
## Task description

In this assignment, we will use the RxJS Observable stream explored in the Week 4 worksheet to create the [classic Frogger Arcade Game (YouTube)] in an SVG image hosted in the **dist/index.html** webpage.

The YouTube video is meant to give you an idea of the gameplay, but yours needn't look the same or work in precisely the same way, especially with regard to graphics. **Note that only a subset of the features shown in the video will be part of the requirements.**

**You will also need to write a report detailing the design of your game.**

# Requirements

**The game must be implemented in a good functional reactive programming style to get marks**. A subset of the game's features will be required to get a passing grade. Additional features will be required to get a higher grade. To achieve the maximum marks for this assignment, you will have to use a little creativity and add some non-trivial functionality of your own choice – see the additional information document for some ideas.

Correct collision behaviour:

- In the **ground** section, Frog may move/stand on the **ground**, but dies when colliding with a car object
- In the **river** section, Frog may move/stand on **plank objects**, but dies when landing in the water (ground)
- Frog dies when colliding with any enemies (e.g. snakes, crocodiles)

## Minimum requirements

All of these requirements must be reasonably executed to achieve a passing grade

- Frog which can move forwards, backwards, left, and right using one of the keyboard or mouse
- **Multiple rows** of objects **(at least 6)** appear and move across the screen
- Objects move at **different** speeds and directions (left-right)
- **Correct collision behaviour** (defined above) including at least one **ground** section and one **river** section
    - For minimum requirements, you do not need to include enemies
- Game ends when the Frog dies
- Indicate the score for the player
- Player scores points by landing the Frog in a **distinct** target area
- A 1-2 page PDF report detailing your design decisions and use of functional programming techniques discussed in the course notes

## Full Game requirements

Meets minimum requirements and has additional features

- Keeps track of high score achieved across previous rounds
- **Multiple** distinct target areas that must be “filled” (as per the video)
- At least **3 distinct objects** with different interactions/behaviours (e.g. crocodile, turtle, car/plank, snake, fly) that aren’t just movement
    - Cars and planks count as one distinct object
    - An example might be cars and planks, crocodiles, and turtles
- At least one of the “ground” and “water” sections, including at least one row in the middle where there are no objects (safe zone)
- Smooth and usable game play.
- **Able to restart when game finishes**
    - This must not be done by refreshing the page, and should also not be done by recursively calling the main function (you should use state management to handle this)
- The game increases in difficulty after some **non-score-based** condition is met (for example, landing the frog in 5 target areas)
- *See video for an idea of appropriate gameplay*

## Report

Your report should be 600-1200 words in length, plus up to 600 words for each significant additional feature, where you should:

- Include basic report formatting headings/paragraphs
- Include diagrams as necessary
- **Summarise** the workings of the code and **highlight** the interesting parts (don’t just describe what the code does, we can read the source code!)
- Give a high level overview of your **design decisions** and **justification**
- Explain how the code follows FRP style
- How state is managed throughout the game while maintaining purity
- Describe the usage of Observable beyond simple input
- **Important**: Need to explain **why** you did things
- **Do not include screenshots of code unless you have an exceptionally good reason**

We will be fairly lenient with word count, but excessively long reports may be penalised.