



Universiteit  
Leiden  
The Netherlands

# Opleiding Informatica

Training Logical Operations  
through Gameplay

Marije Faas

Supervisors:

F.F.J. Hermans & G. Barbero

BACHELOR THESIS

Leiden Institute of Advanced Computer Science (LIACS)

[www.liacs.leidenuniv.nl](http://www.liacs.leidenuniv.nl)

22/06/2021

## Abstract

Logical operators are essential to programmers for writing Boolean expressions in control-flow statements. These are fundamental components to understand conditional logic, which is in turn an essential skill for programming. Novice programmers are often expected to learn this skill along the way, through trial and error, or through traditional logic exercises. In this thesis, I will present a modern and engaging way to practice logical operators through a simple mobile game. By giving students a solid understanding of logical operators and their role in writing logical expressions, I argue that they will be much better prepared to start programming. This thesis will describe how the first ideas for the game came into place and how a subsequent prototype was developed. The methods and results of the user testing will be discussed as well as the resulting improvements that were implemented in the final prototype.

The final results show that users starting with no knowledge of logical operators can correctly answer relevant test questions after playing the prototype and the majority reports that they would play the game over doing traditional exercises. Additionally, most of the planned game features were positively received by users. This shows that the game has potential when considering new methods of teaching logical operators to novice programmers or anyone else that is interested in learning conditional logic.

## Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Logical Expressions . . . . .	1
1.2	Current Solutions . . . . .	2
1.3	Problem Definition and Research Question . . . . .	2
<b>2</b>	<b>Related Work</b>	<b>3</b>
2.1	Computational Thinking . . . . .	3
2.2	Game-based Learning . . . . .	3
<b>3</b>	<b>Design Process</b>	<b>5</b>
3.1	Brainstorming Phase . . . . .	5
3.2	First Prototype and User Testing . . . . .	11
3.3	Second Prototype and Concept Testing . . . . .	16
<b>4</b>	<b>Discussion and Future Work</b>	<b>22</b>
4.1	Evaluation of the Design Process . . . . .	22
4.2	Discussion of Results . . . . .	22
4.3	Future Work . . . . .	24
	<b>References</b>	<b>25</b>

# 1 Introduction

In this introduction, logical expressions and their common operators will be explained, as well as their role in computer science. I will describe some traditional ways on how students learn about logical expressions and how they are practiced. I will then explain how and why this process can be improved by designing a mobile game. In the related work section, I provide the scientific background on the role logic plays in computational thinking, and consequently, in programming. Furthermore, game-based learning and its benefits are explained.

The rest of this thesis will focus on the steps that were taken during the design, development and subsequent testing of the mobile game that is the result of this project. Lastly, in the discussion I will evaluate the design process, discuss the results of the concept testing and describe possible future steps to improve the game.

## 1.1 Logical Expressions

A logical expression, or more commonly ‘Boolean expression’ in computer science, is a statement that evaluates to either true or false (equivalently 1 or 0). Two expressions can be combined using logical operators, the most common ones being AND, OR and NOT. An easy way to show the effect of these operators on the outcome of a logical expression is using the so-called ‘truth tables’.

a	b	a AND b
true	true	true
true	false	false
false	true	false
false	false	false

(a) Truth table for AND

a	b	a OR b
true	true	true
true	false	true
false	true	true
false	false	false

(b) Truth table for OR

a	NOT a
true	false
false	true

(c) Truth table for NOT

Table 1: Truth tables for common logical operators

As shown in *Table 1*, the AND operator only evaluates to true when both its operands (in the example, **a** and **b**) are also true. On the other hand, the OR operator only requires one operand to be true. The NOT operator simply ‘flips’ the logical state of its operand. In the examples above, the operands are represented as **a** and **b**; however, they could be replaced by another logical expression, creating much more elaborate expressions. For example, replacing **a** from *Table 1a* with **a OR c** and **b** with **NOT b** gives us the expression **a OR c AND NOT b**.

Looking at this expression might raise the question: which of these operators should be evaluated first? Similarly to arithmetic operators (addition, subtraction, multiplication, division), logical operators also have an order of precedence. Of the three operators that have been discussed, NOT has the strongest precedence, followed by AND, and lastly OR. Therefore the example above is equivalent to **a OR (c AND (NOT b))**.

When programming, logical operators are extremely important. They are widely used to test certain conditions and make corresponding decisions using the AND, OR and NOT operators. As a result, you will mainly find them in binary control-flow statements (such as ‘if’ and ‘while’).

This can be something very simple like running a piece of code when conditions **a** and **b** are both true (**if (a AND b)**) but it can get very complex when many conditions need to be tested. Examples are long expressions in if-statements or chains of if-else statements.

It's clear that logical operations are essential to programming, and therefore when students learn to program they also need to become comfortable with logical operations. The common ways to teach this, using traditional learning techniques, are discussed in the next section, followed by my proposed method to teach these skills.

## 1.2 Current Solutions

Since programming involves logical operations, students will likely be introduced to them when they learn about if-statements and later encounter it during while-statements as well. Standard teaching during Computer Science classes usually involves lectures, which will go through the theory, followed by practical programming exercises to get the students familiar with the discussed material. Most of the time this means that the focus is on programming and students are simply expected to become familiar with logical operators when writing if-statements over time.

Another way that students might be introduced to logical operations is through a dedicated logic course, as logic is often part of the curriculum of Computer Science programs. In logic, logical operators are more commonly known as logical connectives and they include the ones discussed earlier: logical negation (NOT), logical conjunction (AND) and logical disjunction (OR). During logic courses, the theory is often learned through repeated exercises.

On the other hand, people that want to learn programming while not doing a Computer Science program might instead find the information they need on the internet, using websites like W3Schools. In this case, they will likely read the information that can be found online and try to apply it themselves while programming, similarly to the first scenario.

## 1.3 Problem Definition and Research Question

While all the different methods of learning logical operations mentioned in the previous section work, learners engagement can be improved. The learning process is positively influenced when students are engaged with the material, making it worthwhile to increase engagement while learning. Moreover, instead of trying to figure out increasingly complicated logical expressions while also learning to program, it could be beneficial to first get comfortable with logical expressions before diving into programming. This way, the focus can be shifted to relevant programming concepts and less time is spend figuring out why a particular if or while statement isn't doing what it should.

Rather than studying the use of traditional and repetitive exercises, this thesis will describe a different approach using game-based learning. The aim is to design a mobile game that will allow students with no prior knowledge in programming or logic to practice solving logical expressions involving the three common logical operations (AND, OR, NOT). With an emphasis on fun and engaging gameplay, I argue that they will be much better prepared to start programming after completing some levels in the game.

The research question for this thesis is: *"How to design a game for practicing logical operations?"*

## 2 Related Work

### 2.1 Computational Thinking

Computational thinking is a concept in computer science and is generally described as a way of solving problems like computer scientists[Win06]. This broad description involves a wide range of different skills, leading to multiple definitions of computational thinking. The definition I will describe here is defined by Kazimoglu et al, which is commonly used in modern literature.

Contrary to what one might assume, computational thinking is not just applicable in computer science. Kazimoglu et al. states that thinking computationally can solve problems in any discipline, using the methods, language and systems of computer science. Based on previous literature, five categories of computational thinking are defined as follows: 1) conditional logic, 2) building algorithms, 3) debugging, 4) simulation and 5) distributed computation[Kaz12].

The first category, *conditional logic*, is the focus of this project: understanding the local consequences of true/false values and how they can be used in control flow statements. *Building algorithms* use a step-by-step approach to solve a complex problem containing multiple conditional logic instances, while *debugging* is the process of finding and solving problems in an algorithm. *Simulation* involves modelling or implementing an algorithm to identify which circumstances and abstractions apply and lastly *distributed computation* refers to the social aspects of programming involving multiple parties[Kaz12][GB20].

### 2.2 Game-based Learning

The principle of game-based learning is the use of video games for teaching and learning[PHK15]. The main focus is creating fun and engaging gameplay throughout which learning takes place. By creating a learning environment that is interesting and exciting, students are more likely to voluntarily play the game and learn the concepts that are taught through it. Games designed for game-based learning should implement common game design elements, progressive difficulty being an important element as it benefits the learning process by giving increasingly challenging tasks as the player gets more familiar with the material. The game should also have a constructive feedback system to allow the player to learn from mistakes and improve on them. In addition, in-game rewards, achievements and social aspects can be additional motivations to play the game[CP13].

#### Gamification

There is one other term that is commonly used when talking about combining gaming and learning: gamification. Gamification could be considered a ‘lite’ version of game-based learning: while the latter designs a game that allows the player to learn something as they’re playing, gamification is the addition of game-like elements to non-game environments. Similar to game-based learning, this is also done with the intention of increasing motivation and engagement, often in educational contexts.

While gamification and game-based learning both have their use, I chose to use game-based learning to design and develop a brand new game, rather than simply adding game elements to existing methods. Designing enjoyable gameplay with the intention of teaching logical expressions seemed like a nice challenge. It requires a careful balance between adding enough educational elements to allow learning to take place, while also having many game elements that make the player ‘forget’ that they are actually there to practice logical expressions.

# 3 Design Process

## 3.1 Brainstorming Phase

### Gamifying logical expressions

The first step to design this game was devising a way to represent logical expressions in a more game friendly manner. For inspiration, I searched for any existing ‘games’ that do something similar. Pretty quickly, I ran into the Boolean Game. This is a very simple game consisting of a 3x3 grid where each tile has a color and a number (see *Figure 1*). Each level will show a Boolean expression, in this example ‘orange AND NOT 3’. The player then has to click all tiles that satisfy the given expression, in this case all tiles that are not 3 and are orange. With each level, the Boolean expression becomes more complicated (introducing AND, OR and NOT) and the player has less time to click all tiles. When a wrong tile is clicked, it is game over[Pro].

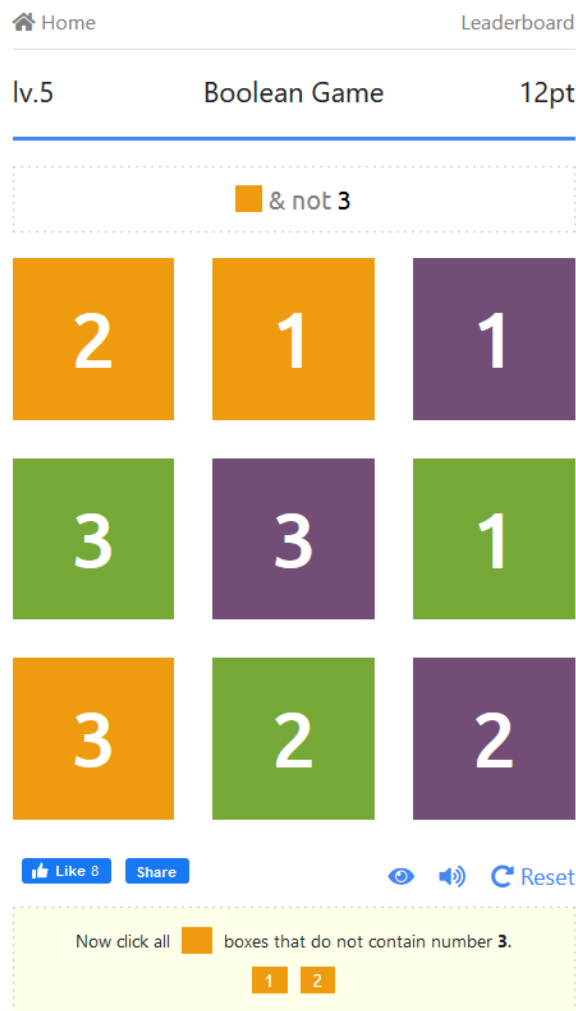


Figure 1: The Boolean Game

Source: <https://booleangame.com/>

The way logical expressions are represented in the Boolean Game is a good start, but rather than numbers in colored tiles, I decided colored symbols would work even better. This way, a logical expression could be ‘purple OR triangle’ (*Figure 2*), which would return true for any color triangle as well as any purple symbol (see the truth table in *Figure 3*). The benefit of this representation is that it consistently uses graphical symbols, instead of mixing in numbers. Having decided on this, the next step was to decide how the player would have to ‘find’ the logical expression and indicate that a certain colored symbol is true or not.

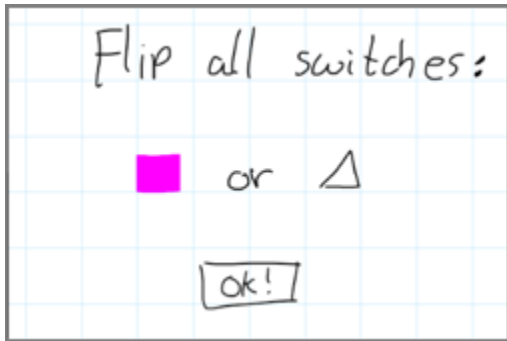


Figure 2: An early draft of a gamified logical expression





		 OR 
<b>T</b>	<b>T</b>	<b>T</b>
<b>T</b>	<b>F</b>	<b>T</b>
<b>F</b>	<b>T</b>	<b>T</b>
<b>F</b>	<b>F</b>	<b>F</b>

Figure 3: Truth table for a gamified logical expression

While we usually work with true and false, any binary action would work the same way: true or false, 1 or 0, on or off, active or not active, light or dark. Considering alternative terminologies that could replace true or false makes it easier to generate game metaphors. These function to motivate and visualize the effects of a symbol being true or false. For example, controlling water flow or light by turning respectively taps or switches on and off at different places in the level, based on their symbol. After brainstorming different options, I decided to go with light switches. Each level would have one or more logical expressions with several switches connected to one expression. The switches each have one colored symbol, which should be used to determine whether the switch needs to be turned on or off. If the symbol satisfies the expression, it should be turned on, otherwise, it should be turned off. When all switches are flipped correctly, the level is complete.

Having come up with a basic goal in the game, the next step is to determine a clear sequence that has to be followed in each level. Then, more game elements should be added to increase players’ engagement and enjoyment as well as features that will help the player learn and practice at an appropriate pace by adding progressive difficulty and tutorials. Additionally, a basic interface of what the game would look like had to be designed.

### Gameplay sequence

Firstly, the steps that are needed to complete a level had to be determined (see *Figure 4*). Each level will contain several areas, each with their own logical expression and several switches. After starting the level, the player should find the first logical expression they’re going to solve by tapping the computer icon in an area. This will display the expression somewhere on the screen so the player can easily refer back to it later. Next, a switch in the same area should be tapped to display its symbol as well as two buttons that will turn the switch on or off. After making their choice,



the same should be done for all switches in that area. Then, the whole process is repeated for the other areas in the level. The last step is to tell the player if they made the correct choices and giving them the opportunity to fix any mistakes that were made. This process will be described in a future section. *Figure 5* on the next page shows the first draft of the basic layout of the game.

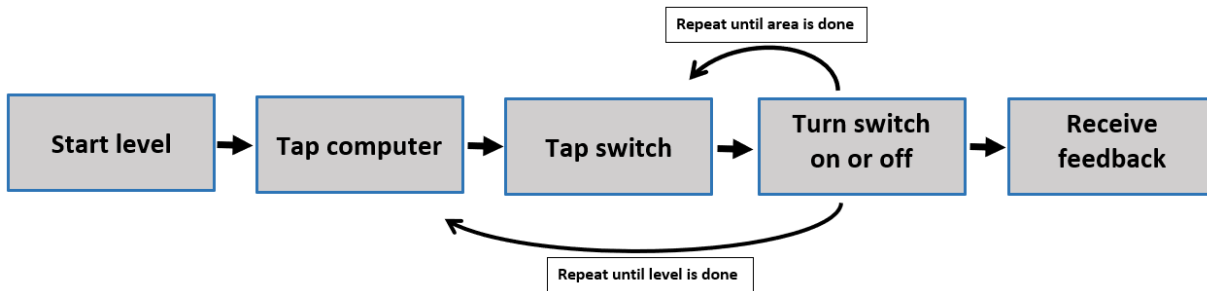


Figure 4: First version of the game sequence

### Adding gameplay elements

Many of the game elements in this section were defined with the design patterns described by Björk in ‘Patterns in Game Design’[SB05] in mind.

Possibly the most important element to add to the game is **difficulty**. Increasing the difficulty slightly with each level will not just ensure that the game remains challenging for the player, but also benefits their learning experience. At the same time the difficulty should be modulated to minimise frustration. Finding the right middle ground is called the flow[Che07]. The difficulty of a level can be influenced by different factors, possibly the most important one being the complexity of the logical expressions. The introduction of operators (AND, OR and NOT) will require the player to understand how these operators work not just separately but also combined in an expression. Starting off, the difficulty should be low enough that anyone without prior knowledge of logical expressions can play the game after finishing the tutorial. At this point, only the OR operator will be used, but as the player progresses another operator will be introduced. Moreover, once the player has had some time to get comfortable with an operator, more difficult expressions will begin to appear combining that operator with another one.

To create more engaging gameplay, the player should not just be challenged by the difficulty of the expressions. Other elements that can increase the difficulty of a level are **time limits**, requiring the player to think faster the more they progress in the game. This shouldn’t make it so challenging that a level requires many attempts to get past it, but it should add just enough pressure to prevent the player from taking their sweet time for every switch. In addition, the extra challenge of having to complete the level on time, and the satisfaction when successful, increases the enjoyment players get out of the gameplay.

Additional difficulty factors could be limiting how many times a player can check for the results, preventing the option of simply going through with trial and error. In other words, the player starts a level with a limited number of **lives** and ‘loses’ a life every time they check if their answers are correct. Higher levels with more complex expressions can also contain more areas and/or switches, further testing the player’s understanding of the expression with more symbols to solve.

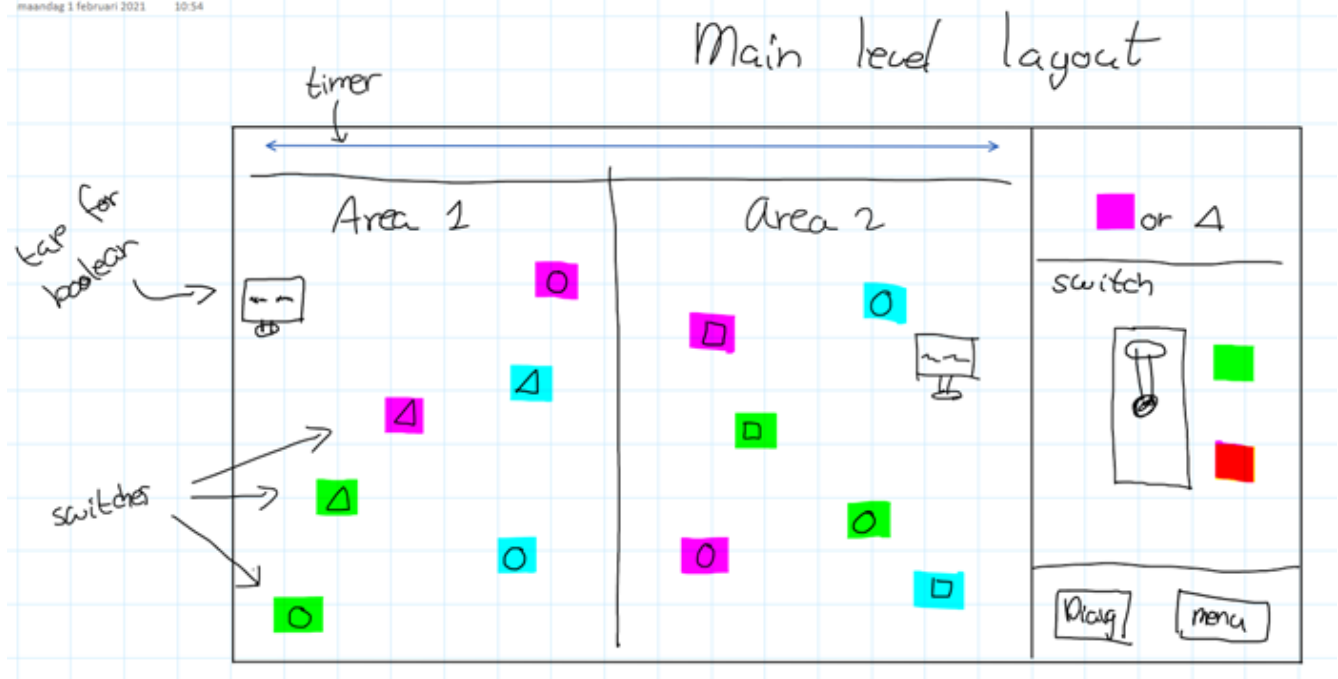


Figure 5: The first draft of the interface

To serve as both a reward and motivation to try and complete levels fast and/or without mistakes, the player will receive a **score** based on their performance upon completion of the level. The score takes into account different factors like the amount of time the player had left upon completing the level, the number of mistakes that were made and how many times the player checked their answers before completing the level. The performance of the player can be expressed as a number, which can be translated to a one, two or three star score or a bronze-silver-gold star score to clearly represent how well the player did. This can increase the player's motivation to do well on the levels. Bonus points can also be awarded when the player completes several levels in a row without failure, doesn't make a single mistake or only flips each switch once. The score element also allows for the addition of a **leaderboard**, where players can compare their own performance to that of their friends and other players.

**Tutorials** are essential in many games to give the player instructions on how they should play the game and this one is no different. Since I want everybody to be able to play this game regardless of their prior knowledge on logical expressions, two things need to be explained in the tutorial: how to play the game and how to 'read' logical expressions, subsequently using them to decide if a switch needs to be turned on or off. Since there's quite a lot to go over, the best way to teach players is by guiding them through a full level, explaining each step along the way. After completing the tutorial, players should understand which steps they need to take to go through the level, what a given logical expression means, and how to use it to determine if a switch's symbol satisfies the expression or not. The tutorial will explain the meaning of OR and, after completing a few levels, AND and NOT will be introduced, including a short tutorial players can choose to read if they are unfamiliar with them.

Besides explaining how an operator works when it's introduced, we have to consider the possibility of people not playing the game for a while and not remembering exactly what an operator does when they come back. Additionally, some people might need to go over the explanation several times. To make sure all the information players need is in the game, a **diary** could be added. This diary would keep track of all the operators the player has learned, including explanations, examples, and how they behave when used together with other operators in an expression. The player can access this diary at any moment should they need it. However, the level's timer will keep running, so they need to check it quickly.

### Designing engaging feedback

The last aspect of the game that needed to be designed is how to give feedback to the player. Once all switches have been flipped, the game has to check if they are correct or not. If not, the player needs to fix any mistakes before they can continue to the next level. However, due to the binary nature of each switch (either on or off), it would be far too easy to tell the player directly which switches are wrong. If a switch is turned on, and the player is told it's incorrect, all they have to do is go and turn that switch off. This would mean that after checking the results once, they would know exactly which switches should be flipped the other way. Obviously, this isn't very beneficial for the learning process as the player can completely avoid thinking and reasoning to solve the level.

Because of this, a different system had to be designed that informs the player about their mistakes indirectly. Rather than pointing to a specific switch, the player should know roughly in which area one or several mistakes were made. Since the switches were originally thought of as light switches, each of them could turn one or more lights on. All switches together would, when flipped correctly, create some sort of figure with the lights they turn on. On the other hand, flipping switches incorrectly would disturb the figure (see *Figure 6*).

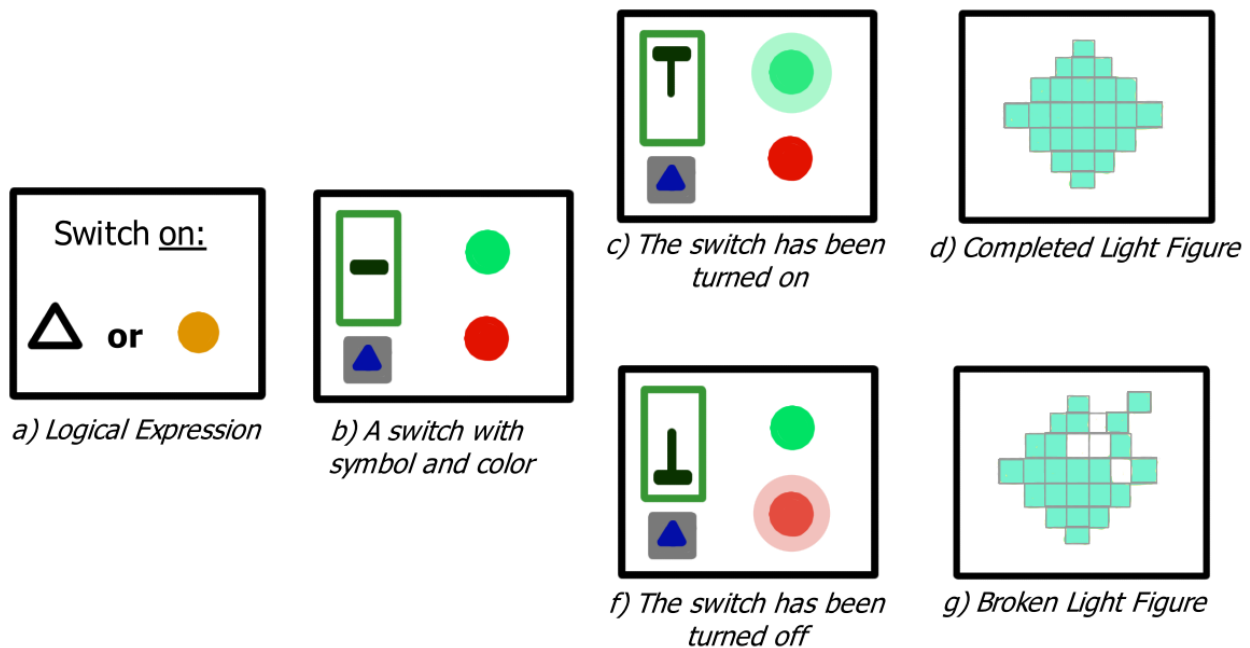


Figure 6: A simplified sequence showing how the lightboard functions

This idea was visualized as a two dimensional board that consists of a number of tiles that are controlled by the switches (from now on referred to as *lightboard*). If a switch is turned on, it will activate all tiles connected to it. The location of the switches in the level should roughly correlate to the tiles they are connected to. Therefore, if the pattern seems to be disturbed on the right side of the board, the incorrect switches will likely be located on the right side of the level layout. This will give the player enough information to know in which area they made one or more mistakes, without pinpointing the exact switch that is wrong. As a result, the player needs to think about each symbol to determine if they may have made the wrong choice the first time around.

## Final remarks

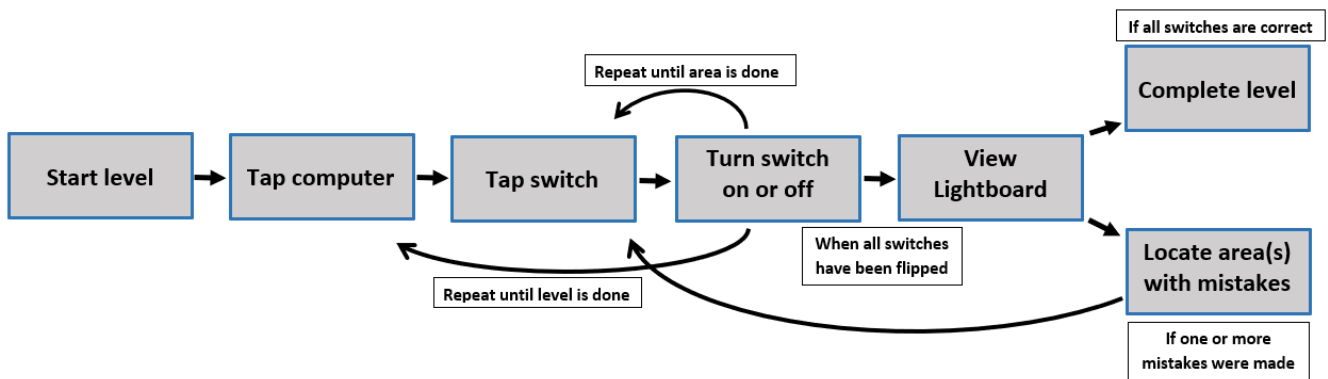


Figure 7: The game's sequence after brainstorming

After this idea generation phase, the main steps that the player needs to go through to complete a level (*Figure 7*) are in place. In addition, several difficulty factors, gameplay elements and educative features have been designed, as well as a basic draft of what the game might look like (see *Figure 8*). At this point, user testing is needed to gather feedback on the current design of the game.

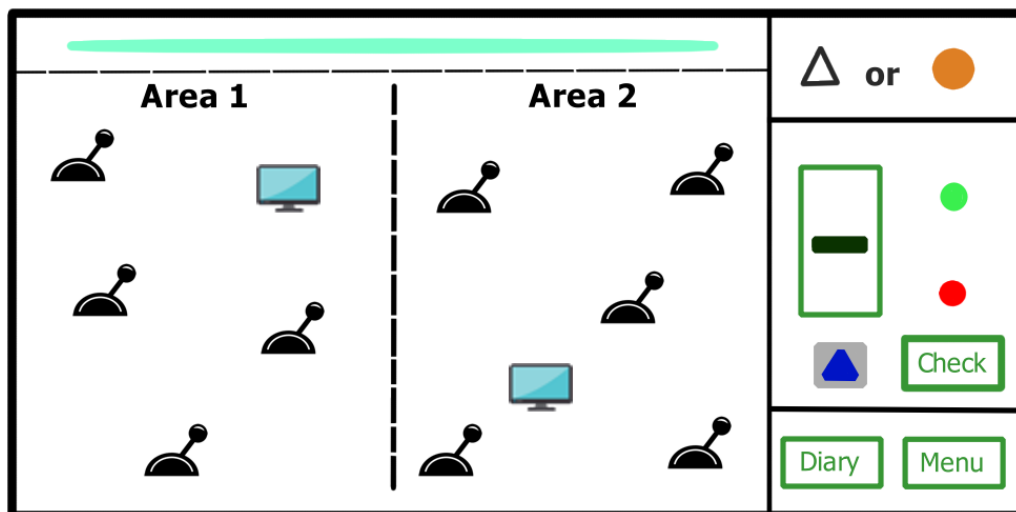


Figure 8: The final draft of the game before testing

## 3.2 First Prototype and User Testing

### The AdobeXD prototype

Before any user testing can be done, a prototype needs to be developed. There are several ways to develop a simple prototype: Proto IO[Inc], Adobe XD and paper prototypes were considered. The free version of Adobe XD seemed best fit for the game I wanted to create and was very easy to use.

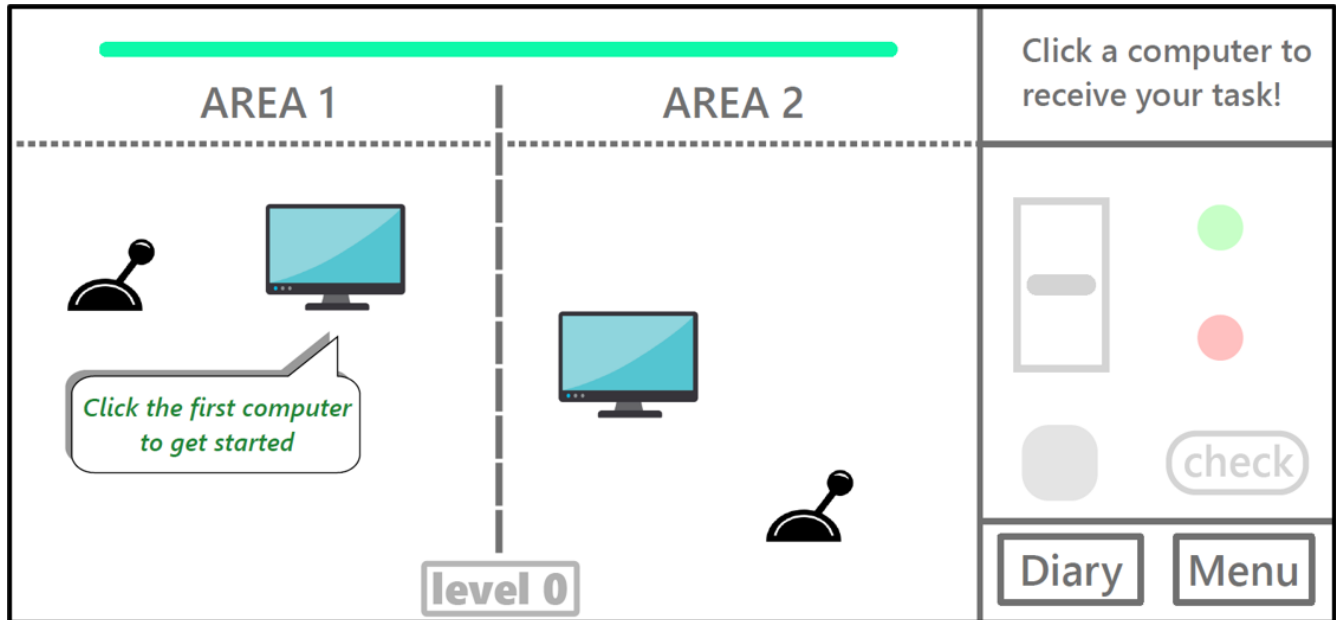


Figure 9: The first screen of the Adobe XD prototype

The aim for this prototype was to create a tutorial that explains how the game works, followed by a test level that the player has to complete on their own. For the interface, the design in *Figure 8* was used as a basis, though some small aesthetic changes were made (see *Figure 9*).

During the **tutorial**, the game takes the player through a guided level step by step. Text bubbles explain the different steps and aspects of the game and indicate where the player has to click to continue to the next step. Because the player is assumed to have no knowledge of logical expressions, the tutorial explains what each expression means when the player encounters it (see *Figure 10*).

The tutorial follows a linear path, meaning that the player can only click where they are told to click and cannot go back. Clicking on an icon when not in the correct step of the tutorial will simply do nothing.

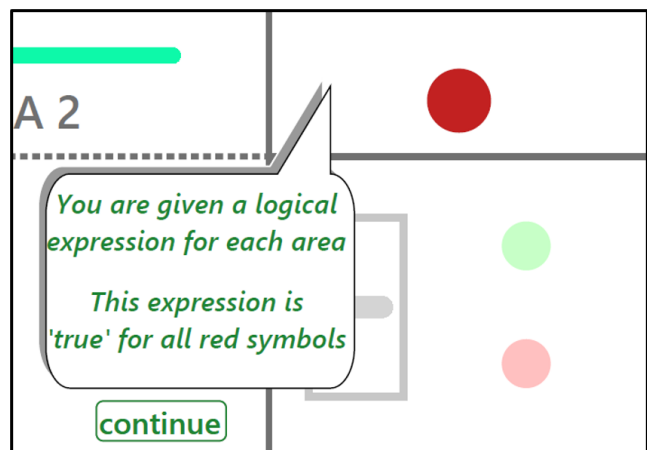


Figure 10: Example of the tutorial

After completing the tutorial, the player will attempt to complete **level 1** on their own. There are no longer any text bubbles or arrows to guide them, but the layout of the game is identical to the tutorial. Thus, the player must use what they learned in the tutorial to go through the level.

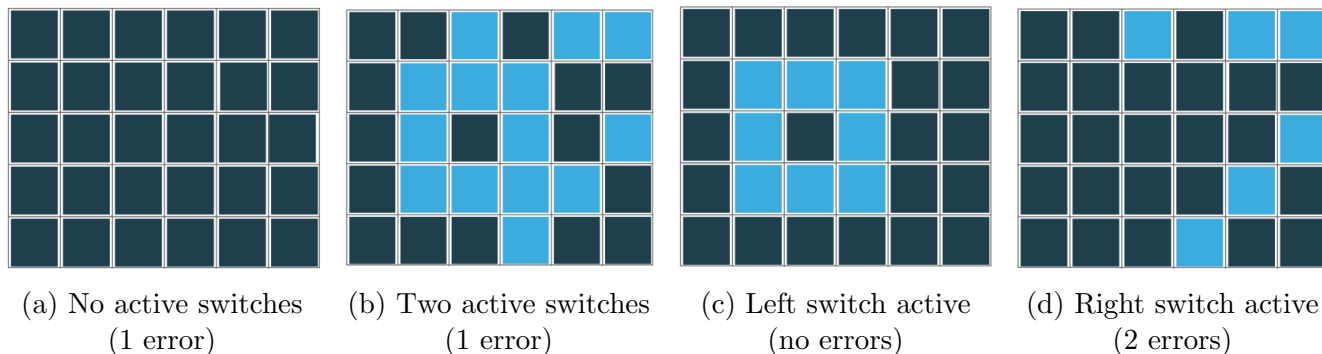


Figure 11: The prototype’s four different possible lightboards

To start off, the player must choose one of the two computers in the level to make the logical expression appear on the screen. Next, they need to tap the switch that is in the same area as the computer they picked. The switch’s symbol will appear on screen and the player needs to determine if this symbol satisfies the expression or not. After deciding, they have to turn the switch on or off by clicking on the green or red circles (*Figure 12*). The process is then repeated for the other area. When both switches have been flipped, the player can press ‘check’ to view the lightboard.

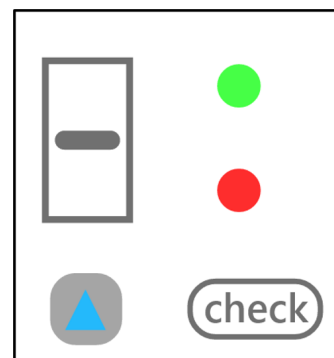


Figure 12: The switch interface showing the switch’s symbol and on/off buttons

Because there are two switches in this level, there are four possible combinations in total. *Figure 11* shows the four lightboards that the user can get. If neither of the switches are turned on, none of the tiles are lit up and the lightboard is dark. Out of the three remaining options, only one shows a clear symbol (*Figure 11c*), while the other two appear to have randomly lit tiles. This means that for this level, the correct answer is to turn the switch in the first area on, and the second switch off.

If the player does not flip the switches correctly, they will see the incorrect lightboard, as well as a message saying they did not complete the level yet. The message will state if the player made one mistake or more. By clicking ‘back’, the user can go back to the level to change the switches they flipped incorrectly. After pressing check again, they will be presented with the correct lightboard and finish the prototype.

### User testing

The goal for the testing in this phase is to determine 1) if the interface is clear and easy to understand and 2) if the tutorial gives adequate information for the user to understand which steps they need to take to get through the game. The data from these tests has been evaluated and any issues with the interface or tutorial were noted.

The user testing for this game was done using the think-aloud protocol. Before the test started, the users were given a basic explanation on what was expected from them. They were told to play through the prototype while saying everything they were thinking out loud. Both the browser in which the prototype was played and the user’s voice were recorded, so that the whole process can be watched back later for further evaluation.

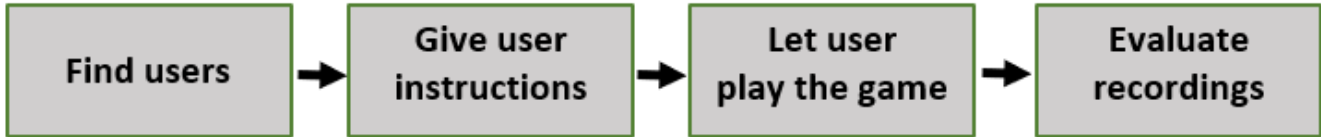


Figure 13: The steps taken during user testing

This process is visualized in *Figure 13*. First, I had to find candidates that were willing to test the prototype. Due to the Covid-19 pandemic, it was hard to reach users in the target group (students that are just starting out or going to start learning to program soon). To make sure the user testing could still be done I settled for users that, while not in the target group, were still able to give relevant feedback on the interface of the game. The participants were also informed that their voice and the computer screen would be recorded and evaluated later.

After receiving the instructions and having no more questions, the user played through the prototype. Occasionally, they would have to be reminded to keep talking out loud, but otherwise no further instructions were given during testing. After finishing the prototype, some users were asked to elaborate on elements that confused them while playing the game. This feedback was also used in the evaluation of the data to establish common issues that users encountered.

## Results

When all user tests were finished, the recordings were evaluated and a list of common issues that were encountered was created. For every problem, a possible solution was determined. These changes and additions are included in the development of the second prototype. The reported issues and the corresponding solutions are found in *Table 2*.

Problem	Solution
Link between symbol and switch unclear	Give a reminder during the tutorial
Link between symbol and expression unclear	Explain why the symbol satisfies the expression
The (end)goal of the game is unclear	Explain the goal of the game before the tutorial
Colored circle to indicate color is confusing	Use a paintspat icon to indicate color
Lightboard too confusing	Explain the lightboard slowly step by step

Table 2: User testing results

Most of these problems appear to come down to the tutorial not giving the user enough information, the information not being clear enough, or the user being able to easily move on to the next step without understanding the text (see *Limitations*). In the future, these issues should be resolved by tweaking the tutorial text to make it more clear and giving the user more freedom during the tutorial. Moreover, increasing the engagement during the tutorial, by avoiding a sequence of ‘click here’, will likely improve the user’s understanding and information retention.

## Limitations

While Adobe XD was the best choice for this prototype, it does bring some limitations with it. To explain why, I need to say something about how Adobe XD works. Starting off with an empty page that simulates a phone screen (called an *artboard* in Adobe XD), elements (pictures, lines, shapes, text) can be added to it with a simple drag and drop method. The prototype consists of many artboards, all connected to each other. Clicking a specific icon will send the user to the artboard that has been connected to that icon.



Figure 14: Level 1 of the prototype as seen in Adobe XD

As a result of this, all possible paths that the user can take through the level have already been determined, and the user cannot go a different path if it hasn’t been ‘programmed’ into the prototype. After choosing which computer to start with, the player can only make a choice by turning a switch on or off, and is unable to go back. Figure 14 shows what level 1 of the prototype looks like in Adobe XD, showing all the artboards I have made. The blue and green line show two possible paths that the player can take when they play through the level.

As a result of this method, the choices the player can make in the game are limited. They are not able to go back and change a switch after moving to the next step, nor can they click on something that isn’t the correct option, as it would have no effect at the step they’re currently at. Once the first computer is clicked, the full sequence of the game is basically set in stone (apart from choosing whether to turn a switch on or off).

This potentially impacts their learning and understanding of the game, as they can keep clicking icons until they try the correct one, which will take them to the next screen. There is also no way for the user to differentiate between icons that are currently clickable and icons that are not, which might lead to confusion.



For the same reasons, the tutorial is completely linear. At each step, the user is only able to click where the tutorial tells them to click. The result is that the tutorial comes down to ‘click wherever the arrow is pointing’ and the user might get through it while still not (fully) understanding what they are meant to do. After all, if they are confused about the explanation, they are still able to click where the arrow points and move forward while lacking possibly important information about the game. This can impact performance in the following level, when the user needs to follow the correct steps without instructions.

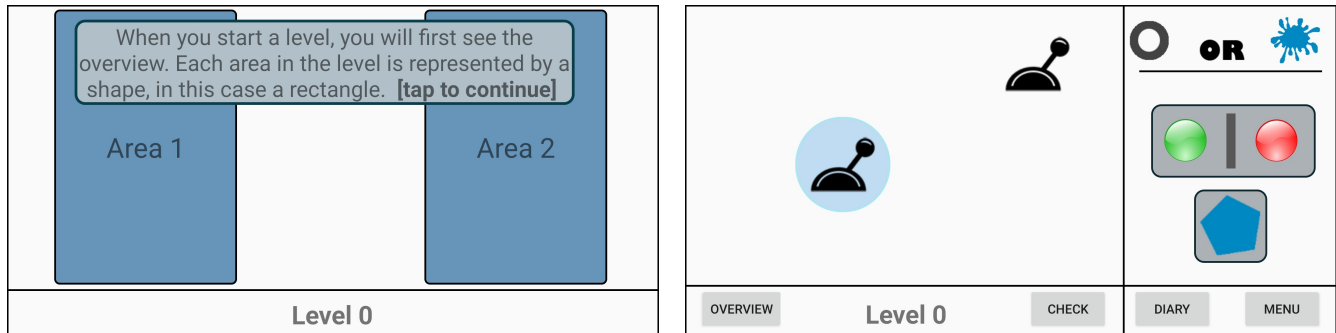
The last major limitation of this prototype is the very low number of switches. Each area only has one switch for a total of two switches in each level, while the game was designed to have a higher number of switches in each area. However, increasing the number of switches would also exponentially increase the number of artboards that are required for the prototype to function, and would have cost too much time to develop. While necessary, this choice does impact the quality of the prototype. With only two switches there is a 25% chance that someone guesses both switches correctly, successfully completing the level yet not understanding what they were supposed to do. Additionally, the lightboard was not designed for only two switches, meaning the lightboard in this prototype is highly simplified and might not represent the intended functioning very well.

### 3.3 Second Prototype and Concept Testing

Many of the limitations in the last section can be dealt with when programming a prototype rather than representing an application in Adobe XD. In order to do further testing, a more advanced prototype had to be developed. Due to previous experience and this game being intended as a mobile game, I chose to develop the second prototype in Android Studio. Android Studio is an integrated development environment specifically designed for Android development [Goo]. The final prototype consists of a short introduction, tutorial and two levels the user completes by themselves. After the first level, the AND operator is introduced, which is then practised in the second level. All the improvements listed in *Table 2* will be implemented in this prototype, so that they can be re-evaluated to see if they have solved the problems encountered during the user testing.

#### Changes and improvements

Instead of directly starting the tutorial, the user will instead be presented with a short **introduction** when they start the app. This introduction explains the goal of the game, what logical expressions and operators are, and a short description of what they need to do in a level. The intention is to give the player a better idea of what they are doing in the game and why. Lastly, the player is told that the tutorial will take them through the level step by step to make sure they don't feel too overwhelmed.



(a) The new level overview

(b) The new level interface

Figure 15: The new interface in the Android prototype

While the **interface** remains largely the same as the first prototype, one important element was changed. Rather than having two areas next to each other at once, a level starts off with an overview displaying the areas in the level as simple rectangles (see *Figure 15a*). This change ensures that more areas can be added to a level without running out of space on a small phone screen. Tapping on one of these areas will bring the user to the area with the switches (see *Figure 15b*). This interface is similar to the original design, with some small tweaks to make it more user friendly. The switch interface on the right is now less cluttered and only contains the necessary elements: two buttons to turn the switch on and off (green and red circle respectively) and the switch's symbol underneath. The 'check' button to view the lightboard has been moved away from the switch controls, and a button has been added to go back to the overview. Lastly, the computer icon has been removed from the area. Since the user will now view only one area at once, the extra step of tapping the computer to receive the expression is unnecessary and not user friendly. Instead, the expression is always visible while the user is navigating in the area.

Other changes to the interface are minor: the lightboard has remained very similar to the lightboard in the first prototype, the only change being that it now has more tiles (15x15). This change was possible because there are more switches in the levels (4-5 compared to 2); having more options for designing light figures, the size of the board was increased.

The menu button now works and shows the three levels (tutorial, level 1 and level 2). The player can tap on a level to start playing it, assuming they have completed the level before it. While there is a diary button, the diary itself was not implemented in the final version of this prototype and the button does nothing. Finally, in the expressions colors are now represented by a colored paintplat icon (see the expression in *Figure 16b*) to make it clear that it only indicates a color, not a type of symbol.

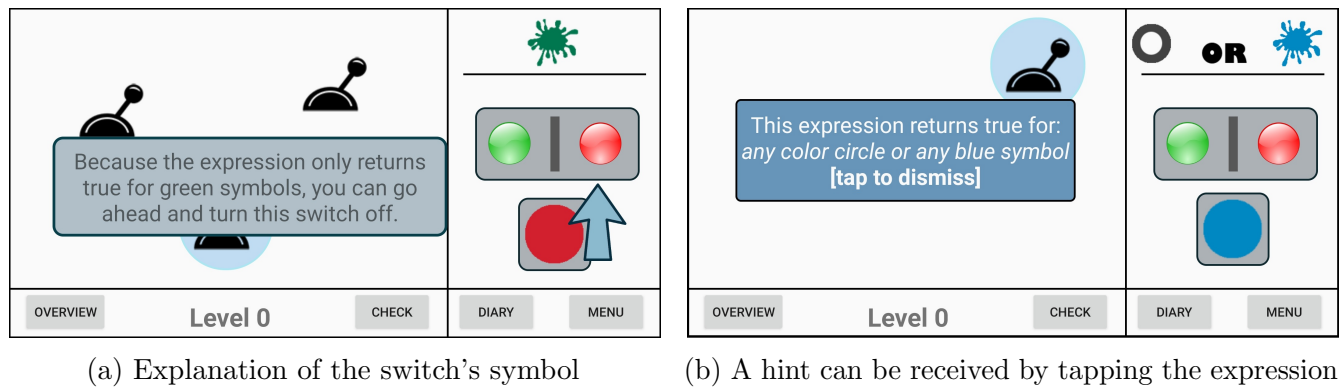


Figure 16: Some changes in the Android tutorial

The **tutorial** has undergone some major changes in the Android prototype. While the tutorial in the Adobe XD prototype was completely linear and had the player do exactly what the tutorial said, the new tutorial is more engaging and relies much more on the player thinking about what they just learned, allowing them to make their own choices. Starting off, the player is asked to pick any of the three switches in the area. The game then shows them where the switch's symbol appears and explains if the switch should be turned on or off, and why (see *Figure 16a*). Once the player has tapped the correct button, they are told to repeat the process for the other two switches by themselves. Once finished, the game will ask them to go back to the overview, regardless of how they did.

The second area has a more complicated expression introducing the OR operator. Each element of the expression is explained, after which the user is asked to handle both switches in the area. After tapping a switch, the player receives a hint: the player can tap the expression to receive a description of it in words (see *Figure 16b*). The player can choose to ignore this hint if they want to, and once both switches have been flipped, the tutorial will instruct them to go view the lightboard.

The explanation of the lightboard is now more structured and easier to understand. Rather than having one screen with several text bubbles all over it like in the previous prototype, only one text bubble is now visible at a time. By tapping this text bubble, the player can go through the explanation at their own pace. If the player made at least one mistake, the tutorial will tell them to go back and find it. After the tutorial is done, the lightboard will still differentiate between one mistake or multiple mistakes and inform the user accordingly.

There is one more change to the tutorial: in addition to the OR operator, the Android prototype also introduces the AND operator. After completing level 1, the player will see a screen that briefly explains how the AND operator works, including an example (see *Figure 17*). This information, along with the practice in the tutorial and level 1, should be enough for the player to figure out the two expressions in level 2, both using the AND operator.

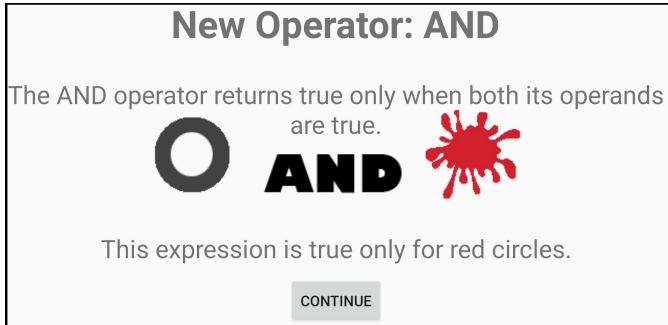


Figure 17: The AND operator is introduced

Taking the changes to this prototype into account, the game sequence is now slightly different from the original design. *Figure 18* shows the game sequence in the final prototype.

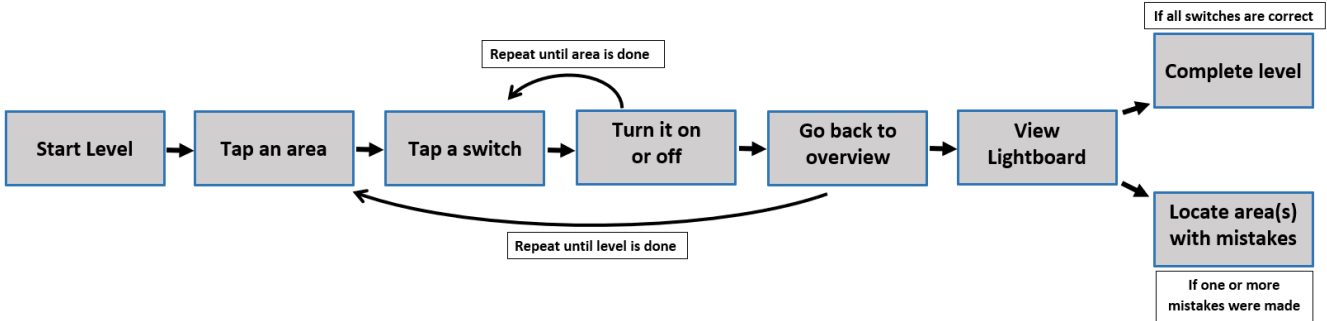


Figure 18: The game sequence in the Android prototype

**The Android prototype**

As mentioned before, the final prototype was developed in Android Studio using Java and XML. The use of Android Studio and Java classes allowed for a much more flexible prototype compared to the first prototype in Adobe XD. The main components of an Android app are called *activities*. One activity implements one screen in the application. For example, the overview of level 1 is one activity, and the first area of level 1 is another activity. An activity consists of a user interface, implemented by an XML file, and a Java file which contains all the functions required for the activity to work. Additionally, Java classes are implemented in separate Java files.

Figure 19 describes the Java classes in the final prototype in a UML Class Diagram. The prototype implements a **Player**, **Level**, **Switch** and **Tutorial** class. The **Tutorial** class is fairly straightforward and serves to keep track of where the player is in the tutorial. The **Switch** class contains information about a specific switch, for example it's *status* (on or off, depicted with a boolean), the correct solution for this switch (*answer*) and to which tiles on the lightboard it is connected (*tiles*). For each level, one **Switch** object is created for each switch and all are added to an `ArrayList` (*switches*), which is passed to a **Level** object. The **Level** class holds all necessary information about a level. This includes an integer that keeps track of which switch is currently selected (*currentSwitch*) and whether the level has been cleared or not (*cleared*). Similarly, all **Level** objects are added to an `ArrayList` and passed to the **Player** object. This way, the **Player** class has access to all information of the other classes. After the introduction, a **Player** object is created as well as the **Level** and **Switch** objects. The **Player** object is passed to different activities for easy access to the level data.

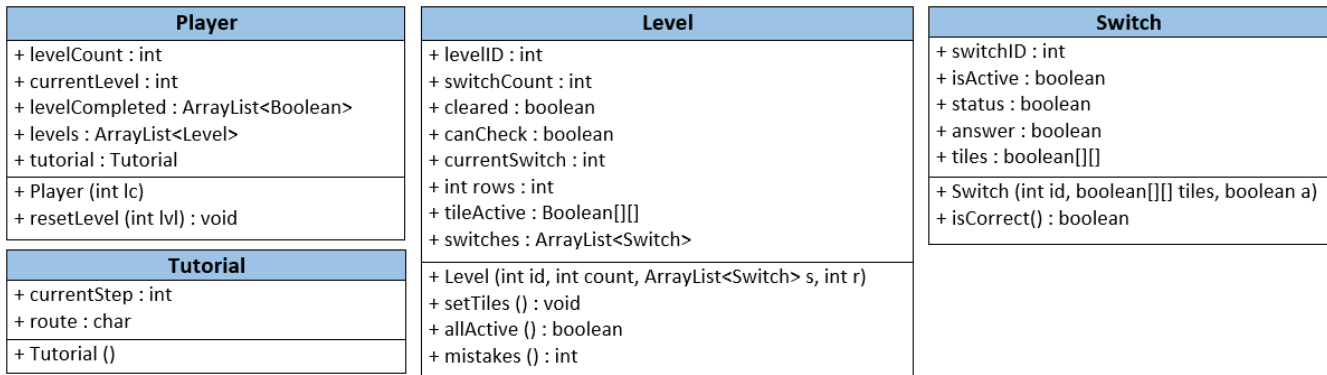


Figure 19: The Java classes for the Android prototype

## Concept testing

The final step in this project was to test the Android prototype. Since the gameplay in this version is more complete than in the first prototype, this final prototype can be used for concept testing. Once again the player will go through a tutorial level, which has been improved since the last version. Then, they will play level 1 to get familiar with the OR operator. Finally, the AND operator will be introduced and practiced in level 2. There are a few things I wanted to test with this prototype:

1. The learning that has taken place
2. The quality of the tutorial
3. The quality of the game elements

This data has been gathered using a survey. Users are asked to fill in the first questions of the survey before they play the prototype. These questions ask some basic information about the user (age, gender, educational background) and some logic related questions. The aim of these questions is to find out how familiar the user is with logical expressions in general as well as the OR and AND operators. After these two sections, the user is asked to play through the prototype and come back to the survey when they finish it. The process is shown in Figure 20.

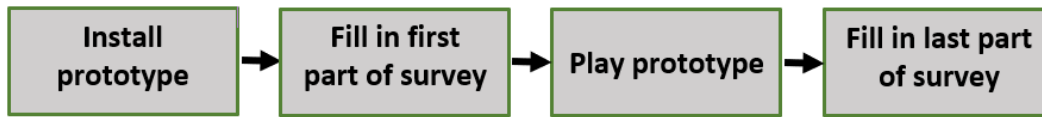


Figure 20: The steps the user takes during concept testing

After playing the prototype, the user has to answer a few more logic related questions. To test how well players understood the game’s logical expressions, some of the questions use the same gamified logical expression format as in the prototype. Finally, the last section asks questions specifically about the prototype. Because many of the originally designed game elements didn’t make it into the prototype due to time constraints, the user is instead presented with a list of some potential game features and are asked whether they would like to see a feature implemented or not.

## Results

To get an idea of the **learning process** that has taken place for each user, users are asked how familiar they are with logical expressions at the start of the survey. One user responded that they are familiar with them and subsequently made no mistakes before playing the game, but did make one mistake after playing. Two users stated that they do not know what logical expressions are. Both users made at least one mistake in the logic questions before playing the game, but did not make any mistakes after playing. However, one user selected ‘I don’t know’ to one of the non-gamified questions. In addition, both indicated that they felt they have a better understanding of logical expressions after playing the game.

Several questions were asked to check the **quality of the tutorial**. When asked how well the user understood the game after completion of the tutorial, 2 out of 3 users said they understood it completely (scoring 5/5). Interestingly, one user said they didn’t fully understand the tutorial (scoring 3/5), but reported that they had no issues completing level 1 and 2.

The remaining questions aimed to gauge the **quality of the game** over a traditional learning process of doing exercises. When asked what they prefer, two users answered that they would lean towards playing the game over standard exercises (scoring 4/5 and 5/5) and one user remained neutral (scoring 3/5). However, when comparing the lightboard system to simple correct/incorrect feedback, two users preferred the simpler form of feedback (scoring 2/5), while one user prefers the lightboard (scoring 4/5). None of the users used the lightboard to locate their mistakes.

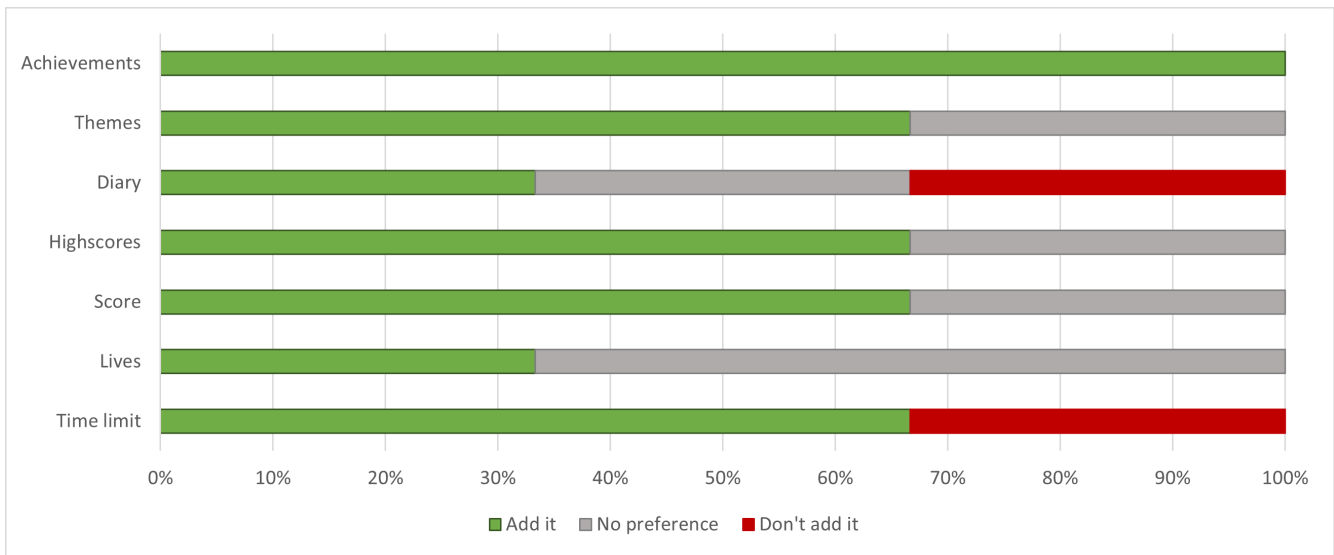


Figure 21: Users' response to additional game features

Lastly, users were given a list of possible **game elements** that were not present in the prototype and are asked if they would like to see them added or not. The results of this question can be seen in *Figure 21*. As shown in the graph, most features were positively received, with only two cases where a user selected to not add an feature.

## 4 Discussion and Future Work

### 4.1 Evaluation of the Design Process

During the brainstorming phase, the core of the game was designed and several additional game elements were brainstormed and worked out. However, in hindsight doing some more research on existing similar games might have been beneficial. I found the Boolean Game fairly early on and used its representation of Boolean expressions as inspiration for my own idea. From there, all the other ideas fell into place and I didn't do much more research on alternatives. While I don't think this was a mistake considering I'm satisfied with how the design of the game turned out, it would not have been a bad thing either to spend a bit more time on the first step of the process.

I'm satisfied with how the Adobe XD prototype for user testing turned out. While it is simple and has some limitations, it was able to do the job of allowing users to evaluate the tutorial and interface of the game. Additionally, this prototype was designed and ready for user testing in only three weeks time. Keeping this prototype simple left enough time to develop a second, more advanced prototype in Android Studio. The only downside during user testing is that only three users were able to test the prototype, and none of them were part of the target group (students starting out with programming). This is partially to blame on the Covid-19 pandemic and resulting lockdown making it more difficult to find people willing to test using the think-aloud protocol. Nonetheless, I believe the issues that were identified during user testing likely covered the most important problems with the prototype and the user testing was successful.

The last step of this project was developing an Android application that would be the final prototype of my game. Using Android Studio was a good choice, as I was already familiar with the program and find it easy to work with. Overall, I didn't get as far with the prototype as I had initially hoped, but for four weeks work I think I did as much as is reasonably possible. All the important elements made it in and the prototype is a significant upgrade from the first one.

### 4.2 Discussion of Results

When discussing the results of the concept testing, it is important to note that, similarly to the user testing, there weren't many participants. While I can analyze these results and interpret them, further testing is required to confirm my assumptions.

#### **Learning process**

Starting with the learning process, both users that started with no knowledge of logical operations reported that they had a better understanding after playing the game, indicating that learning has taken place. This is further confirmed by their answers to the logic related questions that show improvements after playing. Interestingly, one of them did respond with 'I don't know' when asked what the outcome was of 'true AND false'. But when asked the same question using the gamified representation with symbols and colors, they were able to answer correctly. This might indicate that they were not able to fully transfer what they learned in the game to a representation using true and false.



In addition, one user commented that they found the colors and shapes easier to understand than true or false, likely referring to the two types of logic questions that were asked. This suggests that the colors and symbols are a good choice to teach the concepts in this game.

One user reported that they were familiar with logical operations before playing the game, but did make one mistake after playing when asked to select which symbols make an expression return false. However, this might be due to misreading the question, as they had instead selected the symbol that makes the expression return true. Since the user indicated they were already familiar with the concepts the game taught, not much can be said about their learning process.

## **Tutorial**

The tutorial appears to be successful, as two users reported that they understood the game completely after finishing the tutorial. One user did not fully understand the tutorial and noted that they missed the connection of the green and red lights turning the switch on or off until after the tutorial. This does not seem to be a major concern as they reported no problems with completing level 1 and 2. One user reported that they had some minor issues completing level 1 and/or 2, but were able to resolve them easily. In conclusion, the tutorial seems to be a significant improvement compared to the version in the first prototype, where all three users ran into various problems when having to complete a level on their own.

## **Game elements**

One notable feature that requires improvement is the lightboard. While one user made no mistakes, and therefore had no use for it, the other two users reported that they did not use the lightboard despite making one or more mistakes. Perhaps they already had an idea of which switch might be wrong or they thought it would be easier to go back and check each switch. In addition, two users indicated they would rather receive more straightforward feedback that tells them if a switch is correct or not. It should be noted that the levels in this prototype were still relatively small, having no more than five switches. With so few options, it might indeed be faster to check them individually compared to trying to understand the hidden light figure in the lightboard. With bigger levels and more switches, a lightboard will likely have more value. The addition of a time limit and/or lives system would further benefit the use of the lightboard.

While I am not ready to give up on the lightboard completely, as I think it has value over simply telling the player the answer, some changes are definitely necessary. More testing is required to find the correct balance to where the lightboard is usable and players are willing to use it, without giving too much information away.

As many game elements were missing from the prototype, users were asked to indicate if they would like to see a feature added or not. In general, the response was positive: all suggested features had at least one vote to add it to the game. Achievements is the most popular feature, as all users voted to add it. Themes and score systems were also positively received. This could be because both add elements that make the game more interesting without directly influencing the gameplay. On the other hand, the diary and lives system were the least popular. The time limit is an interesting case: two users voted to add it, while one user is against adding it. Because users were not asked to explain their choice, it is hard to say for sure what causes these differences.

## Conclusions

In conclusion, the design of the game appears to be promising. Users with no knowledge of logical operations were able to learn from the game and correctly apply their new knowledge to relevant logic questions. Most users were also positive towards playing this game over learning from traditional exercises and were interested in the implementation of additional game features. However, further testing with more participants and additional prototypes will be necessary to confirm the findings in this thesis and continue the development of the game.

### 4.3 Future Work

At the end of this project, the core elements of the game are in place. However, much can still be added to improve the player's experience, starting with the game elements described in section 3.1. Increasing difficulty can be explored when there are more levels, building up to create more complex logical expressions consisting of more than one operator, allowing the player to learn how to deal with multiple operators in one expression. The addition of a score system is something I believe will greatly increase motivation. With scores, there can also be a social aspect to the game by allowing players to compare their score to that of other players.

Another feature that has not been discussed much before, but would also be very beneficial for the game, is to implement different themes. Currently, the background of the interface is uninteresting and doesn't add anything to the game or the player's experience. While it might be hard to add a story to the game, it would be possible to have a more thematic interface and background. This theme could change every few levels and be accompanied by a level selection screen that has a similarly themed background. For example, the game could move through a forest, a lake, a jungle, the sky and many more. Every theme switch can introduce a new element to the game, like an operator, a new combination of operators, or even a new game mechanic.

The fundamental aspect of the game in its current state is flipping switches on or off based on their symbol and the logical expression. For variation, it would be good to vary this mechanic. A simple example would be asking the player to select the symbol that satisfies the expression, or one that does not satisfy the expression. This simple variation would keep players on their toes instead of having them do the same thing over and over. It would be even better to introduce more similar mechanics as the player progresses, potentially also adding new difficulty factors.

To bridge the gap between flipping switches and writing expressions in a programming context, a more advanced variation of the game could be added. Instead of receiving an expression and flipping switches, the player would see which switches (with their corresponding symbols) should be turned on and which should be turned off. Alternatively, the switches could be left out completely. The player's task is to then 'write' a logical expression, using symbols and colors, that would lead to the desired result. One way to do this is through a drag-and-drop system where the player is given a selection of symbols, colors and operators to work with. Once they have found the correct expression, the area will be completed.

This kind of variation would be a significant increase in difficulty and could be introduced towards the end of the game. This way, it serves as a final preparation before the player starts programming.

## References

- [Che07] J. Chen. Flow in games (and everything else). *Communications of the ACM*, 50:31–34, 04 2007.
- [CP13] G. Featherstone C. Perrotta. ame-based learning: Latest evidence and future directions. *NFER Research Programme: Innovation in Education*, 04 2013.
- [GB20] F.F.J. Hermans G. Barbero, M.A. Gómez-Maureira. Computational thinking through design patterns in video games. *International Conference on the Foundations of Digital Games (FDG '20)*, page 4, 09 2020.
- [Goo] JetBrains Google. Android studio. <https://developer.android.com/studio>.
- [Inc] PROTOIO Inc. Proto.io. <https://proto.io/>.
- [Kaz12] C. Kazimoglu. A serious game for developing computational thinking and learning introductory computer programming. *Procedia - Social and Behavioral Sciences*, pages 1991–1999, 02 2012.
- [PHK15] J.L. Plass, B.D. Homer, and C.K. Kinzer. Foundations of game-based learning. *Educational Psychologist*, 50(4):258–283, 2015.
- [Pro] Break Project. Boolean game. <https://booleangame.com/>.
- [SB05] J. Holopainen S. Björk. *Patterns in Game Design*. Charles River Media, Inc., 2005.
- [Win06] Jeannette Wing. Computational thinking. *Communications of the ACM*, 49:33–35, 03 2006.