FIT2102 Programming Paradigms 2022 Marking Criteria and Suggestions

See the Assignment 1 Specifications in the Moodle Assessments section for the description of the task.

This document is not essential for completing the assignment, and is provided purely for context and additional information to answer common questions students may have.

Marking

The goal of this assignment is to assess your understanding of FRP and Functional Programming. The marking has three broad sections:

1. Implementation of game features
2. Use and understanding of proper functional programming style
3. Use and understanding of RxJS and Observable

It is important to realise that:

- If you implement the **Minimum requirements**, demonstrating application of functional programming ideas from our lectures and tutes, you will achieve a pass grade.
- You can receive up to a D for perfectly implementing the **Minimum requirements** and demonstrating an excellent understanding of how to use Observable to write clean, clear functional UI code.
- To achieve an HD, you will need to implement the **Full game requirements**
- To achieve the maximum possible marks, you will need to implement the full game requirements plus some aspect of **additional functionality** as described below.

Note that it is essential to follow the submission instructions, as a deduction of up to 5 marks may be applied for failing to follow the submission instructions.

Marking criteria in more detail

We will mark on 5 sections – Report, Functional Programming style, Code Quality, Observable and RxJS usage, Game Features (including extensions) – weighted equally.

Code that does not use Observable will **not** get a passing grade; games that use imperative, impure, or mutable code will be heavily penalised.

A general rubric has been provided below which gives a general description of what we expect at each grade point. **That is for reference and context only and is not the actual marking rubric.**

Report

The report is intended to demonstrate your theoretical understanding of functional reactive programming, highlight design decisions, and help your marker appreciate the work that you have put into this assignment.

The contents of the report should (from the specifications):

- 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**

Other important considerations for the report:

- Design decisions need to be correct
- Need to display understanding of course material
- Reports must demonstrate knowledge of FRP to achieve a passing mark
- **Marks can be awarded for students identifying issues with the code and how they can be addressed**
- Avoid filler in the report, but include enough information to show your marker that you have understood the core concepts

Functional Programming style

This section is about using what we have covered in lectures and tutorials. This involves concepts like:

- Small, granular functions
- Reusable functions, avoiding duplicate code
- Purity / referential transparency
- Fluent interfaces and fluent coding style
- Manipulation of different complex types and generic types
- HOF, curried functions
- Function composition/chaining

To achieve the maximum available marks, it is important to not only use advanced functional programming concepts, but do so in a useful way – for example, improving the readability of the code or following a declarative programming style. For example, simply currying all your functions will not receive marks unless they are partially applied somewhere and used appropriately

You may also attempt to use Lambda Calculus concepts in your code; however, be careful as they can often just make things hard to understand – it will be important to explain their usage in your report, so your marker can better appreciate your work.

Deductions will be applied for improper usage of types, including unjustified “any” types.

Code Quality

This section loosely covers anything to do with how readable and understandable your code is. Applying a good functional programming style tends to increase the readability of your code. **It is important that your code can be easily understood to help your marker appreciate your work.**

Some examples of what we look at are

- Appropriate line lengths (<80 characters)
- Documentation and commenting (should explain why the code is the way it is)
- Logical structuring of functions and variables, including overall flow of program logic
- Appropriate variable naming
- Consistent and understandable formatting

Using a linter and formatter may help greatly with this section. See below for tips and suggestions.

Observable and RxJS usage

This section covers usage of FRP – did you use Observable well?

Some important considerations:

- **Must include game state in Observable, and use the scan and merge operators to get a passing mark (please refer to the Asteroids example)**
- Usage of Observable as per discussed in the lectures, tutes, workshop, and in the Asteroids example, while maintaining purity, is sufficient for a High Distinction grade if implemented very well and without issues
- **To achieve the maximum marks available, we want to see extra usage (original work) of Observables and RxJS over Tim's Asteroids**
    - This can involve implementing custom Observables

Other considerations:

- Side effects should be contained as much as possible
- Using additional RxJS operators that are not covered in class, or using the ones we introduce in interesting and novel ways, will be awarded additional marks (given that they are appropriate and useful)

Game Features

This section is about whether your game fulfils the requirements, and the overall complexity of your game (and thus the implementation).

It is important to remember that adding features should not come at the expense of the other criteria – a well implemented game with fewer features may and, often will, achieve a higher mark than a less well implemented game with more features.

Some marking considerations:

- Minimum game requirements is roughly a Pass grade, and full game requirements is roughly a High Distinction grade for this section
- Extra features must follow FRP
- Extensions can be not just gameplay but extra FRP features too
- Tests: for full marks, tests need to be **comprehensive** and not just simple/random test cases – they should guide development
- Bugs and other gameplay related issues will be deducted from this section

To achieve the maximum available marks, features should be significant and change how state is managed in interesting ways. Discussed further below.

General Rubric (for reference and context)

Code/Report quality	**Frogger implementation**		
	Minimum requirements	*Full game*	*Full game + extension(s)*
Any of the following are not acceptable: Use of imperative code, TypeScript compile errors, `any` types, Not using rx.js, No comments, Missing or unreadable report, Missing instructions for how to play the game	Not passing.	Not passing.	Not passing.
Pure functional code (except in `subscribe` handlers), no compile/runtime errors, basic comments, basic report covering the implemented features.	P	C	C
The code leverages Observable, has generic types, and side effects are identified; comments only describe the code, are brief, or do not adequately explain why. In addition to covering the implemented features, the report demonstrates basic understanding of FRP principles.	C	D	D
Small pure functions, immutable data and reusable code exploiting parametric polymorphism, side effects are contained; complete comments explaining the rationale and choices made in code. Advanced usage of Observable, including custom implementations. Detailed report of implemented features that demonstrates strong understanding of Functional Programming and FRP.	D	HD (80-90)	HD (90+)

How to get an HD or High HD

To achieve a mark in the HD range, you need to implement a complete game with good style. To get in the high HD range, you will also need to implement additional features.

One or more of the following (or something of your own devising with a similar degree of complexity) done well (on top of the basic functionality described above) will earn you a high HD, provided it is implemented using the functional programming ideas we have covered in lectures and tutes:

- Create unit tests and create a file **tests/main.test.js** which are **comprehensive** and **guided the development of the program**
- Incorporate gameplay from other classic arcade games -- breakout, galaga, etc.
- Add power-ups or debuffs to the game (e.g. Frog can eat obstacles like in pac-man, double jump, increasing/decreasing size of obstacles)
- Advanced (not recommended unless you already know how): Make a distributed multiplayer version, wrapping the comms in Observable (you'll have to provide your own server for this).

In general, **additional features for achieving HD and high HD will have to non-trivially impact your state management and/or overall complexity of the game**. For example, a power-up that changes the speed of the Frog does not require interesting usage of state on its own, but if power-ups decay over time, then that would be more interesting and non-trivial.

Note that adding features will grant you a higher grade **under the condition that it is done in proper Functional and FRP style**. For an example of the proper style, refer to the example Asteroids Game described in the Course Notes.

For reference, an implementation that essentially ports Tim's Asteroids implementation to the full game requirements, maintains purity, and the same quality of code, would receive a Distinction grade.

A note on Observable

Usage of Observable within what is found in the Asteroids Example, tutorials, or workshop can achieve high but **not** the maximum available marks. **You will need to come up with interesting and creative uses for Observable and RxJS operators** which may involve implementing some custom Observables and research into the RxJS operators documentation.

Tips and suggestions

These are not part of the explicit requirements, but are things we may look at as part of the marking criteria. For example, poor choices of variable names may have an explicit deduction but may impact your code quality mark as it makes the code hard to read.

Tips for getting started.

- Complete the Week 4 Tutorial Worksheet Observable exercises and begin studying Observable in the course notes.
- In the week 4/5 Tutorial, you will complete **observableexamples.ts**
- Once you have completed the above, work through the example Asteroids Game described in the Course Notes. Follow the same framework to begin adding functionality to **main.ts** as above.

More tips.

- Finish all the JavaScript and TypeScript tutes (up to the first part of Week 5) and the course notes FRP material first. They are designed to give you the experience you need to prepare for this assignment
- Come to the workshops and tutes for important tips and assistance
- **Attend consultations given by the teaching team.** They are often sparse or empty around the time assignments are released, so it can be a great opportunity to get more detailed guidance and feedback
- Any general questions should be directed to the Ed forums when possible. However, try to avoid posting potential solutions. If you cannot make the consultations, you may make a ***private*** post for the assignment with your code.
- Your code should include brief comments to explain logic and design choices where necessary, or to refer to detailed explanations in your report. Please do not add comments that are self-evident from the code, e.g.
  - `const x = 1; // variable x is set to 1.`
- **Start as soon as possible**. Do not leave the assignment until it's too late.

Recommended coding practices.

- Structure your program in a consistent and coherent manner (group relevant functions, declarations, and variables together)
- Use block/section comments to clearly lay out each part of your code
- Use nice indenting and formatting (here are some common formatters prettier beautify)
- Use camelCase for names and UPPER_CASE for constants

Plagiarism

We will be checking your code against the rest of the class and the internet using a plagiarism checker. Monash applies strict penalties to students who are found to have committed plagiarism.