Coursework 2 (Frontend Development)

For this coursework, you are asked to implement a tower defense (`https://en.wikipedia.org/wiki/Tower_defense`) in TypeScript. Please follow the following steps to do so. If you have a reason to deviate from the described implementation, please document this and hand in the documentation. If you want to document anything additionally (like a deviating architecture), please create a small PDF file.

Exercise 1: Tower Defense: User Actions (14 Marks)

A. Create a basic React project (using `npm create vite@latest`). Create states that store the following things:
   - Whether the game has stated yet (initialize with false),
   - how much gold the user has (initialize with an appriopriate amount),
   - which tower the user currently chose (regular tower, ice tower, fire tower),
   - when the next wave starts (initialize with a frame count, e.g., 250 for 5 seconds assuming 50 frames / second), and
   - which level the player currently is at (initialize with 1).

   (3 Marks)

B. Please add an appropriate HTML element showing the user the current state. Additionally, implement a `ToggleButtonGroup` allowing the user to choose a tower (choosing the tower should obviously result in having changed the state). (3 Marks)

C. Add a canvas, draw its background and define a fieldsize as a constant (recommendation: 30 pixels). (1 Mark)

D. Please add a reaction on a click into the canvas:
   - Add a `Tower` class representing a tower and its current state, and add an array of towers as state,
   - calculate the x and y position based on your `fieldsize`, and
   - if enough gold is present, deduct the gold and create the tower (of the type that the user selected).

   (3 Marks)

E. Create a `draw` method to the `Tower` class, create a main loop that draws all the towers and let this main loop be executed with a certain frame rate. Let every tower type have a different visualization (as easiest implementation, just let them be rectangles with some color). (4 Marks)

Exercise 2: Tower Defense: Monster Behaviour (8 Marks)

A. Create a `MonsterPath`, that stores a `position` and follow-up position for each position. The last position can be empty (indicating that the goal is reached). (2 Marks)

B. Create a `Monster` that has a `position`, a `displayPosition`, a count of `lives` and a `MonsterPath`. Implement an `Monster.update` function that updates the displayPosition going to the next position in the `MonsterPath`. (3 Marks)

C. Create a `Monster.display` method that visualizes the monster (easiest implementation would be using a rectangle with a color different from the towers). Create monsters periodically (depending on the next wave status) and call the `Monster.display` method from the main loop. (2 Marks)

D. Implement that if a `Monster` reaches the last position of the `MonsterPath`, the monster is deleted and the `lives` state of the user is reduced by 1. (1 Mark)

Exercise 3: Tower Defense: Shot Behaviour (10 Marks)

A. Define a `Shot` class, that stores a `position`, a `type` and a `goal` (that is of type `Monster`). (1 Mark)

B. Implement that the towers have a `cooldown` that is reduced on every call to an (to create) `update` function. Call the `update` function from the main game loop. (1 Mark)

C. Implement that in the games main loop, for every tower with an appropriate cooldown and every monster it is checked for whether the distance between tower and monster is below some threshold. If the check is true, create a new shot of the `type` of the given tower. (2 Marks)

D. Implement that once a shot reaches a `Monster`, the lives of the `Monster` are reduced and the shot is deleted. (2 Marks).

E. Implement an effect of the shots for the ice and fire tower. Typically, the effect would be stopping the monsters movement (freeze) and reducing the lifepoints of a monster (fire tower). You can choose other behaviour if you consider this appropriate for the game balancing. (2 Marks)

Exercise 4: Additional Features (8 Marks)

Please add an additional feature of your choice. There are three aspects to extend: The user choices, the game loop and the monster behaviour. Please choose a features that influences **at least** two aspects (Every Aspect: 4 Marks).

Examples would be:

- A temporary block for monster that the user can place on the game for some time.
- The option to upgrade a tower, which increases the power of the shots.
- An option for the user to increase its lifes by reaching a special goal, e.g., saving 100 gold.

To avoid wasting time for writing boilerplate code, you are allowed to use all automated help (including ChatGPT). What you hand in will be considered **your** work. Therefore, make sure you understand your code and are able to produce the same quality of code in a coursework.

Submission: Finally, please upload your result (the **implementation** - excluding the `node_modules` folder and everything that can be automatically generated, and potentially the **documentation** in one .tar.xz or .zip archive) on Moodle before **Friday 15th of December, 15:00 German Time**. In case you have technical problems with the Moodle upload, please send your results to me via mail (d.g.reichelt@lancaster.ac.uk).
Please only include your student id and **not** your name in the submission.
Handing in after the deadline will be considered a late submission (and the penalties will be applied accordingly).