FIT2102 Assignment 1: Flappy Birb
Dilhan Ficker Karci, 33906750

Overview:
Implemented a flappy bird clone using typescript and the RxJS library with functional reactive programming principles. This separates state, rendering and animations (GhostManager, Pipe). I have implemented a Ghost bird to track the movement of the current bird's state, to replay them when the current bird loses a life.

State:
State Objects: birdY, vy, lives, score and gameEnd.
The scan method will accumulate state over time producing a new state for each and every tick in the game.
Event Management: “R” for restarting initial State. “Space” and gravity merged into single stream for flap motion.
Collision Management: Constant checks for isColliding every tick to see if the bird is ever hitting a pipe. Lives decrement on collision, or out of bounds. If lives == 0, game over.
State stays pure as rendering and DOM is not focussed in this area of my code.

Rendering:
Birds and pipes are SVG elements being rendered every tick. Game over text printed on screen when gameEnd is true.
Ghost Birds: Rendered semi-transparent, replaying previous runs via setInterval stopped completely if game ends then stopAllGhosts() ensures no lingering intervals. Rendering logic is isolated from state updates, aligning with FRP principles

Pipes & Animations:
Pipes are dynamically generated with random gaps (gapY), and move from right to left on screen. Off screen pipes are removed from DOM. animatePipes has helper functions such as resetPipes for restarting, and isColliding for collision checks. Pipes have spawn threshold keeping consistency of the game.

Observables:
flap$: maps “Space” to upward velocity.
tick$: interval produces gravity.
restart$: restarts the game “R”.
Merged Streams: movement streams combine flap, tick, restart using merge.
Scan will accumulate and transform state over time. Map will extract relevant properties to be rendered.

Ghost Bird:
Record Y position for every tick, saved to currentRun. On life lost, GhostManager.spawnGhost replays position with the bird object, displayed with opacity making it appear as a ghost. Ghost animations are tracked with SetInterval and tracked in ghost Intervals.

stopAllGhosts: will halt all intervals of ghosts, if gameEnd. The idea of ghost Bird's implementation is to show previous runs so the current ghost can try to outperform its past run.

Design Decisions:
Immutable State for managing bird movement: pipe movement, and current states of the game.

Parse pipes from CSV: random spawning gaps between the pipes, also splitting the pipes allowing for the bird to travel through. Lower 'x' on each tick to give the smooth animation of pipes moving off the screen.

Collision Detection and jump logic**:** as states are constantly checking for collision with pipe, or off the screen. Checks bird position against the pipes positions.

Scoring and restart: lives attribute being constantly updated as bird travels through gap of pipe. If lives == 0, the game is over, and text will show, and the bird will respawn in default position. Game end triggers global cleanup of ghost intervals, maintaining memory safety and correct rendering.

Ghost Bird: record bird positions as an observable stream during gameplay. In the new game, create a Ghost Bird observable that replaces previous positions with the current game tick. Render in parallel with the current bird showing. Ghost bird integration demonstrates FRP usage: parallel animations derived from past state streams.

Conclusion
This project uses FRP with RxJS to manage Flappy Bird's state reactively. Observables handle input, gravity, and ticks while maintaining purity. Ghost birds and pipe animations showcase parallel with each other and demonstrate clean, modular, and reactive game design.