ENGF0002

Scenario 1

Design Document

X $\vee$ A

Group 19

Team Members & Student Numbers:

Mikoto Ando	22154769
Kenson Chung	21003603
Ayush Khatiwada	20007938
Jan Pytel	22020011

University College London

2023

1. Introduction

One of the more tricky maths topics that computer science students face at university is logic. Propositional logic and, in particular, the implies operator is a difficult concept to grasp upon first exposure. It is unfamiliar to many students as it is not taught in any A Level or IB course. Therefore, a learning tool is needed for many students to help bridge this gap in knowledge.

X $\vee$ A is a learning tool aimed at high school students who are interested in studying maths/computer science at university. The tool helps them learn propositional logic formulas and truth tables. It is gamified to produce a fun and enjoyable learning experience. This will give them a strong foundation in logic and results in them being better prepared for their logic courses at university.

2. Game Description and Graphics

Game Description

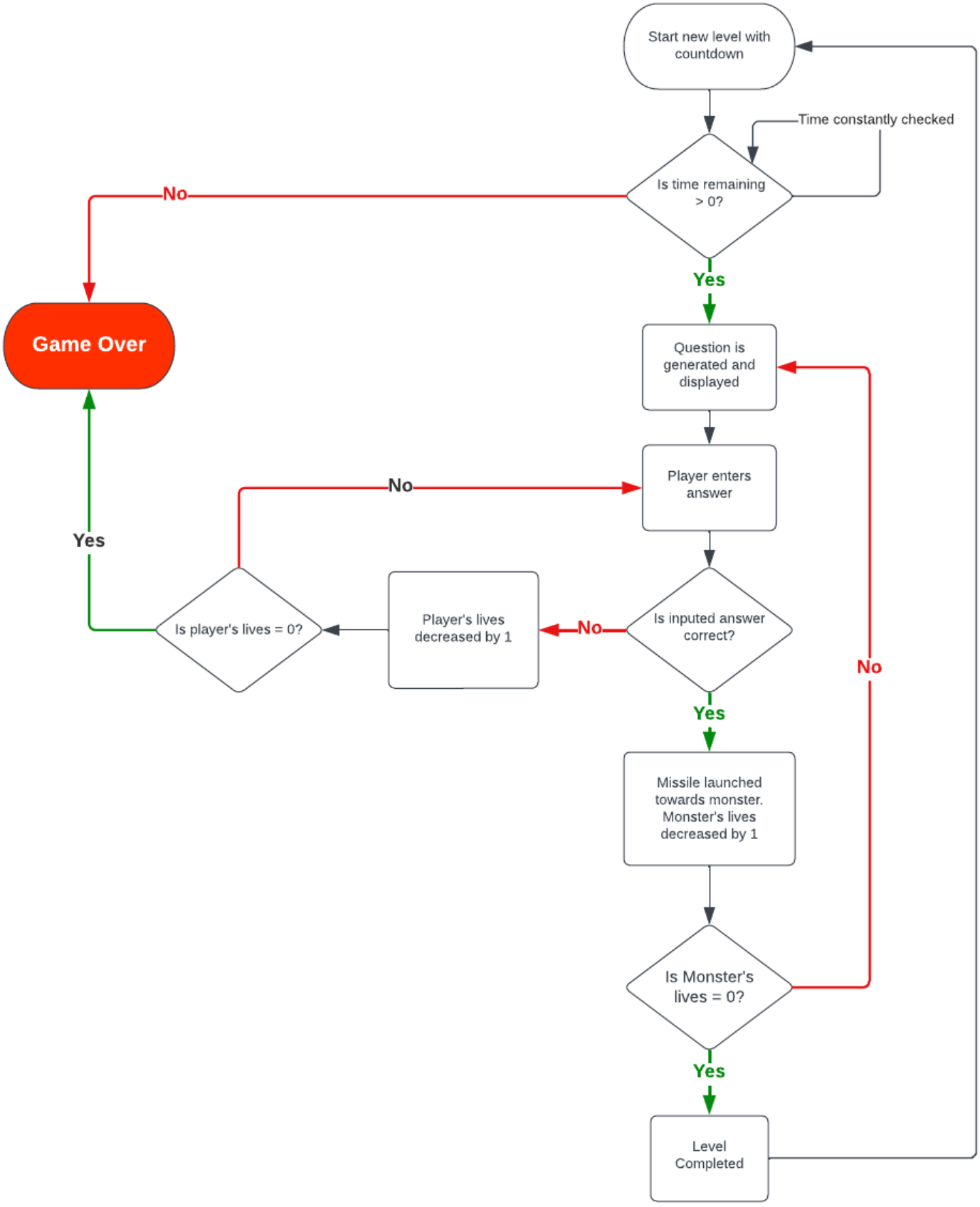
The learning tool is a game made up of unlimited levels. In each level there is a meteorite hurtling towards Earth. The player, represented by humans, will be armed with missiles, and will have a limited number of lives. The player must destroy the meteorite using the missiles before it is too late. A timer is displayed to show the amount of time remaining before the meteorite impacts with the Earth.

Each level consists of several questions. In each question a truth table representing propositional formula(s) will be displayed. Valuations for each variable in the formula will also be given. However, the table will include gaps where the player must enter 1 or 0 which represents if the formula is true or false given the valuation. If the correct numbers are entered into all the gaps, a missile is fired towards the meteorite, weakening it and decreasing its size. The next question is then displayed.

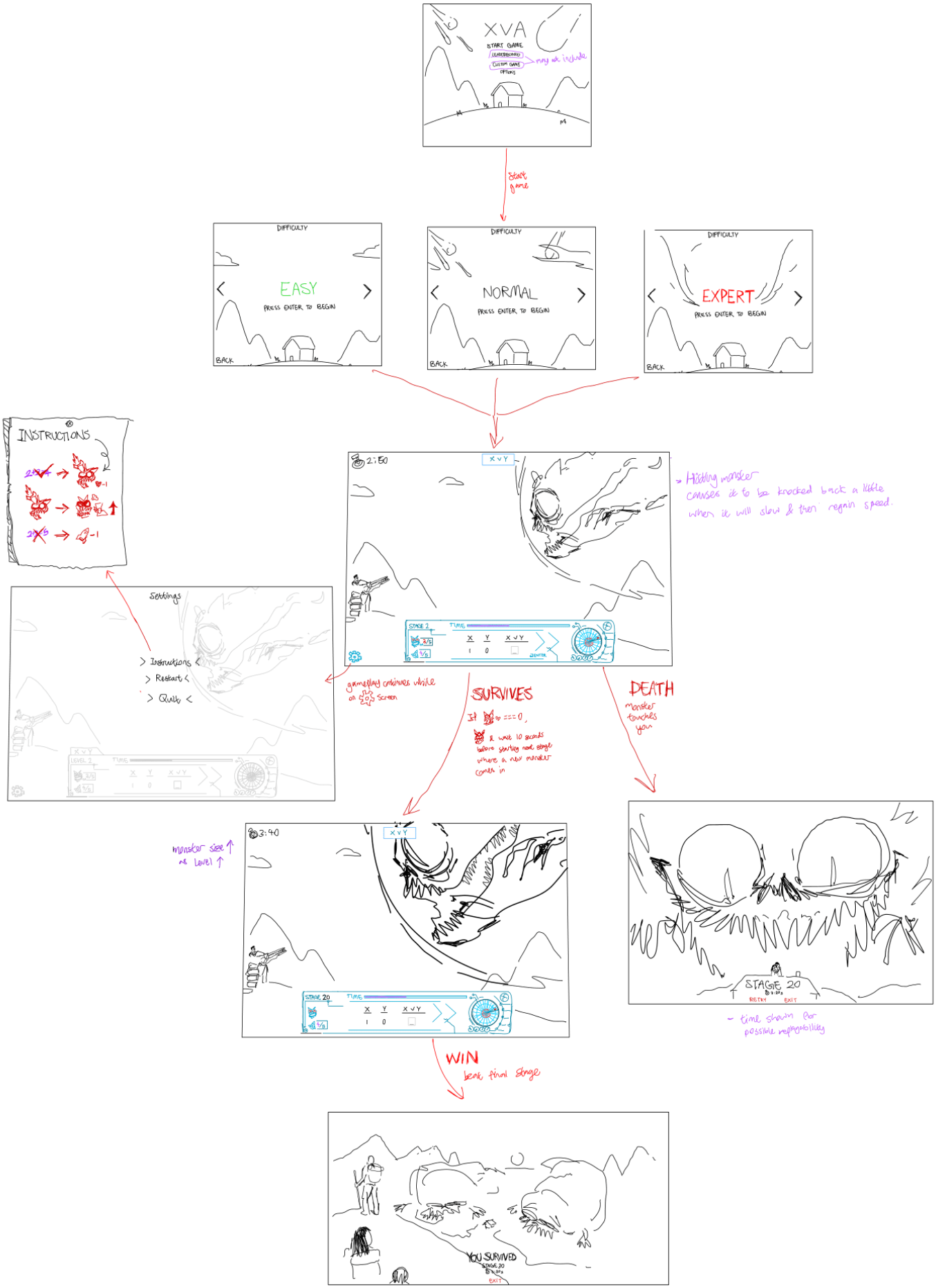
However, if an incorrect combination of 1s and 0s are entered, the player loses 1 life, and the next question is displayed. If the player has run out of lives or the time remaining becomes 0, the game ends. Once the player has answered a certain number of questions, and hence fired a certain number of missiles. The meteorite is destroyed, and the level is complete. The next level will then start.

As the levels progress, the questions will become increasingly harder and more complex. The player will also have a score showing how well they performed. The score increases when the player gets a question correct. More points will be awarded for completing harder questions, and for completing questions in a shorter amount of time.

Flow Chart Showing Game Logic



Rough Sketch of Gameplay

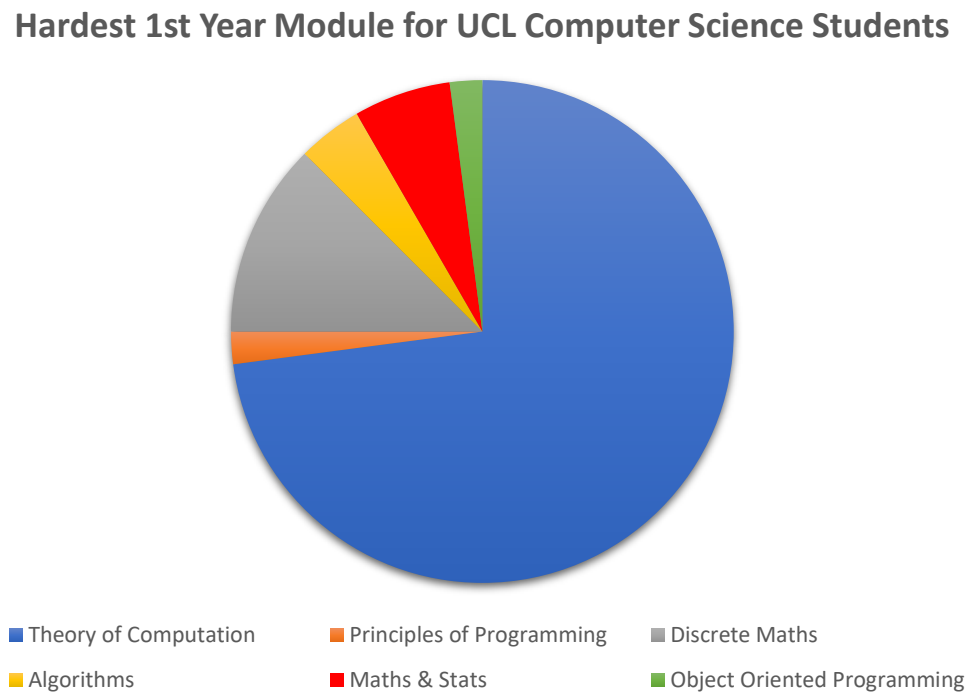


3. Justifying Target Audience and Topic Choice

For the target audience of our project, we were encouraged to select a group that we have some personal stake in. As a group, we had all commented that some parts of our degree were particularly difficult, and we wished that we were better prepared for our course. We decided that we wanted to make a learning tool that would help future computer science students have an easier time settling into their course.

After interviewing and surveying our fellow CS students, a majority of them had agreed that the hardest module for them is "COMP0003: Theory of Computation". This module introduces formal reasoning methods and begins with propositional logic. This topic was very unfamiliar to everyone as they had not come across anything similar in their previous studies. As a group, we had unanimously agreed that this was also an issue for us.

As a result, we decided to create a learning tool to help teach propositional logic for high school students. Our tool will especially help those who are not able to study computer science at high school level, since those students are often the ones who struggle the most when starting their computer science degree. It will also help current CS students who are revising for their logic modules. We will also try to picturise the game instructions as much as possible. This will break language barriers and allows all students to learn and have fun, regardless of their background.



The figure above shows the results of a survey asking UCL computer science students what their most difficult 1st year module is/was. Two modules: Design and Professional Skills, and Engineering Challenges are not included in the chart since they both received 0 votes.

4. Algorithm and List of Components

Algorithm to Generate Game Questions

Firstly, the algorithm randomly generates variables equal to the number of levels plus one. We have made it so that there is a slight chance that any variable after the first one could be the same as the previous one e.g., (A, B, A). This is done to increase the difficulty of the question. Secondly, random connectors are added between them as functions, selected from our pre-set list of connectors e.g., (A $\rightarrow$ B $\wedge$ A). The number of connectors is equal to the current level. This is done using a recursive function where a random connector is added for an atomic formula e.g., (A $\rightarrow$ B), at which point the atomic formula is seen as a single variable. This is done to randomise the order of formulas to solve, making it more difficult.

The valuation for the question is created by turning the list of variables into a set. This automatically removes duplicate variables. Then, iterating through the set, a random truth value (1 or 0) is assigned to each of them.

By replacing the variables with the truth values we set and applying them in the function, the answer is generated. It is then used to check the user's answer.

List of possible connectors:

$\wedge$,$\vee$, $\rightarrow$, $\leftrightarrow$

Question example:

Level 1 - A $\rightarrow$ B
Level 2 - A $\vee$ (B $\rightarrow$ C)
Level 3 - A $\wedge$ ((B $\vee$ A) $\leftrightarrow$ C)

Pseudocode of question and answer generation

Input:

- Current level number

Output:

- ValuationList – List of valuation for each variable, needed to display on screen.
- FormulaList – List of formulas as strings from smallest to its complete form e.g., ["(A $\vee$ D)", ... , "(A $\rightarrow$ B) $\wedge$ (A $\vee$ D)"], needed to display on screen.
- AnswerList – list of truth value/answer for each component up to the complete formula, e.g. [1, 0, … 0], needed to compare player's answer to

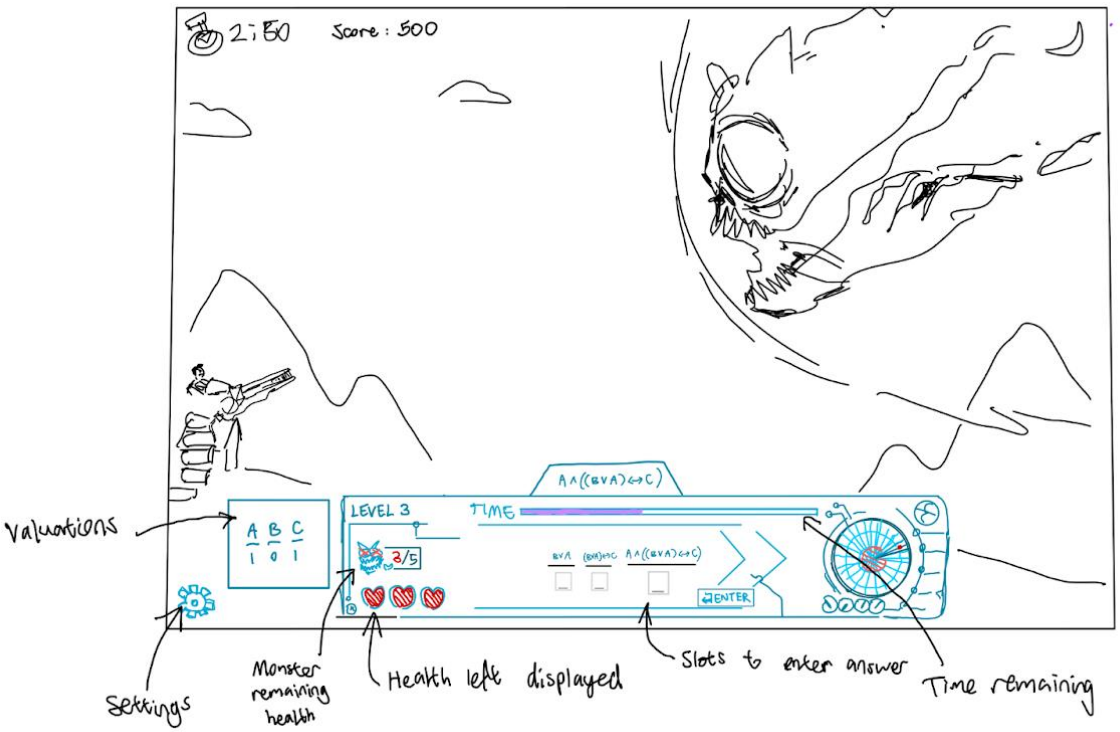
Constants:
List of connectors as strings – ["$\rightarrow$", "$\vee$", "$\rightarrow$", " $\leftrightarrow$ "]
List of connectors as functions (references to functions stored) – [implyFunc, …]
e.g., implyFunc(a, b) = A $\rightarrow$ B = $\neg$ A $\vee$ B

For level x (e.g., level 3)

1. Generate list V_s of 4 (x + 1) random variables.
   # There is a slight chance that any variable after the first one is the same as a previous one
   # V_s will be used to store the string form of the formula
2. Generate valuation for each variable.
   a. Remove duplicates by putting the variables in a set
      $[A, B, A, C] \rightarrow \text{set}\{A, B, C\}$
      ValuationList: (A: 0, B: 0, C: 1)
3. Iterate through V_s, create a new list V_v which replaces the strings of variables with valuations (int 0 or 1). V_v is needed later to find the truth value/answer.
   a. V_s = ["A", "B", "A", "C"]
   b. V_v = [0, 0, 0, 1]
4. Concatenate 2 adjacent variables using a connector as a string and add it to FormulaList.
   a) Choose x to be a variable in V_s, **that is not the same as the previous one**, and randomly pick a connector from conn_strs
      i. ["A", "B", "**A**", "C"]
      ii. Connector = "V"
   b) In the list V_s, perform chosen_variable = "(chosen_variable + chosen_connector + next variable)"
      i. ["A", "B", "(**A** $\vee$ **C**)", "C"]
   c) Remove variable after chosen variable.
      i. ["A", "B", "(**A** $\vee$ **C**)"]
   d) Add the current version of the formula to FormulaList.
      i. FormulaList ++ ["(A $\vee$ C)"]
5. Apply chosen connector function of the 2 variables' valuations and add the answer to AnswerList.
   a. In the list V_v, find the same variable we operated on in V_s and find the same connector as a function using their index.
      i. [0, 0, **0**, 1]
      ii. Fine the connector we randomly selected previously in the connector function list – Here "V" corresponds to orFunc
   b. Perform chosen_variable_valuation = chosen_connector_func(chosen_variable, next_variable)
      i. [0, 0, orFunc(0, 1), 1]
      ii. [0, 0, 1, 1]
   c. Remove variable after chosen variable.
      i. [0, 0, 1]
   d. Add the truth value/answer for the formula to AnswerList.
      i. AnswerList ++ [1]

6. Repeat steps 4 to 6 until V_s/V_v only has 1 element left.

Examples of Exercises



5. Framework and Language

When it comes to the choice of implementing our program, we decided to create a web application. This will make the product more accessible, as anyone will be able to access it from their web browser on different types of devices, such as laptops, tablets, and mobile phones. This is definitely an advantage over something like a desktop or mobile application.

We are going to use React.js, a JavaScript library for creating frontend user interfaces. It will allow us to create the layout, and dynamically update the elements on the website. React is component-based, and the code is logically split into components, reusable pieces of code defining the UI. React is a very popular technology supported by very good documentation, which will be helpful in creating as well as maintaining the application later on.

We chose to use SCSS to style the look of the interface, which is compiled to standard CSS but provides additional features like variables, nesting styles, functions, loops, and more. All of which makes the styling more organised.

We are not planning on building a server-side, and all of the application processing will happen locally on users' devices. This certainly has numerous advantages. The application is simpler to build, test and deploy. It can be delivered to more users at once with no bottlenecks

occurring on the server. We want to focus on delivering a great user experience by making use of the modern web browsers capabilities.

Most of our team already has some level of experience with web development, including HTML/CSS/JavaScript, and everyone is willing to learn and improve their skills. We are going to set up a GitHub repository and use Git as a version control system.