Name: Nisha Shaline Kannapper
Monash ID: 31121993

FIT2102 S22023 Assignment 1 - Report

Section 1: Code and Game Overview (167 words)

When main is called:

- Observable created.
- Game state updated by transforming emitted values.
- Subscription to transformed values renders game state.

"tick" (transforms emitted values):

- If given instruction is:
  - "restart" - game restarts.
  - "calm-mode" - game activates "calm mode".
  - "pause" – game pauses/unpauses.

- State is checked to see:
  - If game is starting/has ended
  - If current tetromino is frozen
  - If current tetromino should move
  - If the game should freeze or start a new round.

- If check says
  - currently moving tetromino can move.
    - state updated to reflect movement.
  - state can update, update state when:
    - round should start (new tetromino generated)
    - tetromino is "frozen" (cannot move anymore)

Interesting code:

- tetrominos only frozen when "tick" receives valid value from Observable emitting values for specific time interval, not at exact moment tetromino can no longer move downwards.
  - allows players to slide blocks.

Features:

- Full game requirements
- Audio:
- Pause:
- Calm/Zen mode

Section 2: Design Decisions (278 words)

Game components:

- State: current game state
- Board: board representing Tetris grid
- Tetromino
- Block: one square in tetromino

Game logic:

- Tetrominos:
    - use Sega rotation system.
    - stops moving one tick after reaching bottom/block below it.
- Game ends if state:
    - Has blocks in board top row.
    - Cannot load a new tetromino without overlapping with existing blocks.
- If board has full rows
    - full rows removed.
    - score += 1 per removed row,
    - rows above full rows moved downwards to fill space.
- Difficulty is increased at score 2, 4, 6 (4 levels)
    - Difficulty increase reduces time interval by 100ms - tetrominos fall faster.
- Calm Mode (additional feature)
    - Press "Z" to toggle.
    - Changes tetromino generation
        - Replaces current tetromino with O or I piece.
        - Only generates O or I pieces while mode is active.
    - Changes music, audio cues
    - Changes time interval – tetrominos fall slower.
- Pause (additional feature)
    - Press "R" to toggle.
    - Tetrominos
        - no longer fall every time interval.
        - can move upwards with user input.

User input

- Captured with Observables
- Processed by mapping keypress events for keys to Instruction (custom type)
- Using Observables to change game state follows FRP principles.

UI/UX

- Controls are listed on left of Tetris grid.

- Audio (additional feature)
    - Separate music tracks play depending on difficulty.
        - Calm Mode: one track plays regardless of difficulty
    - Specific audio track plays when game ends
        - Default: music track, laughter sound effect
        - Calm Mode: music track
    - Audio cues play when:
        - tetromino can no longer move.
        - row is cleared.

Section 3: FP/FRP Principles (157 words)

Submission code (excl. functionality involving rendering visual/audio elements) uses/has:

- pure functions (FP)
- only immutable data (FP)
- curried functions to compose other functions (FP)
    - Used to generate functions for state generation/updating, audio functionality.
- referential transparency (FP)
    - Pure functions are used, which fulfils this.
- declarative style (FP)
    - Code is more readable.
    - e.g., HTML file defines layout, structure, elements, without specifying fixed positions or how to manipulate elements.
    - e.g., "main" function defines game loop with "scan", "tick", expresses game state changes without state management details.
- event streams (FRP)
    - Creates Observables to use in changing game state.
- time-based computation (FRP)
    - Used time interval Observables to update state.
- reactive programming (FRP)
    - "filter", "map", "reduce".
- continuous time (FRP)
    - Time interval Observables represent continuous time.
- high-level abstractions for observable composition (FRP)
    - "merge" combines keyboard input , time interval Observables for main game loop Observable.

Section 4: Observables Usage (40 words)

Event-driven communication

- Observables for keypress events, time intervals allow reaction to events as they occur.

Asynchronous Operations

- Allows asynchronous operations to be managed for smooth gameplay.
- Allows various game components to coordinate.
    - no globals needed: modular.

Section 5: State Management (37 words)

Purity:

- How?
    - State is immutable.
    - Pure functions used to create new states when changes occur.
- Why?
    - No side effects, output is deterministic.
        - Necessary for debugging, output of functions easier to anticipate.