compares to writing the code yourself?
- What do you think are the strengths and weaknesses of working together in this game?
- How do you think a single-player version of this game could affect learning outcomes?
- How can working together in this game create discussion around programming concepts?
- What do you think of the group's communication?
- Were there some aspects of the game that hindered or enhanced communication?
- What did you like or dislike about the game?
- If you could change one thing about the game, what would it be?
- Was there confusion about design or functionality?
- How could this game be used in an introductory programming course?
- How would teamwork be affected by working digitally?
- Do you have any other feedback for us?

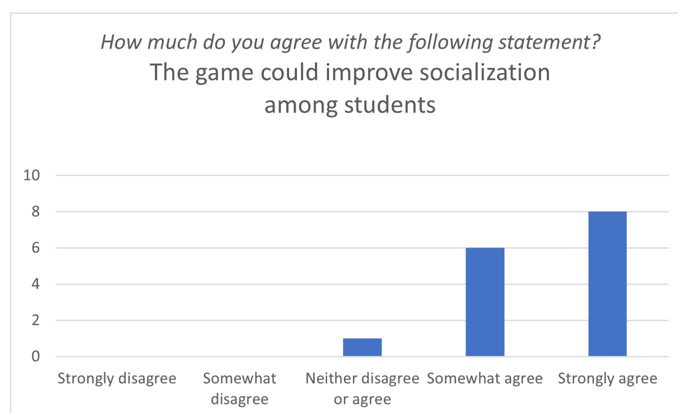
Feedback

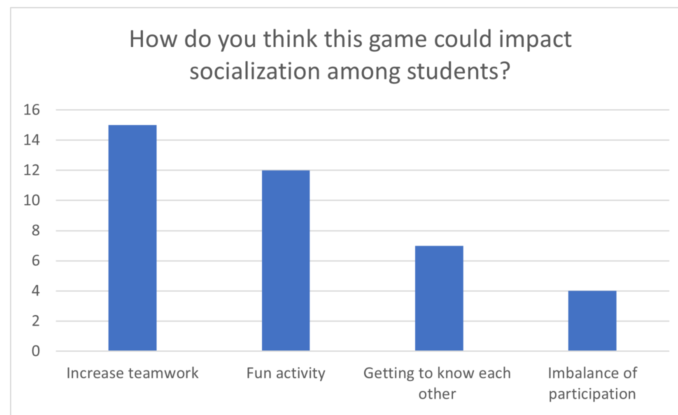
- The participants seemed to enjoy playing the game. They were competitive and expressed that the design overall was simple and intuitive.
- There were no major issues or bugs during the tests.
- Most groups followed the strategy of moving all code blocks into the solutionfield.
- The lock-in system seemed to work well. However, some participants expressed that it was cumbersome to have all players *lock-in* when they had to make small adjustments when re-submitting.
- Player indication seemed to improve clarity to some extent compared to the previous iteration.
- Multiple participants requested execution-based feedback in order to see how the program worked as well as get more direct feedback on how the solution is incorrect.
- The clear functionality was not utilized.
- The hint functionality was utilized, but it did not seem to benefit the participants. Writing good and helpful hints is therefore crucial.

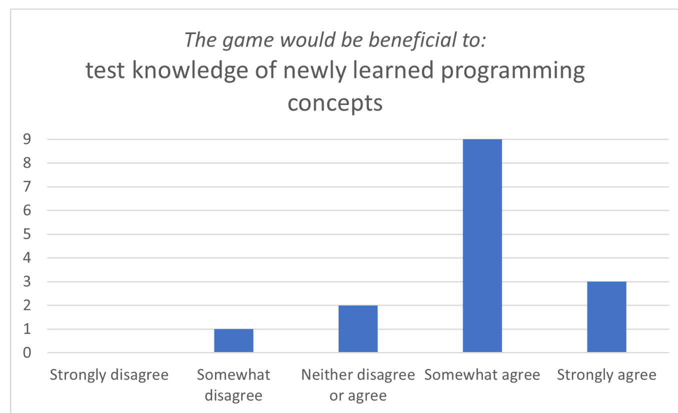
- A majority of the participants complained about some minor synchronization issues when they moved blocks around. This occurred as a result of the preview implementation after the communication was re-designed.
- Some participants expressed the user interface layout was not optimized for their screen size or resolution.
- Some participants requested more competitive elements such as scores and leader-boards.

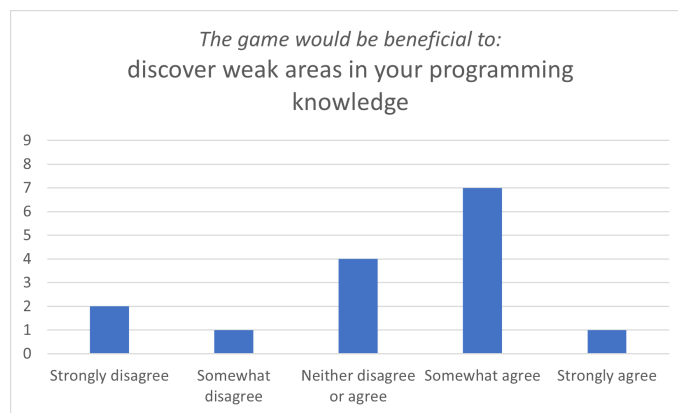


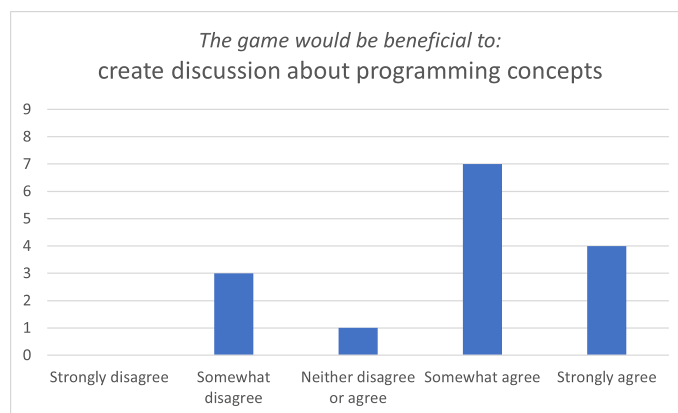


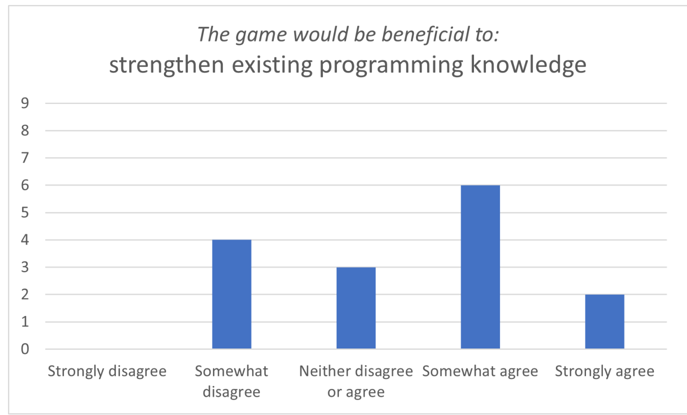


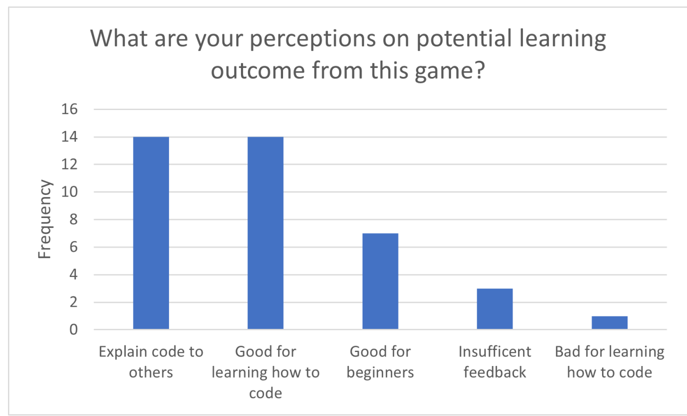

How do you think this game could impact socialization among students?
16
14
12
10
8
6
4
2
0
Increase teamwork
Fun activity
Getting to know each other
Imbalance of participation

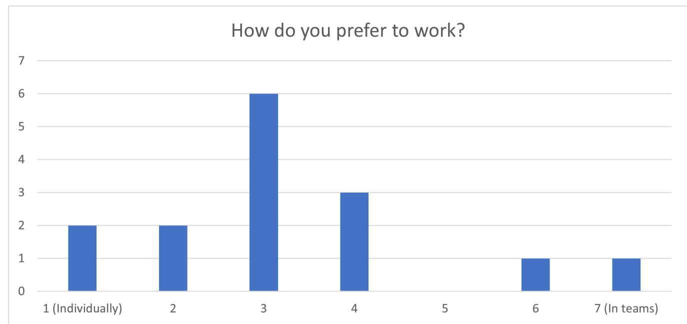

The game would be beneficial to:
test knowledge of newly learned programming concepts
9
8
7
6
5
4
3
2
1
0
Strongly disagree
Somewhat disagree
Neither disagree or agree
Somewhat agree
Strongly agree

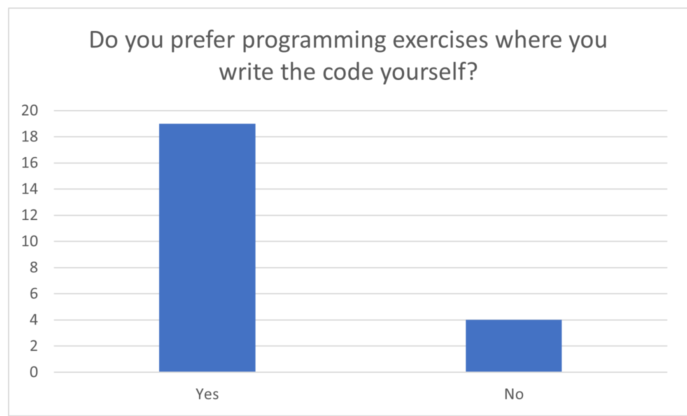

The game would be beneficial to:
discover weak areas in your programming knowledge
9
8
7
6
5
4
3
2
1
0
Strongly disagree
Somewhat disagree
Neither disagree or agree
Somewhat agree
Strongly agree


The game would be beneficial to:
create discussion about programming concepts
9
8
7
6
5
4
3
2
1
0
Strongly disagree
Somewhat disagree
Neither disagree or agree
Somewhat agree
Strongly agree

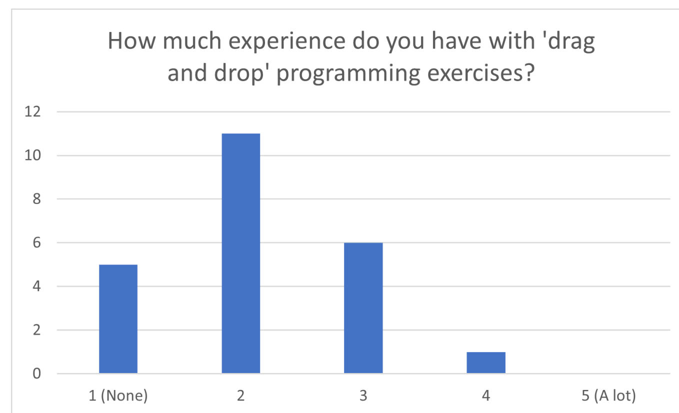

The game would be beneficial to:
strengthen existing programming knowledge
9
8
7
6
5
4
3
2
1
0
Strongly disagree
Somewhat disagree
Neither disagree or agree
Somewhat agree
Strongly agree


What are your perceptions on potential learning outcome from this game?
Frequency
16
14
12
10
8
6
4
2
0
Explain code to others
Good for learning how to code
Good for beginners
Insufficent feedback
Bad for learning how to code


How do you prefer to work?
7
6
5
4
3
2
1
0
1 (Individually)
2
3
4
5
6
7 (In teams)


Do you prefer programming exercises where you write the code yourself?
20
18
16
14
12
10
8
6
4
2
0
Yes
No

How many years of programming experience do you have?
8
7
6
5
4
3
2
1
0
less than a year
1
2
3
4
5
6
more than 7 years


How much experience do you have with 'drag and drop' programming exercises?
12
10
8
6
4
2
0
1 (None)
2
3
4
5 (A lot)

C Prototypes

This appendix shows how the game evolved throughout the different iterations. We performed user tests on each iteration of the game, see appendix B. Feedback from the participants served as the baseline for future iterations.

C.1 Initial sketches

After the feasibility study, the group started sketching possible design solutions. We included some of our initial sketches in figure 57. Henrik proposed a simple interface that built upon a single-player design, see (a) and (b) in figure 57. All players can see all code blocks in (a), and the interface is separated into handout and answer fields. The code blocks can be dragged and dropped within and between the two fields. Additional buttons are placed in the top bar. This design is largely influenced by Js-parsons that we reviewed in chapter 2.3.1. However, in version (b), each player has their own code represented as playing cards. Additional buttons are placed both on the top bar and the left-hand side.

Sindre wanted to create an interface that looked more like a board game. In version (c), the field in the middle represents a table, and the four players 'sit' around it. Players should be able to drag and drop code blocks from their own 'hands' to the table. The player can request hints, reset (clear) the board, and submit the solution. Version (c) includes a score in the left corner and a timer at the top. Version (d) imagines how the game would look if it were to be used for smartphones. We ended up using the task description component from sketch (d).

All the initial sketches included hint, submit, and clear functionality, which we called reset at that time. In retrospect, we can see that a lot of our initial design choices were based upon the Js-parsons library [50]. However, we do not know whether or not this library represents a good design. Another flaw in the sketching phase was that we talked about possible solutions before we started sketching. If we had avoided these conversations, we might have had more creative ideas in this phase. This problem is clearly shown by the fact that both of us included many of the same features. For example, we both included color codes for different types of code blocks, such as variables, functions, conditions, and loops. We also both included functionality for requesting hints, reset (clear) the board, and submitting the solution.

C.2 Prototype 1

After the group had sketched the initial sketches, a meeting was held. In this internal meeting, the group discussed improvements to the sketches. These suggestions were used to make two paper prototypes shown in figure 10 and 11. Both versions were used in the first user test.

C.3 Prototype 2

The second prototype was a digital prototype made with a tool called Figma [75]. The digital prototype had support for a single task that could be solved with drag-and-drop. Figure 58 shows the different features presented in iteration two as screenshots. The game had support for clearing the board, hints, and a submit button. The submit button only had feedback for right or wrong. A chat button with a popup was also included to act as a conversation starter for possible communication tools.

C.4 Prototype 3

Figure 59 shows screenshots of game in iteration 3. This iteration had support for drag-and-drop functionality. A players own code blocks are highlighted in figure 59 (b). These code blocks can be moved into the solutionfield by the player. Other players' code blocks cannot be dragged, and their content appears invisible. Border colors are specific for the player (blue in this case), and background colors are specific for the code content of the block. The blue background represents variables, yellow represents

loops, red represents functions, and the green represents conditions. These colors were chosen arbitrarily, and it was not intuitive for the user testers what these colors represented. We decided to remove the background colors for the next iterations.

Figure 59 (c) highlights the solutionfield. All players can edit the content of this field. The solutionfield also supported intending at this point. The players had to use the provided blocks to solve the task presented in figure (d). If a player attempted to submit the answer in the solutionfield, they would either be presented with figure (g) or (h); based on whether or not they provided a correct solution. Green lines indicate correct lines in figure (g), and red lines indicate incorrect lines. In iteration 3, other players were alerted that a player submitted a correct solution. However, this was changed in iteration 4, where all the players had to submit the solution together.

Figure (c) shows a screenshot of the hint modal. This modal pops up when the player clicks the 'Hint' button. The modal features a text-based hint and the possibility to toggle hints. Figure (f) shows a screenshot of the clear modal. The clear modal asks for a confirmation before it clears the board, meaning that all players get their code blocks back from the solutionfield. Iteration 3 also included a welcoming page where players had to enter a nickname and a lobby number. This welcoming page relates to figure 60 (a), minus the ability to select a room. The lobby relates to figure 60 (b), minus the ability to select a task set. When a team finishes a task set, they are greeted with a congratulation message before they are returned to the lobby.

C.5 Prototype 4

Iteration 4, first and foremost, changed the way TPP communicated. We explain this in detail in chapter 3.4. Interface changes in iteration 4 are mostly limited to the player indicators that we added to show if other team members moved a block, see figure 21. We also made it so that all players had to agree on a task submission, see figure 22 and 23. A player could choose to *lock in* their board, thus limiting them from making any additional move and signaling to the other players that they were ready to submit.

Initially, we wanted to show the cursors of all players in real-time. However, we dropped this, as we feared that this could lead to problems caused by the sheer amount of messages. We also would have had to calculate the position of a cursor between different screen sizes. The player indicators thus seemed like a simpler alternative. Additionally, we also made some minor changes to the lobby in iteration 4. This version of the lobby lets the user select room and task set.

D User test tasks

This appendix presents the tasks the user test participants were asked to solve. The tasks were created with consideration of what topics the students had already learned in novice programming courses at NTNU at the time.

D.1 User test 1 and 2

User tests 1 and 2 utilized low-fidelity prototypes, so we did not create any proper tasks for these tests. As we primarily wanted feedback on the designs, we reused a simple program the users had to construct using code blocks.

```
# Calculate the sum of x and y. Finally, print out the result.

x = 2
y = 3
z = sum(x, y)
print(z)
```

Listing 1: Simple task for user test 1 and 2

D.2 User test 3

```
# Create a function that sums elements in a list and returns the sum.

def sum_list(items):
    sum_numbers = 0
    for x in items:
        sum_numbers += x
    return sum_numbers

# Distractors:
function sum_list(items)
for x in items.length:
```

Listing 2: User test 3 task 1

```
# Iterate the given list of numbers and print only those numbers which are
    divisible by 5

num_list = [10, 33, 55]
for num in num_list:
    if num % 5 == 0:
        print(num)

# Distractors:
if num / 5 == 0:
for num in range(num_list):
```

Listing 3: User test 3 task 2

```
# Write a function to return True if the first and last number of a given list is
    same. If numbers are different then return False.

def first_last_same(numList):
    first = numList[0]
    last = numList[-1]

    if first == last:
        return True
    else:
        return False

# Distractors:
first = numList[1]
last = numList(len(numList))
```

```
15 if first = last:
```

Listing 4: User test 3 task 3

```
1 # Write a program to find how many times substring "Emma" appears in the given
    string.
2
3 str1 = "Emma is a good writer."
4 str2 = "Emma is a great developer."
5 str = str1 + str2
6
7 answ = str.count("Emma")
8 print(answ)
9
10 # Distractors:
11 answ = "Emma".count(str)
12 answ = "Emma".get(str)
13 answ = str.get(Emma)
```

Listing 5: User test 3 task 4

```
1 # Write a function to remove characters from a string starting from zero up to n
    and return a new string.
2
3 def remove_chars(word, n):
4   x = word[n:]
5   return x
6
7 # Distractors:
8 function remove_chars(word, n)
9 x.pop(n)
```

Listing 6: User test 3 task 5

D.3 User test 4

```
1 # Create a function that sums elements in a list and returns the sum.
2
3 def sum_list(items):
4     sum_numbers = 0
5     for x in items:
6         sum_numbers += x
7     return sum_numbers
8
9 # Distractors:
10 function sum_list(items)
11 for x in items.length:
```

Listing 7: User test 4 task 1

```
1 # Iterate the given list of numbers and print only those numbers which are
    divisible by 5
2
3 num_list = [10, 33, 55]
4 for num in num_list:
5     if num % 5 == 0:
6         print(num)
7
8 # Distractors:
9 if num / 5 == 0:
10 for num in range(num_list):
```

Listing 8: User test 4 task 2

```
1 # Write a function that merges two lists of numbers, such that the new list should
    contain odd numbers from the first list and even numbers from the second list.
2
3 def merge_list(list1, list2):
4     result_list = []
5
6     for num in list1:
```

```
            if num % 2 != 0:
                result_list.append(num)
        for num in list2:
            if num % 2 == 0:
                result_list.append(num)
        return result_list
```

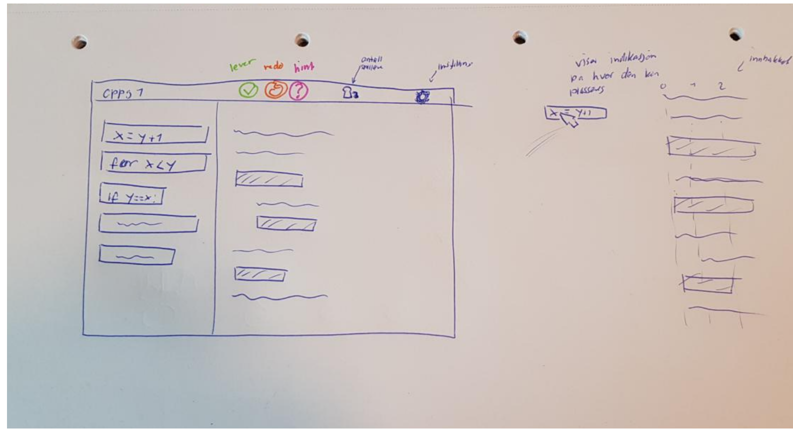
Listing 9: User test 4 task 3

```
# Create a program that asks for a positive integer n. The program then prints out
    the first n numbers in the fibonacci sequence. The first two numbers in the
    fibonacci sequence is 0 and 1. The next number is obtained by adding the
    preceding two terms.

n = int(input())
count, n1, n2 = 0, 0, 1
if n == 1:
    print(n1)
else:
    while count < n:
        print(n1)
        nth = n1 + n2
        n1 = n2
        n2 = nth
        count += 1
```

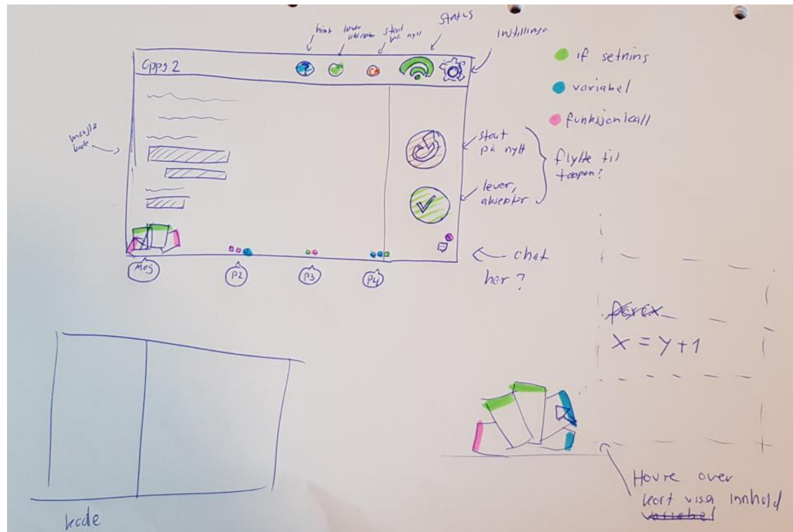
Listing 10: User test 4 task 4

```
# Write a function to check if the given number is a palindrome number. A
    palindrome number is a number that is same after reverse.

def palindrome(num):
    original_num = num

    reverse_num = 0
    while num > 0:
        reminder = num % 10
        reverse_num = (reverse_num * 10) + reminder
        num = num // 10

    if original_num == reverse_num:
        return True
    else:
        return False
```

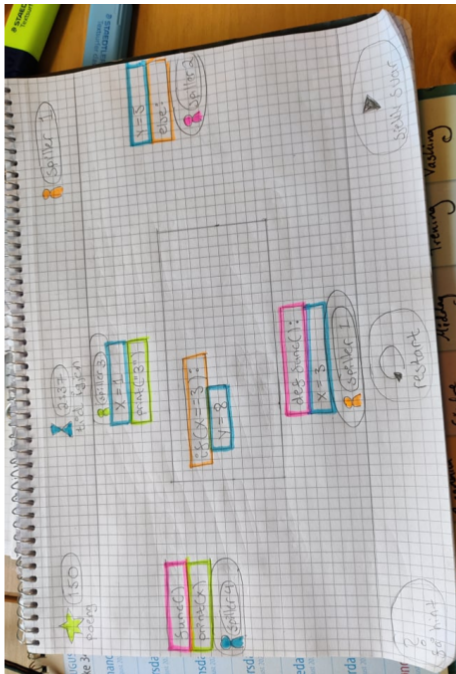
Listing 11: User test 4 task 5



(a) Henrik's initial sketch



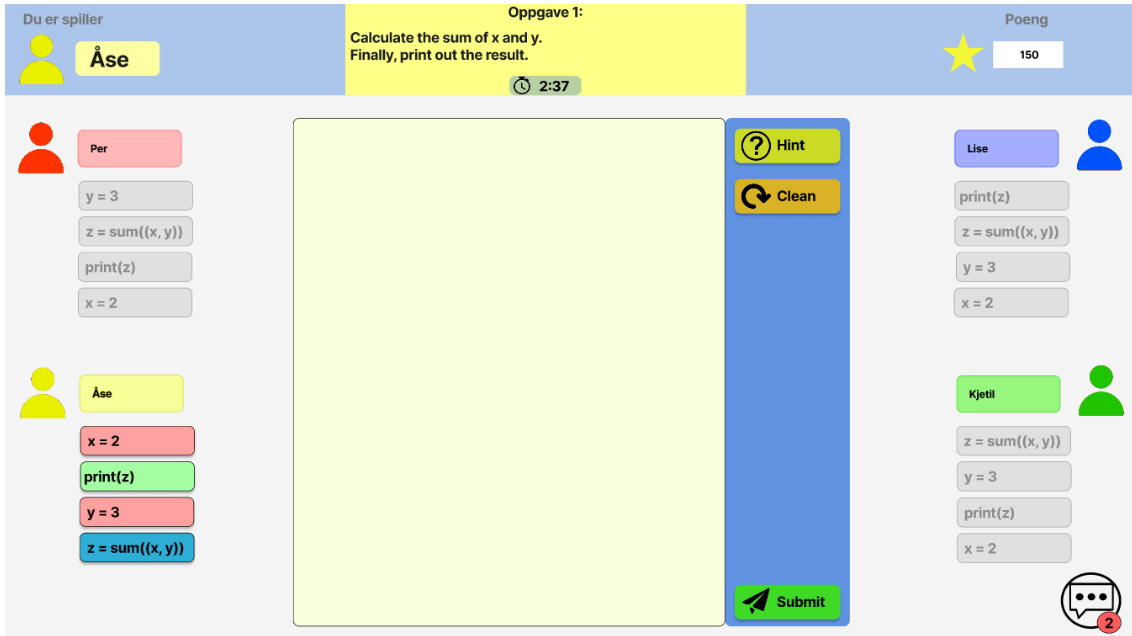
(b) Henrik's initial sketch



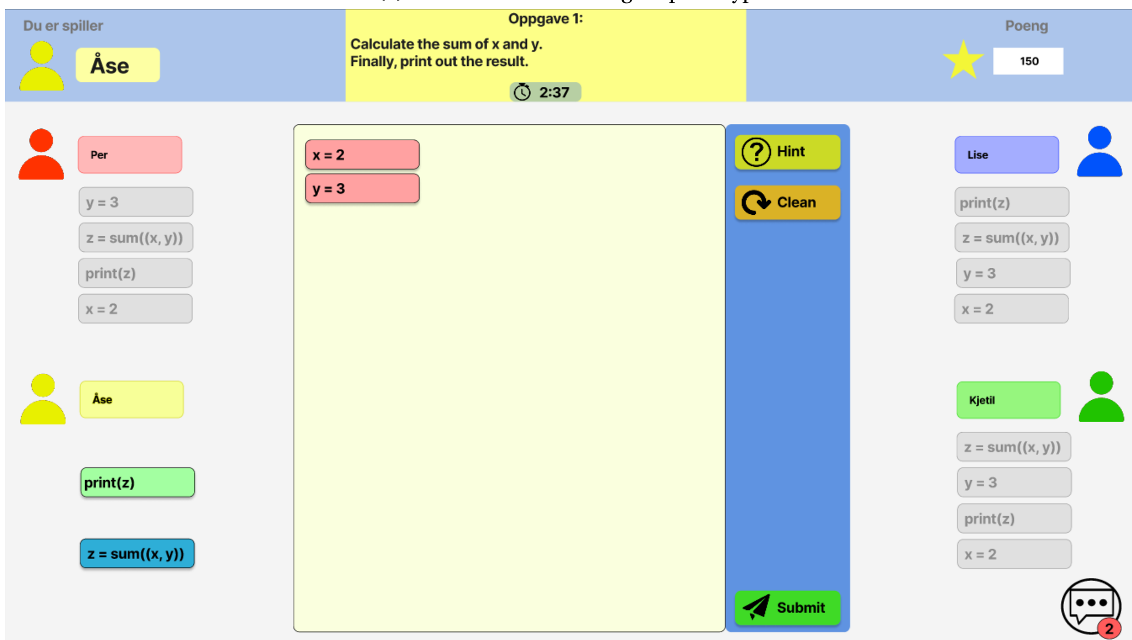
(c) Sindre's initial sketch

(d) Sindre's initial sketch

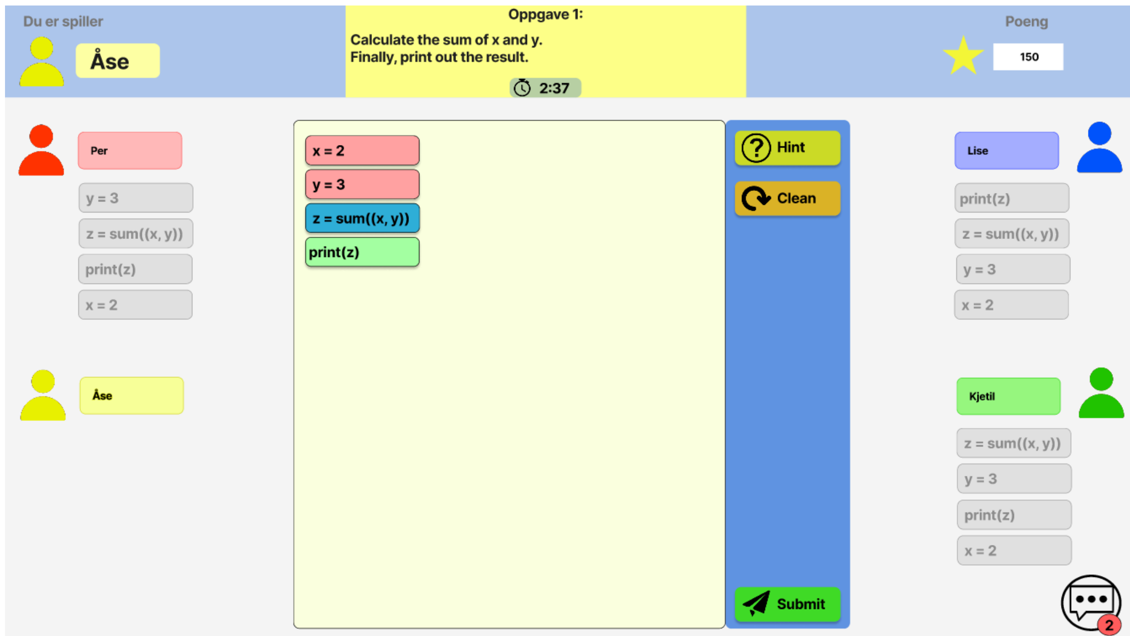
Figure 57: Initial sketches for TPP



(a) Start screen for the digital prototype

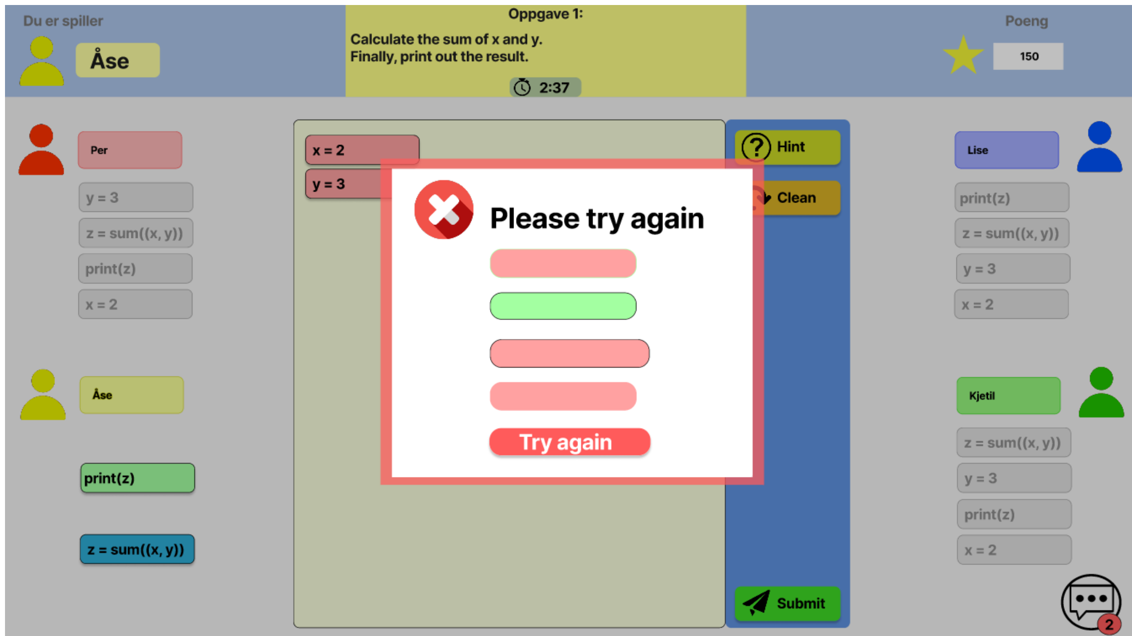


(b) The player have used drag-and-drop to move two code blocks from their hand to the solutionfield

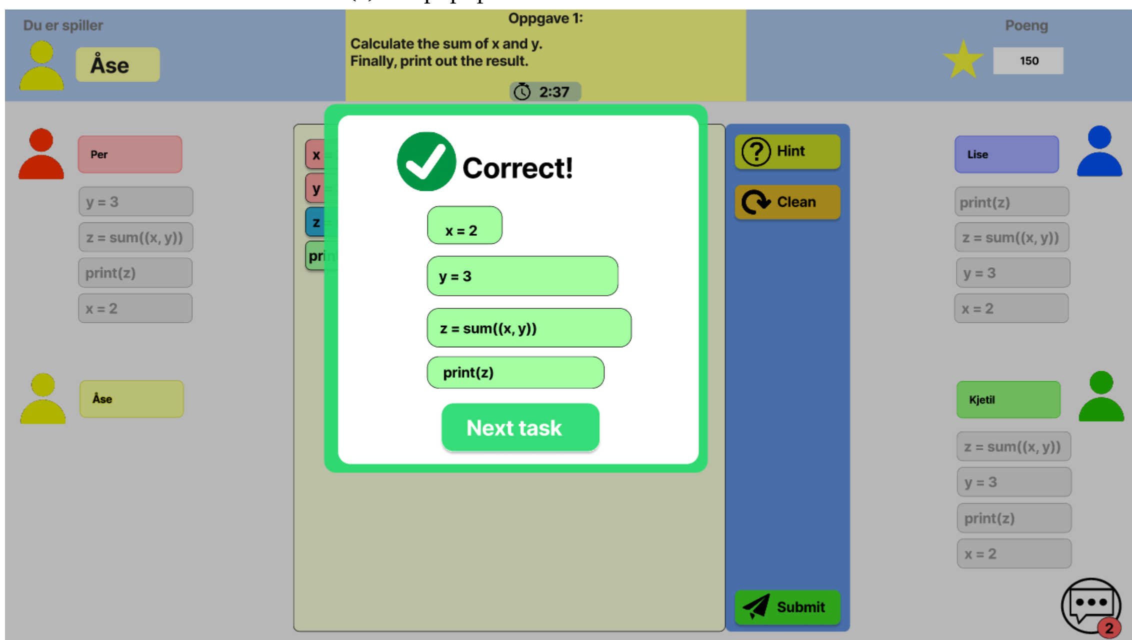


(c) The correct solution state. Submit would at this state tell the user that they have found the correct solution.

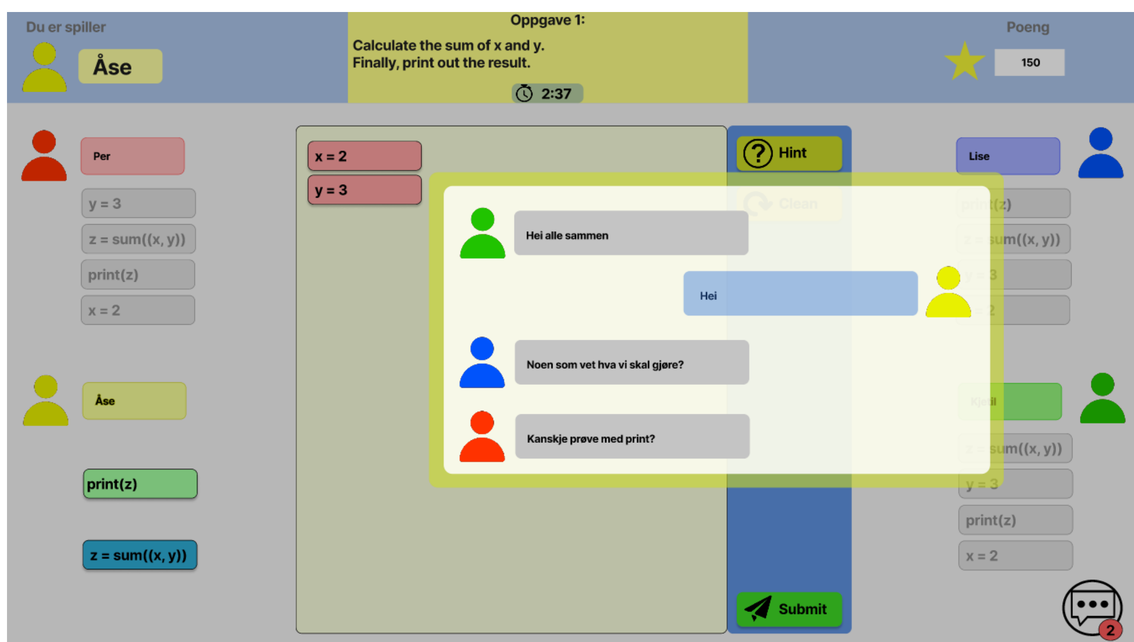
(d) The popup related to the hint button.



(e) The popup related to an incorrect submit.

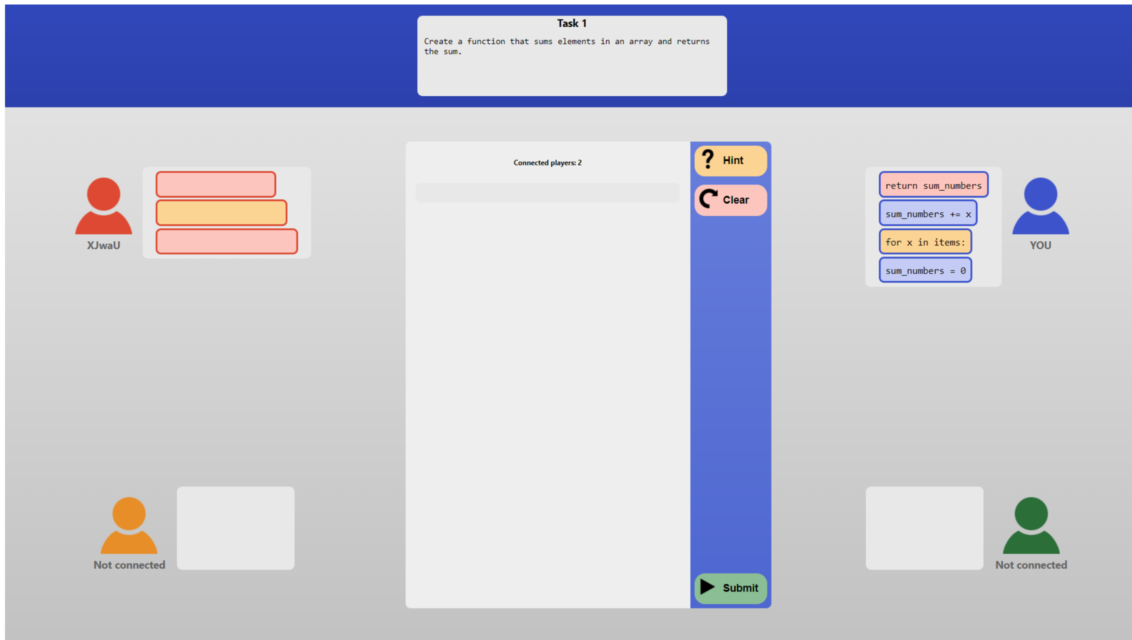


(f) The popup related to a correct submit.

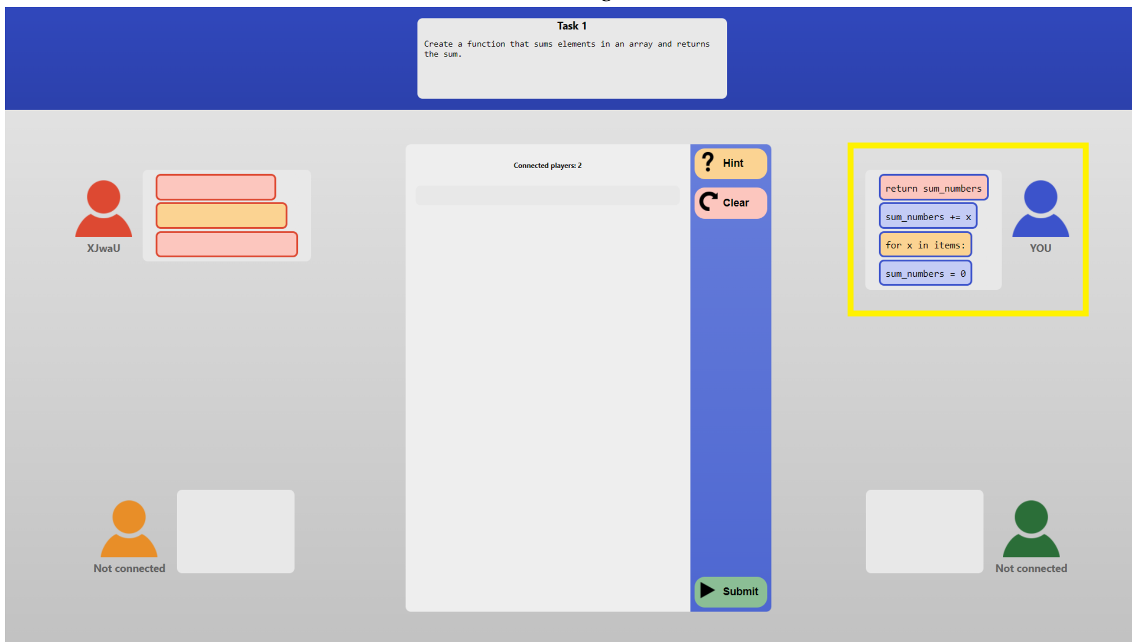


(g) The popup related to the chat button.

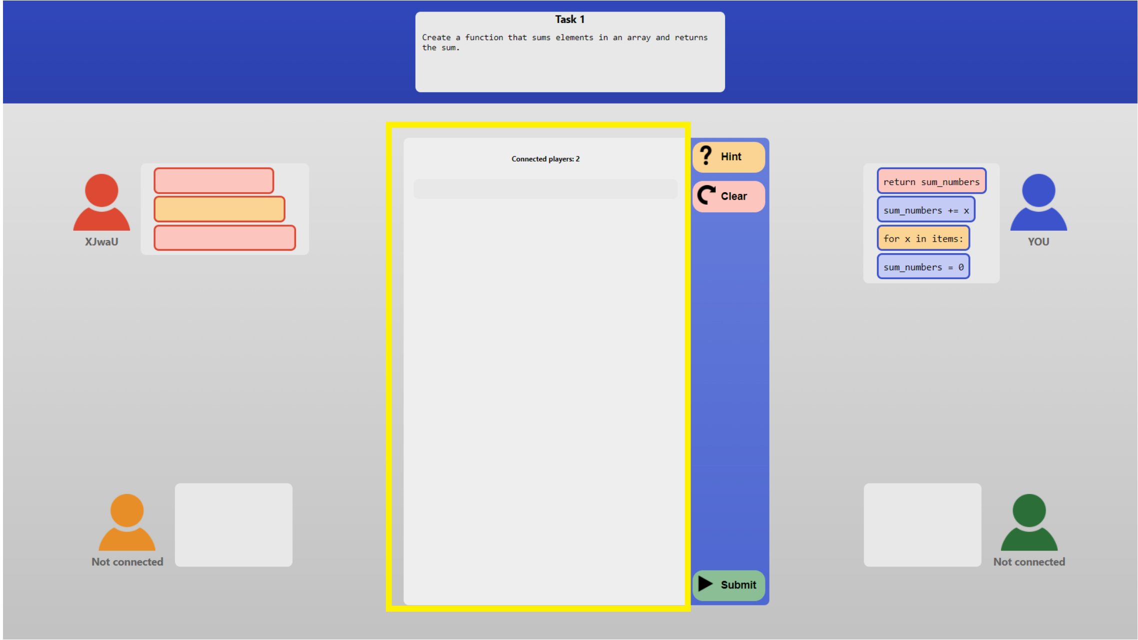
Figure 58: Figma prototype, second iteration of TPP



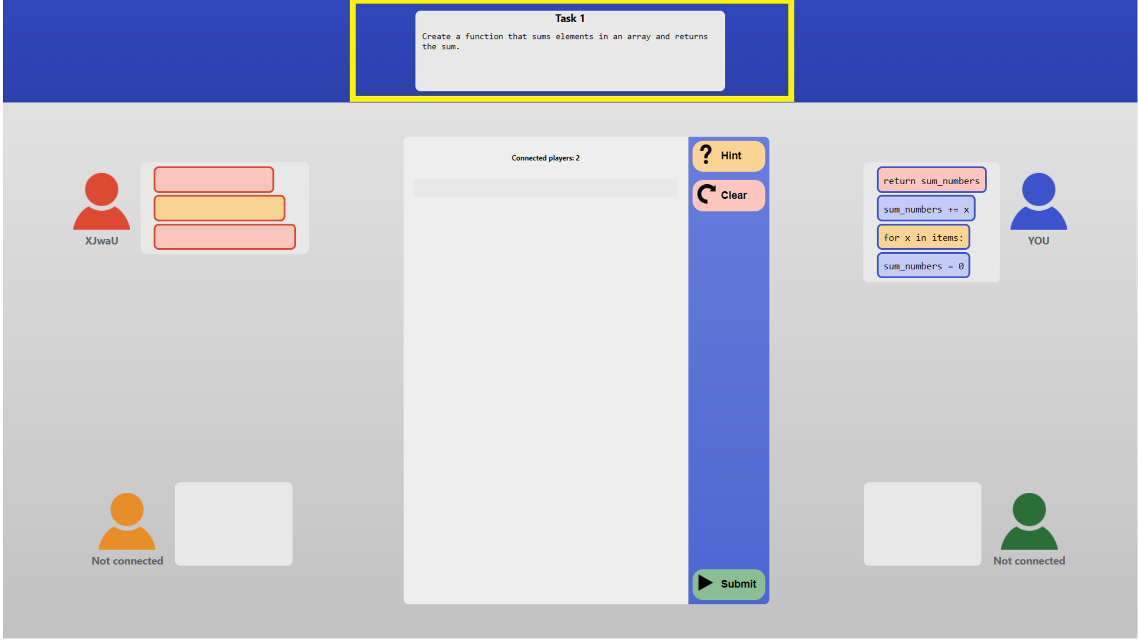
(a) Start screen for the game, iteration 3



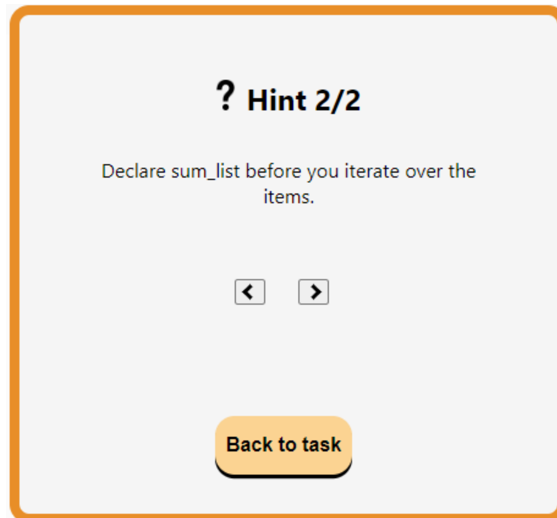
(b) Screenshot that highlights a player's own code block.



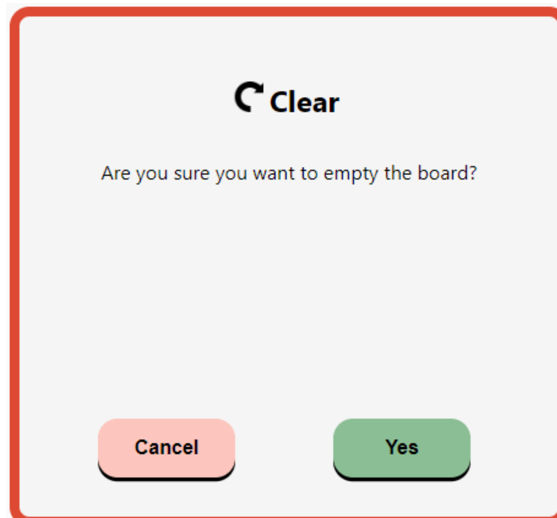
(c) Screenshot that highlights the solutionfield.



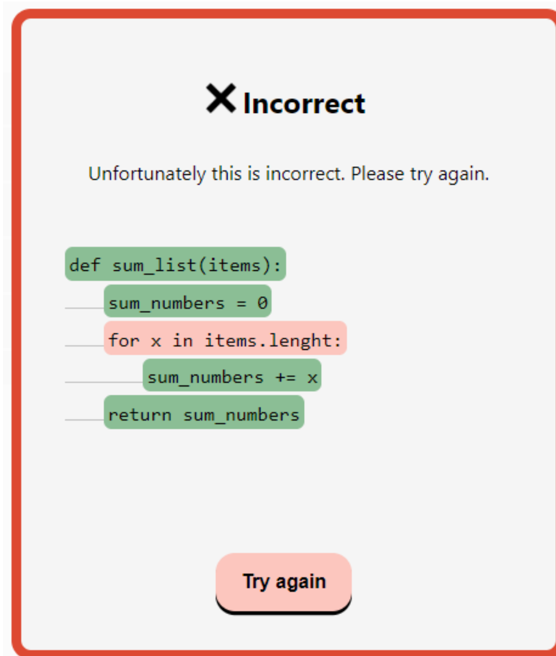
(d) Screenshot that highlights the task bar. This component displays the task description together with the current task number.



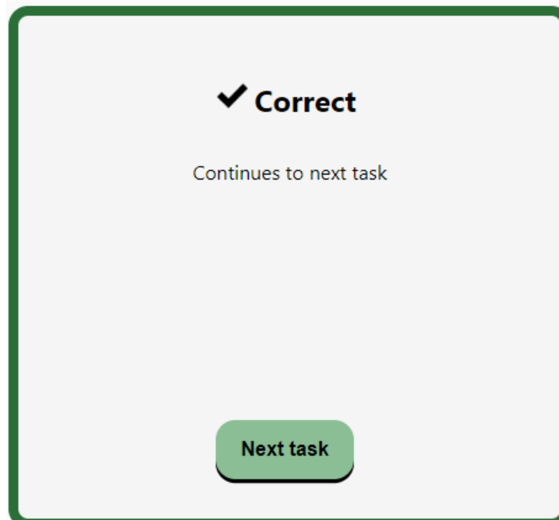
(e) Screenshot of the hint modal.



(f) Screenshot of the clear modal.

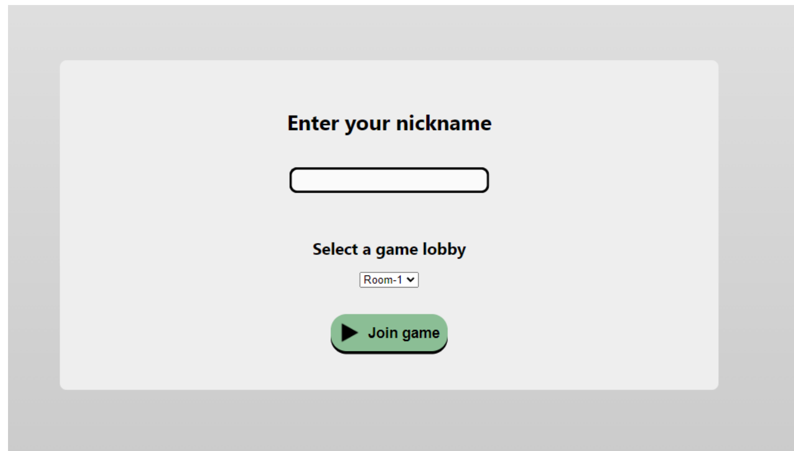


(g) Screenshot of the incorrect modal; which appears after submitting a solution in the wrong state.

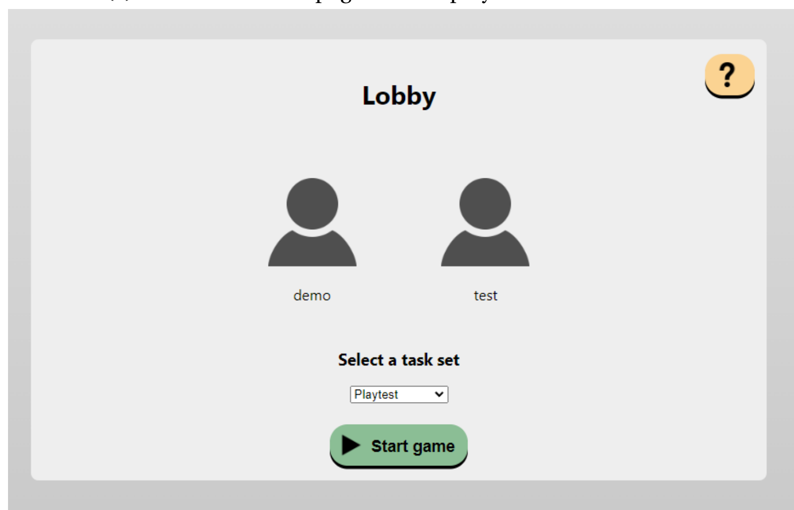


(h) Screenshot of the correct modal; which appears after submitting a solution in the correct state.

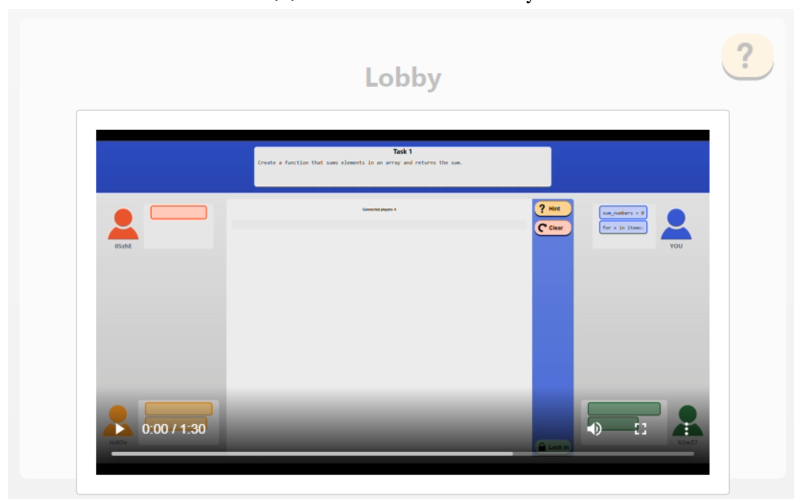
Figure 59: React prototype, third iteration of TPP



(a) Screenshot of the page that the players would connect to.

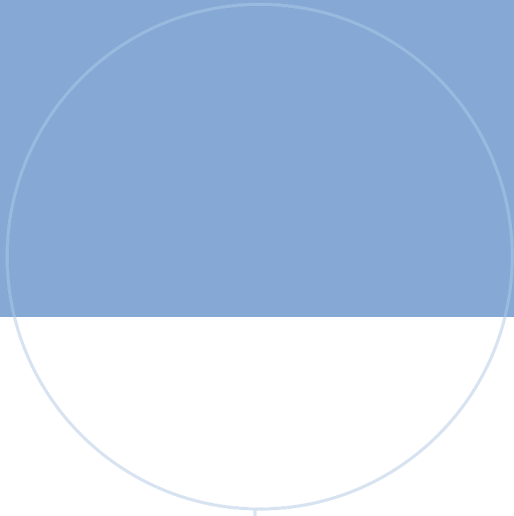


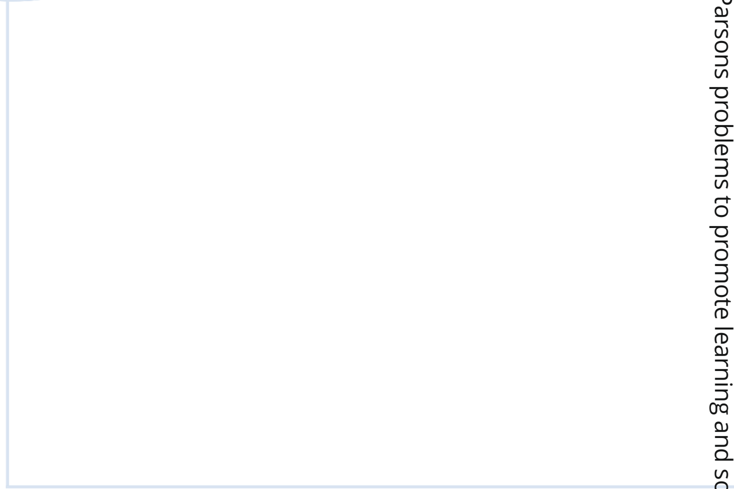
(b) Screenshot of the lobby.



(c) Screenshot of the tutorial modal. This modal would appear if the player pressed the '?' but

Figure 60: TPP's lobby in iteration 4





Solheim & Næss

Cooperative Parsons problems to promote learning and socialization in higher education

NTNU
Norwegian University of
Science and Technology