Race Game Arduino Report

An analysis of project code and implementation

Author: Zeyad S. M. Moustafa (zes3)

Tutor: Andy Starr

Date: Semester 1, 2022/2023

Table of Contents

Introduction

The code is a program for a car game written in the Arduino C programming language. The game is played on an 8x8 grid of LEDs, with the player represented by a green LED at the bottom of the grid. The player can move left and right using buttons and must avoid red LED obstacles that scroll down from the top of the grid. The player starts with three lives and loses a life each time they collide with an obstacle. The game ends when the player loses all their lives. The code includes functions for rendering the player, obstacles, and lives, as well as functions for moving the player and updating the game state.

Model

For this task I've followed the state machine example given to me during the practical classes (and adding some features explained in previous projects and worksheets) which consists of creating the main loop which is the "engine" of all the main functions to calling the states used to create the framework then listing them and adding the functions that helps these states to perform as desired. As asked by the assignment brief, I had to create the three basic states but also the other two which are the "Invalid" and "Pause" states.

This code is defining a player model and some associated functions for a game. The player has a position on the screen represented by the "player" variable, and a number of lives remaining represented by the "playerLives" variable. The "initPlayer" function initializes the player's position and number of lives at the start of the game. The "removePlayerLife" function reduces the number of lives by 1 and returns a boolean value indicating whether the player is now dead (i.e., has no more lives remaining). The "movePlayerLeft", "movePlayerRight" and "renderLives" function is used to display the number of lives remaining in the form of 3 lights on the screen.

The "*writing*" function adds a yellow block to the blocks array at a specified position. The generation function randomly generates a new row of blocks at the right edge of the track. The scrolling function shifts all the blocks in the blocks array one position to the left, effectively scrolling the track.

In this code, the "*hasPlayerBeenHit*" function returns true if the player's position coincides with a block in the blocks array that is not empty (i.e., not 0). If the player has been hit, the "*removePlayerLife*" function is called to subtract a life from the player. If the player is out of lives, the game transitions to the S_END state, otherwise it transitions to the "*S_LIFELOST*" state.

In this game, the track is a series of obstacles that the player must navigate through by controlling a cursor on the left side of the screen. The track is represented by an 8x8 array called `blocks`, in which each element represents a single position on the screen. The track is generated and updated using the `generation` and `scrolling` functions. The `generation` function randomly generates a new row of blocks at the right edge of the track, while the `scrolling` function shifts all the blocks in the `blocks` array one position to the left, effectively scrolling the track.

The score is stored in a variable called `score`, which is incremented by one every time the `updateModel` function is called a certain number of times (in this case, 20, 40, or 60 times). The player's score is displayed on the screen and is used to determine the player's ranking at the end of the game. The player's highest score is also stored in a variable called `highScore`, which is saved in EEPROM memory and displayed on the screen during the `S_START` and `S_END` states.

Problem Encountered

During the development of this game, I've encountered few problems with the code. The first problem was choosing the design of the track and writing the code for it. Since I've never coded for this type of technology (Matrix Shield), I had to try many ways to implement it and came up with two different designs which are: having two red lines on the sides and the other design is using the while screen as track but personally I prefer the second design.

The second problem I encountered was making the obstacles scrolling down. I've tried many ways of doing it but there is a hidden mistake which is not letting the code to run correctly so there were just one obstacle which isn't moving.

Finally, the "pause" function and finding the error that prevents many functions to not work properly along with the other functions, but the code has every single function but unfortunately, I wasn't able to make them work all together.

Your Assessment of the mark you would award yourself (based on the marking scheme)

I applied the programming skills I've developed over the previous ten weeks when writing my code. I logically divided my code among modules that would be executed several times over the course of the code. I tried to utilise as few global variables as possible.

I believe that fewer global variables could have been necessary, but since many values had to be accessible throughout the programme, they were necessary.

Additionally, I believe that if I had more time, I could have figured out how to show the obstacles on the first track.

I do think, though, that my game's programming was effective and employed the right techniques and I would give a 80% as an overall grade for the assignment.