FIT2102 Programming Paradigms 2023

Assignment 1: Functional Reactive Programming

Due Date: 03/09/2023
Weighting: 30% of your final mark for the unit
Interview: Week 7 (week starting from 04/09/2023)
Overview: Students will work **independently** to create a classic arcade game using Functional Reactive Programming (FRP) techniques. Programs will be implemented in TypeScript and use RxJS Observable streams to handle animation, user interaction, and other similar stream behaviours. **The goal is to demonstrate a good understanding of functional programming techniques as explored in the first five weeks of the unit**, including written documentation of the design decisions and features.

- **It must contain all the code for your program** along with all the supporting files as well as the **report**.
- It should include sufficient **documentation** that we can appreciate everything you have done.
- You also need to include a report describing your design decisions.
- The only external library should be RxJS libraries supplied with the starter code.
- **Make sure the code you submit executes properly.**
- **Do not submit the node_modules or dist folders.**

The marking process will look something like this:

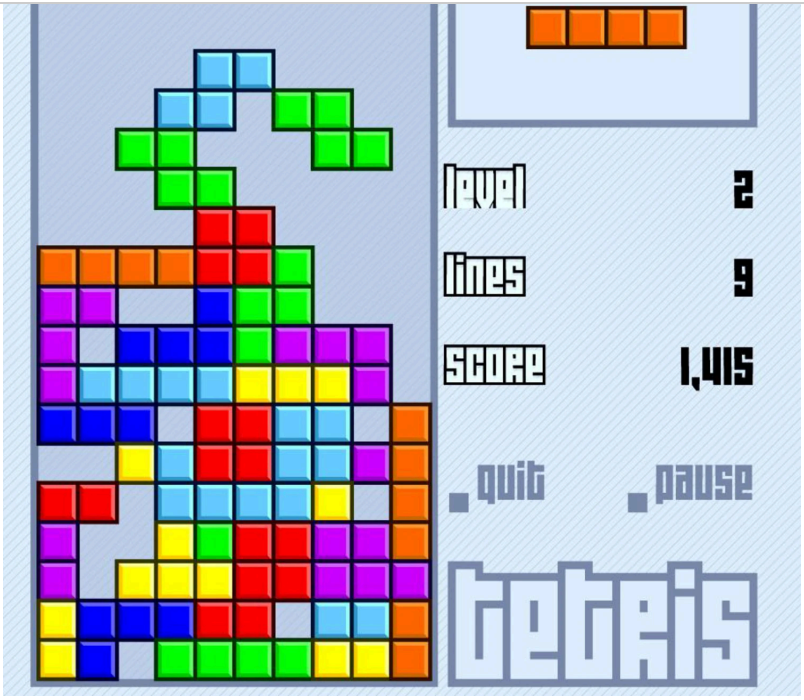
1. Extract **<studentNo>_<name>.zip**
2. Navigate into the folder named **<studentNo>_<name>**
3. Execute `npm install` and `npm run dev`
4. Open http://localhost:5173 in a browser

Please ensure that you test this process before submitting. Any issues during this process will make your marker unhappy, and may result in a deduction in marks.

Late submissions will be penalised at 10% per calendar day, rounded up. Late submissions more than seven days will receive zero marks and no feedback.

Assignment 1 2023 Specifications

Assignment 1 2023 Specifications



Task description

In this assignment, we will use the RxJS Observable stream explored from Week 3 to create the classic Tetris game in an SVG image. You will be provided with a starter code bundle similar to the applied sessions, including instructions on usage.

The image above and the Wikipedia page 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 discussed in the link will be part of the requirements.**

You will also need to write a report, as described below.

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. A greater subset of 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.

Minimum requirements

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

- A game board with minimum **10x20** minimum dimensions
- **Square** blocks **(2x2)** appear at the **top** of the board and move **down** in discrete increments
- Descending blocks can be moved in fixed increments **left and right** by the user using keyboard controls (constrained to the board)
- Blocks demonstrate correct stacking behaviour
    - There are no overlapping blocks
    - Blocks move down until at least one segment collides with another block below on the same x coordinate
    - Blocks stack as a single cohesive piece

Game ends when at least **one column** of blocks stack to exceed the top of the grid
- **Full rows** of blocks are cleared and add to the player's current score
- Indicate the score for the player
- A short 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
- All Tetris pieces that can be built using **4 blocks**
  - **Note:** Since the pieces are no longer limited to 2x2 square blocks, it is now possible to fill single rows
- Correct shape rotation, according to chosen rotation system
  - Clearly specify the rotation system you decided to implement
- Next shape preview
- Upcoming shape is randomly selected
- Smooth and usable game play.
- **Able to restart when game finishes**

management to handle this)
- The game increases in difficulty after some **number of rows** have been cleared
  - Increase block speed, or
  - Board starts with existing blocks/obstacles
- *See video for an idea of appropriate gameplay*

Additional requirements

See the Additional Information and How to get a High HD sections.

Report

Your report should be 300–600 words in length, plus up to 200 words for each **significant** additional feature, where you should:

- Include basic report formatting headings/paragraphs and 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 and interesting usage of Observable
- How state is managed throughout the game

- **why** you did things
- **Do not include screenshots of code unless you have an exceptional reason**
- This should be concise and straightforward, you may use dot points

Your marker will be instructed to stop reading if your report is too long, and only mark the first 600 (+200 per feature) words.

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. Additionally, we will be conducting an interview, which gives you a chance to explain your code and help us understand your code better. As long as you wrote your own code, there is nothing to worry about during the interview process.

AI statement

As per the AI statement on Moodle, use of generative AI in this unit is unrestricted. However, all code generated with AI **must** be properly **cited** in the form of code comments stating what has been generated and the scope of its use. You must

Additional Information: Marking Criteria and Suggestions

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

Marking (30 marks total)

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. Usage and understanding of proper functional programming style
3. Usage and understanding of RxJS and Observable

It is important to realise that:

- To can receive a **Pass** grade by implementing the **Minimum requirements**, demonstrating application of functional programming ideas from our lectures and applied sessions.

demonstrating an excellent understanding of how to use Observable to write clean, clear functional UI code.

- To achieve a **High Distinction**, 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 deductions may be applied for failing to follow the submission instructions.

We will mark on 5 sections – Report, Functional Programming style, Code Quality, Observable and RxJS usage, and Game Features (including advanced features) – that are individually weighted.

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

The rubric and marking guide are provided here.

Report (4 marks)

The report is intended to demonstrate your theoretical understanding of functional reactive

- 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 (8 marks)

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,

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 (8 marks)

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

Observable and RxJS usage (8 marks)

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

Some important considerations:

- **Must manage 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, applied sessions, workshop, and in the Asteroids example, while maintaining purity, is sufficient for a **high** mark in this section if implemented very well and without issues
- **To achieve the maximum marks available, we want to see interesting and creative uses for Observable and RxJS operators (original work)**
  - This can involve implementing custom Observables and research into the RxJS operators documentation
  - Refer to the marking guide for a breakdown of what is required.

Other considerations:

- Side effects should be contained as much as possible
- Using additional RxJS operators that are not

Game Features (2 marks)

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

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.

Important: You will receive marks for implementing game features, but **this mark will also cap your total mark**.

- The maximum mark possible for implementing **minimum game requirements** is 70 (Distinction)
- The maximum mark possible for implementing **full game requirements** is 90 (HD)
- To achieve the maximum available marks (90+), you must implement **advanced requirements**

Some marking considerations:

- Extra features must follow FRP
- Advanced requirements can be not just gameplay but extra FRP features too
- Tests: for full marks, tests need to be **comprehensive** and not just

and be deducted from the total mark

- The total mark cap will be increased when implementing additional features. It is *possible* to achieve an HD by implementing the minimum game requirements and *some* full game requirements

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

Bonus marks are available for particularly novel, impressive, or advanced features. Note that marks cannot exceed 100% of the total available marks.

Rubric

The rubric consists of **Marking bands** that represent the possible grade values for implementing requirements. This will be a **cap** on your final mark.

The **Marking guide** is what TAs will be using to mark your assignment, and what will contribute to your final mark/grade for this assignment.

Assignment 1 2023 Specifications

marking guide)

Code/Report quality	**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. Uses Observable for state management.	P	C	C
Effectively uses Observable for state management, has generic types, and side effects are identified; comments are brief, only describing the implementation. The report demonstrates basic	C	D	D

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.			

Marking guide

	FP Style	Code Quality	Observable
0 - 1 mark	Code is written in an imperative style, use of for/while loops and mutable variables (let/var). Modifies mutable data structures that aren't declared as read only to handle state management. No use of FP	Code is completely unreadable. Contains very large code blocks with complex nested logic and long lines. Excessive use of single letter and/or vague function names.	No use of observables. Uses DOM to store state or does not use Observable to store state.

Assignment 1 2023 Specifications

	modify state.	for variable names and many examples of complex nested logic with lack of documentation.	outside subscribe.
3 marks	Demonstrates some understanding of FP. Code contains some impure code. Use of HOF, but not utilised effectively.	Reader is able to get a general idea of code, but is difficult to read. Contains long lines and large code chunks. Some attempt at using functions and splitting up complex logic.	Uses Observables to handle state management and user interaction. Some Observable methods are not used effectively or not as intended, which demonstrates a lack of understanding.
4 marks	Style and structure is adapted from Asteroids example, but is not adapted to fit Tetris. Code is entirely pure.	Able to get the general idea of code. Contains many complex structures, and large chunks of code that require refactoring. Minimal documentation	Uses observables to handle state management and user interaction. Uses subscribe to handle stream logic; overuse of subscribe callback.
5 marks	Similar style to the asteroids example, effectively adapted to new context. Code is entirely pure and utilises the state management system introduced in the Asteroids example.	Can tell the purpose of each piece of code. Contains documentation, but some comments are redundant. Some long lines and large blocks, but generally minimised.	Good use of basic Observables from the unit. Some methods in the Observable stream are overly complex and can be broken down more appropriately.

Assignment 1 2023 Specifications

	functions and HOF. Shows great understanding of course content.		
7 - 8 marks	Applies FP concepts in original ways beyond the Asteroids example. Great use of HOF, modular functions and a custom type system. Demonstrates fantastic understanding of course content in novel and interesting ways.	Code is easy to read, intuitive and flows well. Self documenting (descriptive variable names, easy to follow code flow). Well documented and comments are provided when needed. No long lines, and code is broken into readable chunks.	Uses interesting Observable methods not covered in course content. Uses custom Observables/Subject.

	0 - 0.5 marks	**1 mark**	**1.5 marks**	**2 marks**
Report	Not written or does not correspond to submission. Provides a summary of the code. Contains some justification, but focuses too much on summarising code. Contains too many screenshots of code.	Provides a summary of code with reference to FRP principles followed. Demonstrates some understanding of FRP and how it was used to manage state. Some justification for design choices with some focus on why.	Clearly written and concise. Provides a good summary of code. Design choices are justified and considers tradeoffs. Relates design choices to FRP and course content. Good understanding of FRP and pure state management.	Clearly written and concise. Highlights only key aspects of the code. Strong understanding of FRP and how it is used to manage state. Design choices are well justified, and considers non-trivial alternatives and tradeoffs.

Features	**Marks**	**Running total**	**Classification**

Descending blocks can be moved by user	**0.25**	0.5	**Minimum (≤ 70)**
Game ends on full column	**0.25**	0.75	
Score	**0.25**	1	
Stacking	**0.5**	1.5	
Row elimination	**0.5**	**2**	
Random next shape	**0.25**	2.25	**Full (≤ 90)**
Next shape preview	**0.25**	2.5	
Difficulty increase	**0.25**	2.75	
Restart	**0.25**	3	
All 4 block types + rotation	**0.5**	**3.5**	
Advanced feature	**0.5**	**4**	**Advanced (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 **advanced 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

tests/mainTests.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. Special blocks that when cleared have special effects, etc.)
- Instant replay
- Tetris wall kicks
- *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 blocks 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.

may look at as part of the marking criteria. For example, poor choices of variable names may not 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 3-5 RxJS exercises and begin studying Observable in the course notes.
- 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 exercises and the course notes FRP material first. They are designed to give you the skills you need to prepare for this assignment
- Come to the workshops and applied sessions 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

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 options prettier beautify)
- Use camelCase for names, UpperCamelCase for types, and UPPER_CASE for constants

Changelog

07/08/2023	Specifications released with marking guide.
02/09/2023	Updated marking instructions to include npm install

Assignment 1 2023 Specifications