FIT2102 Assignment 1 Report

Go Wei Zheng

33273197

## Types, Constants

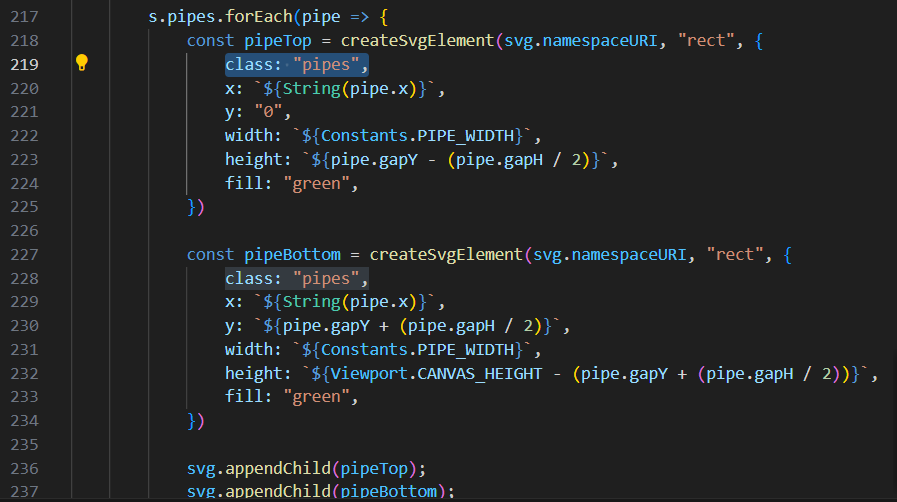
New constants and types have been added to prevent rewriting the same values. Constants.TICK\_RATE\_MS changed to 20 to simulate smoother animations for the birb and pipes.

* Constants
  1. PIPE\_SPEED: Determines the speed of the incoming pipes
  2. GRAVITY: How much y velocity the birb is losing each tick
  3. MAX\_VY: Maximum fall speed that the birb can achieve
  4. SEED: Seed for pseudo-random number generator
  5. INVINCIBILITY\_TIME: Invincibility duration when birb loses a life
  6. TICK\_MS\_RATE: 500 → 20
* Birb
  1. CX: Centre x position of the birb
* Types
  1. Pipe: attributes {gapY, gapH, time}

New types were defined to hold the values that is obtained from parsing map.csv. Type Key altered to be union of string literals to allow additional string literal “KeyR”. This will be used to help implement the restart logic of the game. Additional attributes are also added to type State to allow for feature implementations.

* State
  1. birdY: Y position of birb
  2. birdVy: Y velocity of birb
  3. pipes: Pipes to spawn
  4. lives: Birb lives
  5. invincibility: Invincibility duration of birb
  6. score: Current score

## Rendering (under render())

Rendering of the birb element is done by repeatedly removing the old birb element and creating the new birb element using the values that is emitted by the stream. This creates an effect that the birb is moving and falling while also avoids creating duplicate elements of birb. Similarly, the rendering of the pipes is also done in the same way. One slightly different detail is that all pipe elements created are classed under “pipes”. This allows the document.querySelectorAll to collect all pipe elements without the need of assigning unique id to each pipe which was what I initially tried to do. The pipes that have passed to the left end of the screen do not get created again, this prevents redundant elements from stacking up behind the scenes.

## Accumulating and updating state

Under state$, three sub-streams: tick$, flap$, pipes$ are created each handling a different logic of the game. Tick$ handles the updating of the state every 20 ms, this includes handling collision, gravity simulation, invincibility duration, lives, scores, which pipe to keep rendering and whether the game is over. The tick$ stream in each event returns a new state, this keeps the tick update logic pure as no existing data is being manipulated. Function collided is defined as it can be reused for different types of collisions: colliding with top or bottom of the border, colliding with the pipes.

Flap$ is the stream that listens for user input and allows the birb to flap, pipes$ emits pipes’ data that is passed into tick$ to allow for pipe spawning. The decision to separate these logics allow for easier knowledge of which logic is happening in which. It also allows for easier debugging.