**FIT2102 A1**

**Report**

Chua Shi Xiang

33520879

**Minimum Requirements**

Circles appear from the top of board and move down in a continuous manner, where each circle aligns with the music note

* A interval observable is used to emit numbers periodically in time
* These numbers is mapped to a function which ticks over all four line, returning a new state with all of its circles’ y value incremented based on its speed

Notes can be played by using keys for each note when the circles align with the bottom row.

* Key input is observed using a fromEvent observable, which is then mapped to a function which handles the logic and returns the updated state

The timing of the notes must align with the given CSV file.

* The content of the csv file is used to create an Observable, which is then delayed in pipe using the delay operator.

The notes not for the current instrument, will need to be played in the background (i.e., without being displayed on the gameboard) at the correct time.

* An attribute (music), which has a function as a type, is added to the state
* The function that is given to this attribute is a curried function (playSound), which first accepts a Music, and then accepts a sampleLibrary.
* When a music needs to be played, this attribute is partially applied to the curried function playSound function to generate a function which plays the sound (which is only called in subscribe)

Appear heuristically (a simple heuristic will suffice) across all four columns

* From the 4 available lines, we filter out lines that already has a note in the y location, we then select one of the line based on the pitch.

Notes disappear when they have been played

* In the logic of keypress, if a note is detected within a certain range, it is removed, and its associated music is used to create a partial playSound function
* This partial function is then set as the ‘music’ value in our state, to be played in subscribe

Each note is played for the correct duration in which they are played.

* For each note, the duration can be calculated using getDuration.
* The function triggerAttackRelease is used to play the note with the correct duration

If the key press does not correctly align with a note, it will be played for  
a random duration between 0 and 0.5s

* Key presses that occur when the note isnt in a certain range is considered “missed”, and a random pitch is generated using randomPitch

Each time a key is pressed

* The correct note must be played if the circles align with the bottom row
* Otherwise, a random note is played.

Scores must be kept track during the game, for both hitting and missing notes.

The game should end when the song finishes playing.

**Full Game requirements**

If the note is longer than one second, the notes must have tails, where the tail  
represents the length of the note.

* This is implemented by extending the Note type to include isStream and endY value

The user must hold down the correct key for the length of the tail to ensure it  
is ‘correctly’ played. The score will update, if the note is played for the correct duration. If the player lets go of the key too early, the note stops playing

* This is implemented by extending checkReleaseDetection and checkHitDetection

A score multiplier must be included, starting at 1x and increasing by 0.2 for  
every 10 consecutive notes hit

* This is implemented by extending GameData type to include the combo, the multiplier can then be calculated via the combo’s value

**Additional Requirements**

Ability to restart a game

* A ‘retry’ attribute is added to state
  + When set to true, it will unsubscribe from itself in the old subscription, and calls generateGame again to generate a new subscription, effectively restarting the game
* An attempt was made using takeUntil (for the main source) and finalize (for the retry and back Observables)
  + This failed to work, as finalize is called on every observable when it is unsubscribed from, which takeUntil will call on the source Observable and all observables in takeUntil, causing the finalize function in retry and back to be called together.

The ability for users to choose a song for your game

* A list of playable songs is added into the constants due to not having a server to list out playable songs
* HTML div is then created for each of the playable songs, as well as an observable from the click event on the divs.
* The observables are then merged into one observable and subscribed to render the game whenever one of the divs are clicked