The A-Mazing DS4 Race

LAB #8

SECTION #3

James Gaul

SUBMISSION DATE: 4/12/24

LAB DATE: 3/29/24 - 4/5/24

Problem

For this lab, we were first assigned the challenge of creating a function to determine a moving average from the Dualshock inputs, then using this average to create a simple maze game.

Analysis

This problem could be broken down into a subset of individual coding challenges. Some of these included: creating a function to generate a random maze based on the given difficulty, creating a function to check for collision in the various directions the character could possibly go.

Design

Section 1 Questions:

1. Explain the differences between the raw data and the averaged data in your graph for part A.

   The raw data quickly responds to movements of the controller, but also is highly erratic, quickly swinging between values. The moving average data is slower to respond, since it is being averaged by previous inputs, but also is far more consistent in its motion.

   This is evident in the movement graphs (image 1, 2, and 3), with the moving average forming a smoother line amid the highly variable raw data.

2. Explain the delay you used to ensure character movement is not erratic.

   I used a buffer length of 8 inputs. I experimented with a variety of lengths, and found that this provided enough time to "smooth" the data, without adding excessive delay between a motion and change to the average input.

Section 2 Questions:

1. Describe how you checked if the avatar could safely move down, and go left/right.

   To check if the avatar could move in these directions, I created three Boolean functions, collidesLeft, collidesRight, and collidesDown. Each of these functions checked the next value in respective direction for the maze array. If the position was a wall or outside of the bounds of the maze, it would return true. Otherwise, it would return false.

2. Describe what was necessary to check for the player losing the game.

   In order to check if the player had lost the game, I created a "lostGame" function, which made use of the previous collide functions to determine if the player had lost the game. When the collidesDown function returned true, the function would then use the collides down function to the left and the right. If either path was open, it would then proceed to run the collidesDown function for this new tile, to check if there was an available path for the avatar to use.

If it ever found such a path downwards, the lostGame function would return false. However, if it found collisions on both the left and the right without collidesDown ever returning false, then lostGame would return true.

Testing

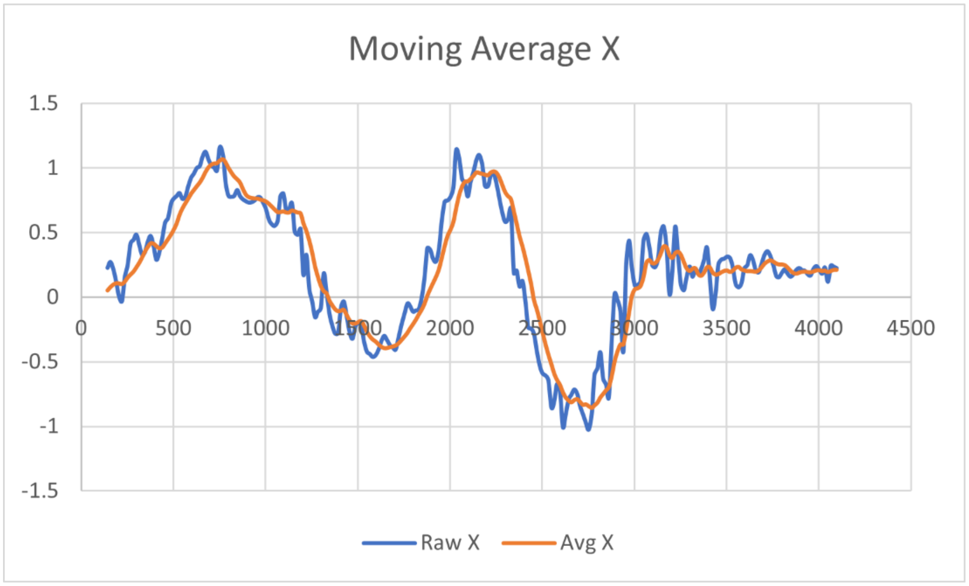
I experienced considerable difficulties with my attempts to add different "speeds" based on controller tilt. Trying to move the character more than one space repeatedly caused it to "jump" over obstacles, particularly when moving diagonally. Ultimately, I chose to limit motion to one space at a time, which greatly simplified the collisions.

Comments

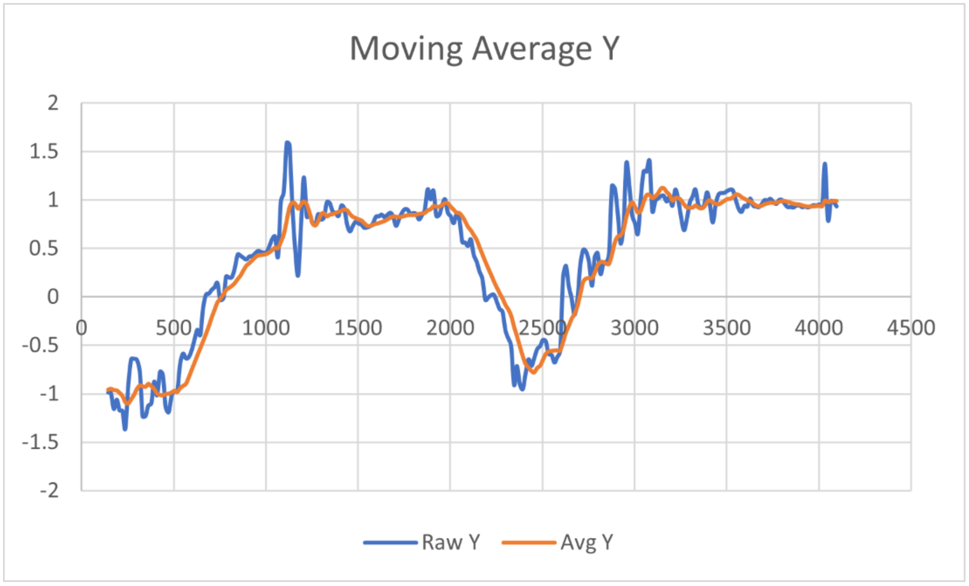
I experienced considerable difficulty trying to convert the command line input from a character string to an integer.

Screen Shots

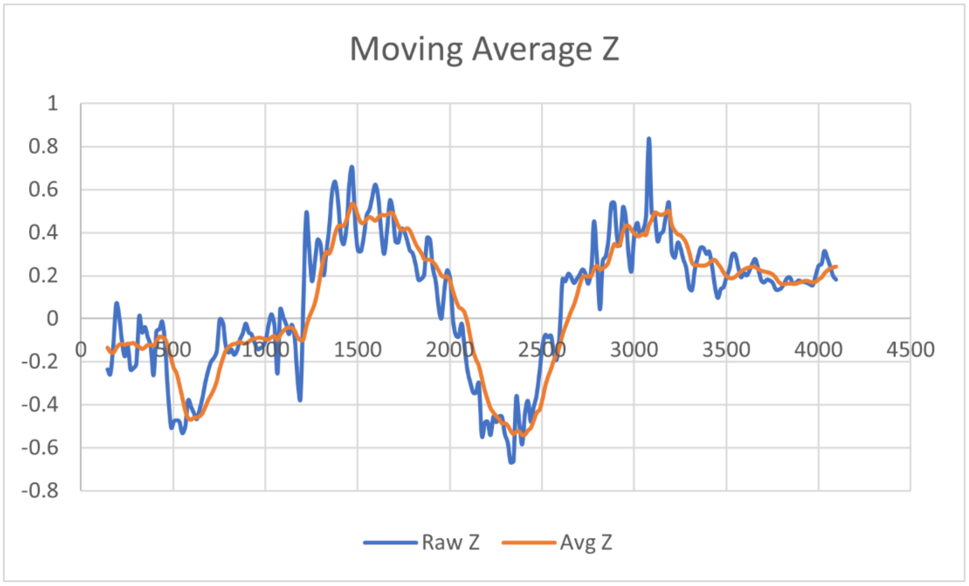
1. Moving Average X Axis



2. Moving Average Y Axis



3. Moving Average Z Axis


Moving Average Z
Raw Z
Avg Z