Code Companion Report

CAB202

Matthew Hutchison

n8548625

June 3,2018

Executive Summary

The second assessed tasks seeks to recreate and improve upon the first assignment. Whereby this rebuilt 'Race to Zombie Mountain' arcade racer game, will need to run on the Teensy microcontroller and 'TeensyPewPew' breakout board. Leveraging the breakout boards, joystick, switch's, LED's and 84 x 48 LCD panel, the player will once again control the race car. Guiding the car towards the finish line, past or through obstacles and without running out of fuel.

Unlike the first assignment, assignment two instead has three parts. In this attempt all basic functionality has been implemented ***(Part A)***, as well as three out of four tasks for the extended game functionality ***(Part B)***. The final part 'Demonstrate Mastery' ***(Part C)***, was of greater difficulty and of the the seven tasks, two were able to be implemented. The code for this exercise has been submitted to the AMS assignment portal and the submission reference number is: *1d1568f5-21ba-4cc3-8915-6323448550e8*.

For a detailed breakdown of the above mentioned assignment parts/tasks, three tables have been created. Each task will be in one of four sections, detailing in what state of completeness it is in. Tasks that have been labeled as no attempt, will not feature a subsection going forward, throughout the report:

- **Completed:** If marked as complete this results in all required criteria being met met and features implemented.
- **Almost:** This state of completeness implies that not all tasks were implemented, but a great portion were. In addition, this also implies that these features that have been implemented are all in working order.
- **Started:** This state references tasks that have been outlined, basic code framework created, variables and sprites included, etc. Aside from implementing theses features the code in this stat is not in working order.
- **No Attempt:** These means that no working has been supplied for this task or that no attempt was made for these tasks.

PART A: PORT BASIC GAME - COMPLETION STATUS				
TASK	COMPLETE	ALMOST (75%)	STARTED	NO ATTEMPT
Splash Screen	☑	☐	☐	☐
Dashboard	☑	☐	☐	☐
Paused View	☑	☐	☐	☐
Horizontal Movement	☑	☐	☐	☐
Acceleration and Speed	☑	☐	☐	☐
Scenery and Obstacles	☑	☐	☐	☐
Fuel Depot	☑	☐	☐	☐
Fuel	☑	☐	☐	☐
Distance Traveled	☑	☐	☐	☐
Collision	☑	☐	☐	☐
Game Over Dialogue	☑	☐	☐	☐

PART B: EXTENDED GAME - COMPLETION STATUS				
TASK	COMPLETE	ALMOST (75%)	STARTED	NO ATTEMPT
Curved Road	☐	☐	☐	☑
Accelerator & Break	☑	☐	☐	☐
More Realistic Steering	☑	☐	☐	☐
Fuel Levels Increase Gradually	☑	☐	☐	☐

PART C: DEMONSTRATE MASTERY - COMPLETION STATUS				
TASK	COMPLETE	ALMOST (75%)	STARTED	NO ATTEMPT
Use ADC	☐	☐	☐	☑
De-bounce all switches	☑	☐	☐	☐
Direct Screen Update	☐	☐	☐	☑
Timers & Volatile Data	☑	☐	☐	☐
Program Memory or PWM	☐	☐	☐	☑
Pixel Level Collision	☐	☐	☐	☑
Serial communication & file system	☐	☐	☐	☑

CONTROLS - EXIT SPLASH SCREEN		
ACTION	BUTTON PRESS	DESCRIPTION
Exits splash screen (OPTION 1)	left Button (Switch 2)	Initially upon starting the game the splash screen will be displayed. To exit this screen and begin playing the game you can choose too press the left switch also known as switch 2 (SW2).
Exits splash screen (OPTION 2)	right Button (Switch 3)	Initially upon starting the game the splash screen will be displayed. To exit this screen and begin playing the game you can choose too press the right switch also known as switch 3 (SW3).

CONTROLS - CONTROL RACE CAR/ OPEN PAUSE SCREEN		
ACTION	BUTTON PRESS	DESCRIPTION
Accelerate race car	Joystick up direction (press)	To accelerate the race car (increase the speed of the car), press the joystick in the up direction. This will take the speed of the race car to its maximum over a period of 5 seconds.
Gradual race car deceleration	Joystick down direction (press)	To gradually reduce the speed of the race car, press press the joystick in the down direction. This will reduce the cars speed by one each time pressed.
Apply race car brake	Joystick center position (press)	To quickly decelerate the race car (decrease the speed of the car), press the joystick in its center position. This will take the speed of the race car too a speed of zero over a period of 3 seconds
Move race car left	Joystick left position (press)	To move the race car in the left direction, press the joystick in the left direction. This will shift the race car left and is proportional to the speed value.
Move race car right	Joystick right position (press)	To move the race car in the right direction, press the joystick in the right direction. This will shift the race car right and is proportional to the speed value.
Open paused screen	right Button (Switch 3)	To view current time in game or the race cars distance traveled in game press the right switch also known as switch 3 (SW3), additionally pausing the game.

CONTROLS - REPLAY OR EXIT GAME		
ACTION	BUTTON PRESS	DESCRIPTION
Replay game again	left Button (Switch 2)	If the player would like to play the game again they can do so by choosing too press the left switch also known as switch 2 (SW2).
Exits game completely	right Button (Switch 3)	If the player would like exit the game completely they can do so choosing too press the right switch also known as switch 3 (SW3).

Contents

1 Part A: Port Basic Game

1.1 Splash Screen

The player upon starting up the teensy will be greeted with the splash screen splash screen It is expected that the splash screen has the games title, student number and student name. to exit this screen and begin playing the game, the player can use either the left (SW2) or right switch (SW3).

Test plan: Upon providing power to the Teensy The splash screen boots up, the title is in a stylised font, surrounded by a box and reads 'Race to zombie mountain'. The title is displayed across three levels and features a pixel tree and house either side. Underneath the boxed title is the student name 'Matt Hutchison' and student number 'n8548625'. If any of the joystic directions are pressed (up, down, left, right, center) nothing successfully happens. The same is also the case for potentiometer 0 (pot 0) and potentiometer 1 (pot 1), no action occurs. Upon the first attempt pressing the left (SW2) successfully exits the splash screen and enters into the game. Upon the second attempt pressing the right (SW3) also successfully exits the splash screen and enters into the game.

1.1.1 SS constants values function

The following function is defined in the source file **a2_n8548625.c** spanning lines 545 - 551. The following global variables are first featured in this function, located on lines 177 - 178, 222 - 224, 227 - 254 and will be briefly defined:

- **SS_x:** Previously undefined global variable of type unsigned integer of length 8 bits (1 byte), features an overflow period of 256 bits. This variable is constant and used to set sprite splash screen at its assigned x coordinate.
- **SS_y:** Previously undefined global variable of type unsigned integer of length 8 bits (1 byte), features an overflow period of 256 bits. This variable is constant and used to set sprite splash screen at its assigned y coordinate.
- **Splash_screen:** Calls the previously defined global sprite id/Sprite, initially defined in the source file **a2_n8548625.c** line 224.
- **SS_width:** This variable remains defined as originally set out, due to the '#define' specifier, the variable has no typing. This variable stores a value for the width of the splash screen.
- **SS_height:** This variable remains defined as originally set out, due to the '#define' specifier, the variable has no typing. This variable stores a value for the height of the splash screen.
- **Splash_screen_bitmap:** Previously defined global variable of type unsigned integer of length 8 bits (1 byte), features an overflow period of 256 bits. This variable is a bitmap representing the image of the splash screen sprite. It is an an array spanning a width of ten bytes and a height of twenty seven bytes.

1.1.2 Splash screen operation function

The following function is defined in the source file **a2_n8548625.c** spanning lines 1070 - 1083. The following global variables are first featured in this function, located on lines 46, 198, 202, 207 and will be briefly defined:

- **switch_closed[ ]:** Previously undefined local global array of type unsigned integer of length 8 bits (1 byte), features an overflow period of 256 bits. This holds seven values of either one or zero, to reflect whether one or more of the switches have been pressed. Should a given switch be pressed, a value of 1 is assigned and when not presses a value of zero.

- **prevState[ ]:** Previously undefined local global array of type unsigned integer of length 8 bits (1 byte), features an overflow period of 256 bits. This holds seven values of either one or zero, storing the previous state of all switches. Should a given switch previously been pressed, a value of 1 is assigned and when not a value of zero.

- **Game_not_started:** Previously defined boolean global variable, initialised with a value of zero. As it is used it gains and stores a logical value, where true is equal to a one and false equals a value of zero. In this instance, if found to have value of zero it implies the game has still not started and the splash screen remains displayed on the LCD. Should it be equal to a value of one, the while loop is then exited, moving on to the next to start the game.

- **Start_time:** Previously undefined global variable of type unsigned integer of length 8 bits (1 byte), features an overflow period of 256 bits. when not equal to one this results in the timer three interrupt service routine (ISR) not assigning a value to cycle count. Upon gaining the required value of one, it now increases in value with each time the ISR is executed.

1.1.3 Display splash screen function

The following function is defined in the source file **a2_n8548625.c** spanning lines 842 - 846.

1.2 Dashboard

Upon entering the game from the splash screen, a dashboard must be present somewhere on screen. Where the dashboard appears is the choice of each student. It is too keep track of the current values for the race cars condition, the race cars speed and the fuel level of the race car.

Test plan: When the game starts, the dashboard information has been displayed at the top of the screen. 'C' is used too represent the condition of the race car and is initially set to a value of 100. 'S' is used to represent the current speed of the race car and is initially set to zero. 'F' is used to represent the fuel level of the car and too is set initially to a value of 100. There is a clear dividing line to separate the dashboard area from the playing area. The use of the 'conceal dashboard' function is also used to help conceal the pixels that are not turned on, from being triggered as a sprite scrolls down from the top of the screen. three steps were used to create this effect. first the sprites are called before the dashboard functions. next the 'conceal dashboard' function is called and through a for loop draws ten invisible lines. This where the y axis coordinate is increased by one on each iteration, moving between y = 0 too y = 9. To make these lines invisible the line colour is set to the background colour (BG_COLOUR).

Testing to see whether the dashboard will update based on when the condition, speed and fuel variables change, three tests were conducted. By pressing the joystick into the forward position the car starts to accelerate, this is reflected in the 'S' value for speed increasing from zero too ten, before the car begins to decelerate. As the car continues to move by accelerating, decelerating and turning the fuel levels proportionally declines. When the speed is greater, so too is the extent of the fuel level decline. Additionally when stopped next too the fuel depot, the fuel level linearly increases over a period of three seconds. Lastly when testing for the collision value, the car sprite was driven into different scenery sprite. This reduced the collision value by a value of twenty with each collision. When colliding with the fuel depot though, the collision value was set too zero and the game over screen is displayed.

1.2.1 Game constant values function

The following function is defined in the source file **a2_n8548625.c** spanning lines 557 - 574. The following global variables are first featured in this function, located on lines 153 - 166, 215 and will be briefly defined:

- **time:** Previously undefined global variable of type floating point. This holds a iterative monitoring the time value before its scaled to ensure 100 millisecond accuracy occurs. Initially set to a value 0.0.
- **minute:** Previously undefined global variable of type unsigned integer of length 8 bits (1 byte), features an overflow period of 256 bits. This holds the minute count, increasing in value every time the second count is equal to 60 seconds. Initially set to a value 0.
- **Condition:** Previously undefined global variable of type unsigned integer of length 8 bits (1 byte), features an overflow period of 256 bits. This holds the condition of the race car, should it collide with a sprite, it will decrease in value by a value of twenty.

Should it collide with a fuel depot, condition is set to zero and the game is lost. Initially set to a value 100 to represent zero damage to the race car.

- **Speed:** Previously undefined global variable of type floating point. This holds the speed value, affected by conditions where the race care is accelerating, decelerating or stationary. Initially set to a value 0.

- **Fuel:** Previously undefined global variable of type floating point. This holds the fuel level of the race car, increasing in value when stationary beside a fuel depot. Decreasing in value when speed is greater than zero (at speed equals zero, car is stationary). Initially set to a value 100 to represent having a full tank of fuel at the start of the game.

- **solve_dist:** Previously undefined global variable of type unsigned integer of length 8 bits (1 byte), features an overflow period of 256 bits. This holds the periodic unscaled distance value. Initially set to a value 0.

- **Distance:** Previously undefined global variable of type unsigned integer of length 32 bits (1 byte), features an overflow period of 4294967296 bits. This holds the actual scaled distance result. uint32_t is used, to ensure that the value for distance remains and is not returned to zero, due to large overflow period present by this variable typing. Initially set to a value 0.

- **pit_stop:** Previously undefined global variable of type unsigned integer of length 8 bits (1 byte), features an overflow period of 256 bits. This is initially set to a value of 2.

- **count:** Previously undefined global variable of type unsigned integer of length 8 bits (1 byte), features an overflow period of 256 bits. This is initially set to a value 0, the value in which the counter will begin counting from.

- **splits:** Previously undefined global variable of type unsigned integer of length 8 bits (1 byte), features an overflow period of 256 bits. This holds a value to represent the interval for each of the three splits.

- **split_center:** Previously undefined global variable of type unsigned integer of length 8 bits (1 byte), features an overflow period of 256 bits. This holds a value to represent the center point of the interval for each of the three splits.

- **split_1:** Previously undefined global variable of type unsigned integer of length 8 bits (1 byte), features an overflow period of 256 bits. This is set equal to the value of split center.

- **split_2:** Previously undefined global variable of type unsigned integer of length 8 bits (1 byte), features an overflow period of 256 bits. This holds the result of adding splits too split center.

- **split_3:** Previously undefined global variable of type unsigned integer of length 8 bits (1 byte), features an overflow period of 256 bits. This holds the value of two times splits plus the addition of split center.

1.2.2 Dashboard function

The following function is defined in the source file **a2_n8548625.c** spanning lines 824 - 836.

1.2.3 Conceal dashboard function

The following function is defined in the source file **a2_n8548625.c** spanning lines 1198 - 1204. There is a clear dividing line to separate the dashboard area from the playing area. The use of the 'conceal dashboard' function is also used to help conceal the pixels that are not turned on, from being triggered as a sprite scrolls down from the top of the screen. three steps were used to create this effect. first the sprites are called before the dashboard functions. next the 'conceal dashboard' function is called and through a for loop draws ten invisible lines. This where the y axis coordinate is increased by one on each iteration, moving between y = 0 too y = 9. To make these lines invisible the line colour is set to the background colour (BG_COLOUR).

1.3 Paused View

Paused view allows the player to view their current in game time and distance traveled by the race car. To view the current in game time and race cars distance traveled the right switch, also known as switch 3 (SW3) must be pressed. Upon opening this screen, it will pause all game operations, displaying the static results for distance and time.

Test plan: To test whether this feature works, was achieved by attempting to press the right switch, known also as switch 3 (SW3). Upon pressing this button the paused screen opened up and remained open for a time of of 1 second. Due to the button press de-bouncing implemented, this means if you want to look at it for longer than 1 second, you must press the right switch again. This Sequence for checking for the the right button press, then sets time equal to its current value. Otherwise the interrupt service routine would continue to run, causing the time also to not pause and keep increasing in value. The delay suspends the other values from updating, ensuring the game left behind when opening the pause screen, remains the same.

When open the paused screen is titled 'Paused View' and features the word 'time' followed by a static time value. Additionally the paused view contains the word 'dist' short for distance, followed by a static distance value. These values do change in the background, whilst the player continues to play the game. this can be seen by waiting for the paused screen to close, before opening it again. In that time the time and distance values will have increased, returning new static values.

1.3.1 Paused view function

The following function is defined in the source file **a2_n8548625.c** spanning lines 808 - 818. The following global variables are first featured in this function, located on line 208 and will be briefly defined:

- **time_sec:** Undefined global variable of type floating point. This constantly changes to reflect the current amount of seconds in game. Each time this reaches a certain value, the second count is reset to zero and the minute variable increases by a value of 1.

1.4 Race car, horizontal movement (non-collision)

For this section Not only were the features required by 1.4 implemented, but the more advanced horizontal movement from 2.2 was also. This part sees the creation and display of the Race car and the horizontal movement of the race car.

Test plan: When the game begins the race car is visibly displayed in the center of the road area, towards the bottom of the screen. It is constrained in the y direction, unable to visibly move up or down. It is larger than the minimum width of 3 and height of four pixels, instead spanning 9 pixels in width and a hight of nine pixels.

When the speed is shown in the dashboard to be of value zero, pressing either the Joystick in either the left or right directions, leads to no movement. With features implemented from 2.1 the player also has to hold the joystick in the down direction. Toggling between the direction the player wants to travel and the down position of the joystick. Through doing so the sprite remains static. Otherwise due to the nature of the features implemented in 2.1, where the race car will begin to accelerate if speed is smaller than one until it reaches speed of one. Whilst the speed value in the dashboard still appears as as a result of zero, this is because it is represented in integer form not as its actual floating point representation. This is due to the pixel constraints in the dashboard. By performing this toggling action the sprite will remain in its position, not moving left or right.

When the speed is greater than zero, the sprite can be shifted left or right, this is achieved through pressing the joystick in the left direction to make the car move in the left direction. Additionally by pressing the joystick in the right direction it will move in the right direction. Originally the race car was desired to move one space, but with the implementation of a more advanced horizontal movement. The horizontal movement is scaled with respect to the speed. When moving at greater speeds so too is the amount the race car can be moved left and right. Due to de-bouncing implementation for the buttons, the buttons cannot be held indefinitely, it will only register a single press. To move horizontally multiple times, the player must toggle the joystick in the desired horizontal direction, repeatedly.

The sprite has been constrained so that it will not leave the bounds of the game area. Vertical constraint is directly visible by the fact that the up, down and center joystick movements do not move the race car vertically, they just affect the speed. In addition to this the collision detection functions prevent the race car from overlapping with the scenery sprites (trees and houses), as well as the fuel depot sprite.

1.4.1 Race car move function

The following function is defined in the source file **a2_n8548625.c** spanning lines 1089 - 1101. The following global variables are first featured in this function, located on lines 144, 146, 169 - 173, 258 and will be briefly defined:

- **RC_x:** Previously undefined global variable of type unsigned integer of length 8 bits (1 byte), features an overflow period of 256 bits. This variable is used to set sprite race car

at its updated x coordinate, updating and changing each time the left or right direction of the joystick is triggered. The distance it shifts is with respect to the changes in speed . Upon collision with a sprite, it is also returned to its original x position, centered in the lower portion of the screen.

- **l_road_edge:**Previously undefined global variable of type unsigned integer of length 8 bits (1 byte), features an overflow period of 256 bits. This holds a constant value, signifying the left road edge x coordinate.

- **r_road_edge:** Previously undefined global variable of type unsigned integer of length 8 bits (1 byte), features an overflow period of 256 bits. This holds a constant value, signifying the right road edge x coordinate.

- **RC_width:** This variable remains defined as originally set out, due to the '#define' specifier, the variable has no typing. This variable stores a value for the width of the race car.

- **left_c:** Previously defined global variable of type unsigned integer of length 8 bits (1 byte), features an overflow period of 256 bits. This holds a constant value for the left coordinate of the LCD screen.

- **right_c:** Previously defined global variable of type unsigned integer of length 8 bits (1 byte), features an overflow period of 256 bits. This holds a constant value for the right coordinate of the LCD screen.

1.4.2 Tree movement function

The following function is defined in the source file **a2_n8548625.c** spanning lines 1108 - 1117. The following global variables are first featured in this function, located on lines 183, 185 - 189, 373 -374, 378 - 431 and will be briefly defined:

- **counter1:** Previously undefined global variable of type unsigned integer of length 8 bits (1 byte), features an overflow period of 256 bits. This is used as a counter increasing in value based on the current speed. Should it be equal too or exceed a value of thirty, it is reset back to zero.

- **sprite_ypos[ ]:** Previously undefined global array of type floating point. This array holds 6 specific values for each of the scenery sprites, setting each of the sprites at its assigned randomly generated y coordinate. These values will increase in value based on the value of the 'Speed' variable.

- **sprite_im_identifier[ ]:** Previously undefined local global array of type unsigned integer of length 8 bits (1 byte), features an overflow period of 256 bits. This array holds 6 specific values for each of the scenery sprites, holding a value assigned to which of the bitmaps (four tree or four house) correspond with each of the scenery sprites.

- **scenery[ ]:**

- **sprite_xpos[ ]:** Previously undefined global array of type floating point. This array holds 6 specific values for each of the scenery sprites, setting each of the sprites at its assigned randomly generated x coordinate. each of the x positions is constrained to the perimeter of either the left or right off road regions

- **T_width:** This variable remains defined as originally set out, due to the '#define' specifier, the variable has no typing. This variable stores a value for the width of the tree sprite.

- **T_height:** This variable remains defined as originally set out, due to the '#define' specifier, the variable has no typing. This variable stores a value for the height of the tree sprite.

- **Tree1_bitmap:** Previously defined global variable of type unsigned integer of length 8 bits (1 byte), features an overflow period of 256 bits. This variable is bitmap version 1 representing the image of the Tree sprite. It is an an array spanning a width of one byte and a height of ten bytes.

- **Tree2_bitmap:** Previously defined global variable of type unsigned integer of length 8 bits (1 byte), features an overflow period of 256 bits. This variable is bitmap version 2 representing the image of the Tree sprite. It is an an array spanning a width of one byte and a height of ten bytes.

- **Tree3_bitmap:** Previously defined global variable of type unsigned integer of length 8 bits (1 byte), features an overflow period of 256 bits. This variable is bitmap version 3 representing the image of the Tree sprite. It is an an array spanning a width of one byte and a height of ten bytes.

- **Tree4_bitmap:** Previously defined global variable of type unsigned integer of length 8 bits (1 byte), features an overflow period of 256 bits. This variable is bitmap version 4 representing the image of the Tree sprite. It is an an array spanning a width of one byte and a height of ten bytes.

1.4.3 House movement function

The following function is defined in the source file **a2_n8548625.c** spanning lines 1123 - 1132. The following global variables are first featured in this function, located on lines 184, 296 - 297, 302 - 370; and will be briefly defined:

- **counter2:** Previously undefined global variable of type unsigned integer of length 8 bits (1 byte), features an overflow period of 256 bits. This is used as a counter increasing in value based on the current speed. Should it be equal too or exceed a value of thirty, it is reset back to zero.

- **HL_width:** This variable remains defined as originally set out, due to the '#define' specifier, the variable has no typing. This variable stores a value for the width of the house sprite.

- **HL_height:** This variable remains defined as originally set out, due to the '#define' specifier, the variable has no typing. This variable stores a value for the height of the house sprite.

- **House2_bitmap:** Previously defined global variable of type unsigned integer of length 8 bits (1 byte), features an overflow period of 256 bits. This variable is bitmap version 2 representing the image of the House sprite. It is an an array spanning a width of two bytes and a height of thirteen bytes.

- **House3_bitmap:** Previously defined global variable of type unsigned integer of length 8 bits (1 byte), features an overflow period of 256 bits. This variable is bitmap version 3 representing the image of the House sprite. It is an an array spanning a width of two bytes and a height of thirteen bytes.

- **House4_bitmap:** Previously defined global variable of type unsigned integer of length 8 bits (1 byte), features an overflow period of 256 bits. This variable is bitmap version 4 representing the image of the House sprite. It is an an array spanning a width of two bytes and a height of thirteen bytes.

- **House5_bitmap:** Previously defined global variable of type unsigned integer of length 8 bits (1 byte), features an overflow period of 256 bits. This variable is bitmap version 5 representing the image of the House sprite. It is an an array spanning a width of two bytes and a height of thirteen bytes.

1.4.4 Counter function

The following function is defined in the source file **a2_n8548625.c** spanning lines 1138 - 1147.

1.4.5 Moving function

The following function is defined in the source file **a2_n8548625.c** spanning lines 1153 - 1167. The following global variables are first featured in this function, located on line 57 and will be briefly defined:

- **tree_or_house[ ]:** Previously undefined boolean global array, consisting of 6 future assigned values. As it is used it gains and stores a logical value, where true is equal to a one and false equals a value of zero. This array is used to define whether a sprite thats been created is a tree sprite or a house sprite. This is assigned when the sprites are first created. In addition this also helps with selecting which of the sprite movement functions to use (tree movement or house movement).

1.4.6 Sprite moving function

The following function is defined in the source file **a2_n8548625.c** spanning lines 1173 - 1180.

1.4.7 Race car operation function

The following function is defined in the source file **a2_n8548625.c** spanning lines 1284 - 1302. The following global variables are first featured in this function, located on line 260 and will be briefly defined:

- **Fuel_depot:**

1.5 Acceleration and speed

Like the horizontal movement, this too has changed based on more advanced sections. Where certain features were implemented based on the original requirements of 1.5, other requirements from other sections, such as 2.2 Accelerator and brake, 3.2 switch de-bouncing and 3.3 Timers and volatile data.

Test plan: The car was required to start the game out at a value of zero. This can be seen as soon as the game first commences, the speed in the dashboard shows a value of zero. Based on section 2.2, the car does not remain at a value of zero for long. The requirements state that when the accelerator (joystick in the up position) and brake (joystick in the center position) are not in use, that the car must increase in speed gradually from zero to one. If it is parked on the road it does this in two seconds, but if it is off road, it takes longer at a total of three seconds. This is to reflect the greater resistance in driving off road. verses on the smooth road. To test this one can park the car on the road use the press the joystick down a couple of times until the speed zeros out. then wait for the car to return to a value of one. repeat this exercise by parking the car off road and it will take noticeably longer. Due to the switch de-bouncing the joystick will have to be toggled to the down position, then released. repeat the before mentioned sequence as many times as it takes, to zero out the speed value.

Additionally the basic functionality for the speed limits has been implemented. this can be tested by accelerating the car on the road, where it will never exceed a value of ten. When moved off the road it will never exceed a value of three. The speed is represented accurately in integer form in the dashboard, where it will cycle up and down in value between zero and ten. Discussing now the rates in which the race car accelerates by, this has been changed from increments of one to now feature accelerator and braking periods of time.

To accelerate the car if on the road the player must first press the joystick into the forward position. Due to the switch de-bouncing the joystick will have to be toggled, then released. The sprites continues to gain in acceleration, rising from a value of one through too ten in 5 seconds. To test for the same acceleration of the car if on off the road the player must first shift the car to a position off road. Then press the joystick into the forward position. Due to the switch de-bouncing the joystick will have to be toggled, then released. The sprites continues to gain in acceleration, rising from a value of one through too three in 5 seconds. whilst taking the same amount of time to accelerate, when off road it increases in speed a much lower rate. This once again reflects the greater resistance when the car is off road. When the acceleration sequence is in operation, all brake or gradual joystick presses will have no effect.

Similarly, to cause the car to brake at a constant rate and time, the player can activate this by pressing the joystick when its in the center position. Due to the switch de-bouncing the joystick will have to be toggled, then released. It is set up to linearly drop from a maximum value of ten through too zero in two seconds, when triggered. This will obviously lead to a quicker deceleration when the car is moving at its max of three when off road. When the brake sequence is in operation the acceleration joystick position will not do anything, until

the car has returned to a speed of zero.

Additionally further likened to a real world scenario, when the accelerator is no longer pressed, the car will begin to decelerate. When the joystick is neither triggered in the up, down or center positions, it will gradually begin to decelerate. By parking the car on the road, triggering a burst of acceleration and then leaving it for a period of time. The car is found to decelerate from speed ten to one in a three second period. Additionally the off road case can be tested for when the car is parked off road and neither the joystick is triggered in the up, down or center positions. It will decelerate at a much quicker rate, dropping from a speed of zero too one in three seconds.

Should the speed ever fall bellow one the race car has been set too accelerate and return to a value of one. It will remain coaching along at a speed of one when no buttons are being pressed. This can be tested by toggling and releasing the joystick in the down position until the speed reaches a value of zero. Each time this is done, the speed will drop in value by one. When zeroed out and on the road, the car has been set up to return back to a speed of one over a period of two seconds. Once again due to the greater resistance when off road, the cars acceleration takes roughly three seconds to return speed back to a value of one.

1.5.1 Acceleration function

The following function is defined in the source file **a2_n8548625.c** spanning lines 1044 - 1065. The following local global variables are first featured in this function, located as function input parameters:

- **min:** Previously defined boolean global variable, initialised with a value of zero. As it is used it gains and stores a logical value, where true is equal to a one and false equals a value of zero. This variable checks the input comparison to see if it is true. It is comparing to make sure the race cars x position is still greater or equal too the constrained minimum off road coordinate. If found to have value of zero it will result nothing occurring. Should it be equal to a value of one, it will begin to run through the process for accelerating the race car.

- **max:** Previously defined boolean global variable, initialised with a value of zero. As it is used it gains and stores a logical value, where true is equal to a one and false equals a value of zero. This variable checks the input comparison to see if it is true. It is comparing to make sure the race cars x position is still less than or equal too the constrained maximum off road coordinate. If found to have a value of zero it will result nothing occurring. Should it be equal to a value of one, it will begin to run through the process for accelerating the race car.

- **speed_limit:** Defined local global variable of type floating point. This holds a of either ten or three and is defined as the upper speed limit for the race car when in one of three zones (left off road region, On road or right off road region)

1.5.2 draw road function

The following function is defined in the source file **a2_n8548625.c** spanning lines 874 - 881. The following global variables are first featured in this function, located on line 147 and will be briefly defined:

- **bottom_c:** Previously defined global variable of type unsigned integer of length 8 bits (1 byte), features an overflow period of 256 bits. This holds a constant value for the bottom coordinate of the LCD screen.

1.6 Scenery and obstacles

This section focuses upon the positioning and movement of the scenery sprites.

Test plan: When positioning the scenery sprites on screen, the bellow functions have been used to assist with the creation, positioning and movement of the sprites. It was important to ensure both the positioning of the sprites was still random but also to ensure that they were never hanging part off the screens edge. Also these sprites were never to cross the left or right boarders that define the road. Constrained to the off road sections on the left and right of the screen. the tree sprites were too appear on the right hand side and the house sprites were to appear on the left side. Each sprite is set 14 pixels apart from the others in the y direction, but the x position is randomly assigned. In addition the sprite images also alternate over time. Typically the first run of sprites will appear the same, but as they scroll off screen, are recreated and then appear back on screen, variance will begin to emerge. This is because the seed for the 'srand()' is time dependent, and when the sprites are first created, it will always be at time is equal to zero.

On multiple attempts, both through resetting the Teensy as well as selecting at end game screen to play another game, the sprites always appeared between their expected boundaries. As the speed increases and the sprites move on and off screen, the images are always changing, as well as a visible variance in the x positioning over time. As the speed increases so too does the rate in which the sprites scroll past. When the speed is slowed the sprites are also slowed. There is visibly six sprites on screen at all times, this is made up of three varying house designs and 3 varying tree designs. Both the trees and the houses feature four possible different designs. Sometimes all are different, other times, certain images will be repeated.

1.6.1 Create tree sprite function

The following function is defined in the source file **a2_n8548625.c** spanning lines 580 - 595. The following local global variable is first featured in this function:

- **randomiser:** Local global variable of type unsigned integer of length 8 bits (1 byte), features an overflow period of 256 bits. gains a new value every time the function is run. It will be assigned a random number between 1 and 4, which will intern determine which bitmap (image) the tree will be assigned.

1.6.2 Create house sprite function

The following function is defined in the source file **a2_n8548625.c** spanning lines 601 - 616.

1.6.3 sprite creation function

The following function is defined in the source file **a2_n8548625.c** spanning lines 622 - 645. The following global variables are first featured in this function, located on line 56, 181 -182 and will be briefly defined:

- **next:** Local global variable of type unsigned integer of length 8 bits (1 byte), features an overflow period of 256 bits. Is always initialised with a value of zero. Upon entering the for loop, this will then increase to a max value of two, allowing certain operations to occur, before being rest to zero again

- **first_position:** Previously undefined local global variable of type unsigned integer of length 8 bits (1 byte), features an overflow period of 256 bits. In this function it is used as a reference point, maintaining its initial value.

- **pixel_interval:** Previously undefined local global variable of type unsigned integer of length 8 bits (1 byte), features an overflow period of 256 bits. upon each loop the pixel interval increases in value by 14.

- **odd:** Previously defined boolean global variable, initialised with a value of zero. As it is used it gains and stores a logical value, where true is equal to a one and false equals a value of zero. In this instance, if found to have value of zero it will result in the sprite number being even in value, setting the sprites x position to a random position on the right side of the road. Should it be equal to a value of one, this means the sprite number is odd, and sets the sprites x position too a random position on the left side of the road.

1.6.4 sprite constant values function

The following function is defined in the source file **a2_n8548625.c** spanning lines 651 - 660.

1.6.5 Draw sprites function

The following function is defined in the source file **a2_n8548625.c** spanning lines 1186 - 1192.

1.7 Fuel depot

This section follows the implementation for the fuel depot, appearing at random intervals and on random sides. Should the race car stop next to it it has a chance to refuel.

Test plan: Similar to the scenery sprites. the fuel depot scrolls onto screen and matches the speed variable with its downward movement. When fuel is set to full the fuel depot has a zero chance of appearing. This is evident in that it never appears on screen at the start of the game. As the fuel level begins to deplete the frequency in which the fuel depot appears is much greater. It will randomly alternate between the left and right off road sections, sometimes even featuring on one side multiple times before switching. Upon stopping beside the fuel depot the fuel depot stops too.

1.7.1 Setup pump function

The following function is defined in the source file **a2_n8548625.c** spanning lines 725 - 731. The following global variables are first featured in this function, located on line 193 - 194, 276 - 278, 282 - 293 and will be briefly defined:

- **FD_x:** Previously undefined global variable of type unsigned integer of length 8 bits (1 byte), features an overflow period of 256 bits. This variable is used to set sprite fuel depot at its new random x coordinate, updating and changing each time it leaves the bottom of the screen. This determines whether it appears on the left or right side of the road too.
- **FD_y:** Previously undefined global variable of type unsigned integer of length 8 bits (1 byte), features an overflow period of 256 bits. This variable is used to set sprite fuel depot at its changing y coordinate value, this updates with respect too the speed value. As speed increases, the faster it moves towards the bottom of the screen and vice versa.
- **Fuel_depot:** Calls the previously defined global sprite id/Sprite, initially defined in the source file **a2_n8548625.c** line 276.
- **FD_width:** This variable remains defined as originally set out, due to the '#define' specifier, the variable has no typing. This variable stores a value for the width of the fuel depot.
- **FD_height:** This variable remains defined as originally set out, due to the '#define' specifier, the variable has no typing. This variable stores a value for the height of the fuel depot.
- **Fuel_depot_bitmap:** Previously defined global variable of type unsigned integer of length 8 bits (1 byte), features an overflow period of 256 bits. This variable is bitmap representing the image of the fuel depot sprite. It is an an array spanning a width of three bytes and a height of ten bytes.

1.7.2 Introduce pump function

The following function is defined in the source file **a2_n8548625.c** spanning lines 738 - 754. The following local global variables are first featured in this function:

- **threshold:** Previously undefined local global variable of type unsigned integer of length 8 bits (1 byte), features an overflow period of 256 bits. This holds the linear equation for determining the frequency in which the fuel station will appear. It is dependent on the fuel value, where when fuel is set at full (100%) the frequency in which it will appear is equal too 0. As it decreases in value from 100% too 0% it will linearly grow in value. The before mentioned equation is as follows:

$$\boldsymbol{threshold = ((-1.1111 \cdot Fuel) + 111.11)} \tag{1}$$

- **random_value:** Previously undefined local global variable of type unsigned integer of length 8 bits (1 byte), features an overflow period of 256 bits. This holds a randomly generated number between zero and one hundred thousand. This will be compared with the threshold value, upon being smaller in value This will trigger the fuel depot too reaper scrolling down from the top of the screen

1.7.3 Process pump function

The following function is defined in the source file **a2_n8548625.c** spanning lines 760 - 772.

1.8 Fuel

The fuel section makes use of both the simple fuel assignment requested in 1.8, as well as the more advanced fuel top ups from section 2.3. Where a full top up takes 3 seconds, but partial top ups now take far less time.

Test plan: To test this the player must allow the vehicle to run out of different levels of petrol, then upon seeing a petrol station it can park beside remaining at a speed of zero. To do this keep pushing and then releasing the joystick from the down position. This will compensate for the switch de-bouncing and allow the race car to remain at a speed of zero, until fuel is topped up. When testing this feature, different levels of fuel depletion do lead to different time frames for each top up. This was achieved by determining the value in which the fuel level would need to increase by on each interval. Requiring greater intervals to pass when low and less when the fuel level is almost full.

1.8.1 Fuel levels function

The following function is defined in the source file **a2_n8548625.c** spanning lines 712 - 719.

1.9 Distance traveled

This section focuses on how the distance traveled is both calculated and recorded, so that it can be viewed periodically through the use of the pause screen.

Test plan: The distance required very few lines of code to set up. initialised and set to start at a value of zero. To test for this keep pushing and then releasing the joystick from the down position. This will compensate for the switch de-bouncing and allow the race car to remain at a speed of zero. Then press the right switch also known as switch 3 (SW3) this will open the paused screen. If done correctly the distance traveled should be equal to zero, otherwise it will be very close to zero. The distance value will increase in size proportionally with that of speed. This can be tested for by continuing to trigger the acceleration button (joystick pushed in the up direction), maintaining the cars movement at around a max speed of ten. Then repeatedly check the paused screen to see if the distance is roughly increasing proportionally in intervals of ten.

1.10 Collision

two core collision functions have been created these are designed to return a logical answer off either true, equal to one or false equal to zero. They check and make a comparison between the race cars top y coordinate, bottom y coordinate, left x coordinate or the right x coordinate. Comparing these to the equivalent of the scenery sprites and/or the fuel station coordinates. should certain comparisons be greater in than one another a false result is returned. all other cases return true implying that a collision has occurred. This is returned to the test sprites function where this true or false response will determine whether the collision value remains the same, loses a value of twenty through normal sprite collision or set to zero from fuel depot collision.

Test plan: To test whether the collision functions were implemented correctly, the race car must be driven into different sprites. By repeatedly driving into scenery sprites, both houses and trees, saw a deduction of 20 from the condition of the car. Upon reaching five collisions the condition of the car was set to zero, resulting in the car being destroyed and the game over screen being displayed. If this same exercise was repeated with the fuel station, it would always set the condition to zero right away, again returning the game over screen.

1.10.1 collided sprites function

The following function is defined in the source file **a2_n8548625.c** spanning lines 1242 - 1258. The following local global variables are first featured in this function:

- **tFD:** Previously undefined local global variable of type unsigned integer of length 8 bits (1 byte), features an overflow period of 256 bits. This is set equal to the fuel depots y coordinate, representing the fuel depots top y coordinate.
- **lFD:** Previously undefined local global variable of type unsigned integer of length 8 bits (1 byte), features an overflow period of 256 bits. This is set equal to the fuel depots x coordinate, representing the fuel depots left x coordinate.
- **rFD:** Previously undefined local global variable of type unsigned integer of length 8 bits (1 byte), features an overflow period of 256 bits. This is set equal to the fuel depots x coordinate plus the width of the fuel depot, representing the fuel depots right x coordinate.
- **bFD:** Previously undefined local global variable of type unsigned integer of length 8 bits (1 byte), features an overflow period of 256 bits. This is set equal to the fuel depots y coordinate plus the height of the fuel depot, representing the fuel depots bottom y coordinate.

1.10.2 collided fuel depot function

The following function is defined in the source file **a2_n8548625.c** spanning lines 1212 - 1234. The following are primarily local global variables, with the exception of one global variable, first featured in this function, located on line 259. All will be briefly defined:

- **tRC:** Previously undefined local global variable of type unsigned integer of length 8 bits (1 byte), features an overflow period of 256 bits. This is set equal to the race cars y coordinate, representing the race cars top y coordinate.
- **lRC:** Previously undefined local global variable of type unsigned integer of length 8 bits (1 byte), features an overflow period of 256 bits. This is set equal to the race cars x coordinate, representing the race cars left x coordinate.
- **rRC:** Previously undefined local global variable of type unsigned integer of length 8 bits (1 byte), features an overflow period of 256 bits. This is set equal to the race cars x coordinate plus the width of the race car, representing the race cars right x coordinate.
- **bRC:** Previously undefined local global variable of type unsigned integer of length 8 bits (1 byte), features an overflow period of 256 bits. This is set equal to the race cars y coordinate plus the height of the race car, representing the race cars bottom y coordinate.
- **RC_height:** This variable remains defined as originally set out, due to the '#define' specifier, the variable has no typing. This variable stores a value for the width of the race car.
- **tS:** Previously undefined local global variable of type unsigned integer of length 8 bits (1 byte), features an overflow period of 256 bits. This is set equal to the the sprites (1 of 6) y coordinate, representing the current sprites top y coordinate.
- **lS:** Previously undefined local global variable of type unsigned integer of length 8 bits (1 byte), features an overflow period of 256 bits. This is set equal to the sprites (1 of 6) x coordinate, representing the sprites left x coordinate.
- **rS:** Previously undefined local global variable of type unsigned integer of length 8 bits (1 byte), features an overflow period of 256 bits. This is set equal to the sprites (1 of 6) x coordinate plus the width of the sprites car, representing the sprites right x coordinate.
- **bS:** Previously undefined local global variable of type unsigned integer of length 8 bits (1 byte), features an overflow period of 256 bits. This is set equal to the sprites (1 of 6) y coordinate plus the height of the sprites, representing the sprites bottom y coordinate.

1.10.3 Test sprites function

The following function is defined in the source file **a2_n8548625.c** spanning lines 1264 - 1278.

1.11 Game Over Dialogue

This section either returns a player wins or player lost game over screen. Player wins when either the player covers a distance of 2400 meters/pixels or when the car has remained alive for a total of four minutes. The player loses, should the car run out of fuel or the condition of the car reaches zero.

Test plan: Based on whether the player wins or loses a different screen is displayed. to show that the the player wins by reaching a certain distance, ensure the race cover covers a distance of 2400 meters/ pixels. If achieved, this will be reflected in the player wins screen being displayed and the distance value shown will be the distance required to win. If the player wins by remaining functional in game for four minutes, by not running out of fuel or remaining in reasonable condition. If achieved, this will be reflected in the player wins screen being displayed and the time value shown will be the time required to win. On the other hand if you allow the car to run out of fuel, the game will end, revealing the player lost screen. It will show the distance and time reached in that play through. For cases where multiple scenery sprite collisions are made or a single collision with the fuel depot, the game too will end. once again, it will show the distance and time reached in that play through.

The player also has two options, where if they want to play the game again, they can follow the instructions (Y: SW2). Meaning if the player wants to play another game, press the left switch also known as switch 2 (SW2). If pressed the game returns to the splash screen, which once cleared restarts anew game. If the player does not want to play again the y can follow the other instruction (N: SW3). Meaning if the player wants to exit the game completely, they have to press the right switch also known as switch 3 (SW3). upon selecting the right switch, game over appears on screen for three seconds, before the program exits completely. At this point the Teensy is left with a cleared LCD and wont respond to inputs.

1.11.1 Game over screen operation function

The following function is defined in the source file **a2_n8548625.c** spanning lines 1308 - 1317. The following global variables are first featured in this function, located on lines 48, 53 and will be briefly defined:

- **game_over:** Previously defined boolean global variable, initialised with a value of zero. As it is used it gains and stores a logical value, where true is equal to a one and false equals a value of zero. In this instance, if found to have value of zero the current game continues to run. Should it gain a value of one, the while loop exits controlling the current game, taking the player too the game over screen.
- **win:** Previously defined boolean global variable, initialised with a value of zero. As it is used it gains and stores a logical value, where true is equal to a one and false equals a value of zero. In this instance, if found to have value of zero this means the player did not meet the criteria of winning the game, displaying the player lost game over screen. Should it gain a value of one, this means the player did meet the criteria of winning the game, displaying the player won game over screen.

1.11.2 Game over screen function

The following function is defined in the source file **a2_n8548625.c** spanning lines 1323 - 1342.

1.11.3 GO function

The following function is defined in the source file **a2_n8548625.c** spanning lines 1348 - 1367. The following global variables are first featured in this function, located on lines 50 - 51 and will be briefly defined:

- **GO_Operation:** Previously defined boolean global variable, initialised with a value of zero. As it is used it gains and stores a logical value, where true is equal to a one and false equals a value of zero. In this instance, if found to have value of zero the current game over dialogue continues too be displayed on screen. Should it gain a value of one, the while loop exits allowing a new game to be set up and played.
- **exit_game:** Previously defined boolean global variable, initialised with a value of zero. As it is used it gains and stores a logical value, where true is equal to a one and false equals a value of zero. In this instance, if found to have value of zero the current game over dialogue continues too be displayed on screen. Should it gain a value of one, the while loop exits followed by the outer while loop. This causes the game to exit completely. The words 'game over' appear on screen for three seconds, followed by a blank Teensy that can no longer be interacted with.

1.11.4 Not started sequence function

The following function is defined in the source file **a2_n8548625.c** spanning lines 1373 - 1379.

1.11.5 Game over sequence function

The following function is defined in the source file **a2_n8548625.c** spanning lines 1385 - 1392.

1.11.6 Player review sequence function

The following function is defined in the source file **a2_n8548625.c** spanning lines 1398 - 1404.

1.11.7 Terminate game sequence function

The following function is defined in the source file **a2_n8548625.c** spanning lines 1410 - 1416.

1.11.8 Exit loop sequence function

The following function is defined in the source file **a2_n8548625.c** spanning lines 1422 - 1434.

1.11.9 cleanup sequence function

The following function is defined in the source file **a2_n8548625.c** spanning lines 1440 - 1448.

2 Part B: Extend Game

2.1 Accelerator and brake

Like the horizontal movement, this too has changed based on more advanced sections. Where certain features were implemented based on the original requirements of 1.5, other requirements from other sections, such as 2.2 Accelerator and brake, 3.2 switch de-bouncing and 3.3 Timers and volatile data.

Test plan: The car was required to start the game out at a value of zero. This can be seen as soon as the game first commences, the speed in the dashboard shows a value of zero. Based on section 2.2, the car does not remain at a value of zero for long. The requirements state that when the accelerator (joystick in the up position) and brake (joystick in the center position) are not in use, that the car must increase in speed gradually from zero to one. If it is parked on the road it does this in two seconds, but if it is off road, it takes longer at a total of three seconds. This is to reflect the greater resistance in driving off road. verses on the smooth road. To test this one can park the car on the road use the press the joystick down a couple of times until the speed zeros out. then wait for the car to return to a value of one. repeat this exercise by parking the car off road and it will take noticeably longer. Due to the switch de-bouncing the joystick will have to be toggled to the down position, then released. repeat the before mentioned sequence as many times as it takes, to zero out the speed value.

Additionally the basic functionality for the speed limits has been implemented. this can be tested by accelerating the car on the road, where it will never exceed a value of ten. When moved off the road it will never exceed a value of three. The speed is represented accurately in integer form in the dashboard, where it will cycle up and down in value between zero and ten. Discussing now the rates in which the race car accelerates by, this has been changed from increments of one to now feature accelerator and braking periods of time.

To accelerate the car if on the road the player must first press the joystick into the forward position. Due to the switch de-bouncing the joystick will have to be toggled, then released. The sprites continues to gain in acceleration, rising from a value of one through too ten in 5 seconds. To test for the same acceleration of the car if on off the road the player must first shift the car to a position off road. Then press the joystick into the forward position. Due to the switch de-bouncing the joystick will have to be toggled, then released. The sprites continues to gain in acceleration, rising from a value of one through too three in 5 seconds. whilst taking the same amount of time to accelerate, when off road it increases in speed a much lower rate. This once again reflects the greater resistance when the car is off road. When the acceleration sequence is in operation, all brake or gradual joystick presses will have no effect.

Similarly, to cause the car to brake at a constant rate and time, the player can activate this by pressing the joystick when its in the center position. Due to the switch de-bouncing the joystick will have to be toggled, then released. It is set up to linearly drop from a maximum value of ten through too zero in two seconds, when triggered. This will obviously lead

to a quicker deceleration when the car is moving at its max of three when off road. When the brake sequence is in operation the acceleration joystick position will not do anything, until the car has returned to a speed of zero.

Additionally further likened to a real world scenario, when the accelerator is no longer pressed, the car will begin to decelerate. When the joystick is neither triggered in the up, down or center positions, it will gradually begin to decelerate. By parking the car on the road, triggering a burst of acceleration and then leaving it for a period of time. The car is found to decelerate from speed ten to one in a three second period. Additionally the off road case can be tested for when the car is parked off road and neither the joystick is triggered in the up, down or center positions. It will decelerate at a much quicker rate, dropping from a speed of zero too one in three seconds.

Should the speed ever fall bellow one the race car has been set too accelerate and return to a value of one. It will remain coaching along at a speed of one when no buttons are being pressed. This can be tested by toggling and releasing the joystick in the down position until the speed reaches a value of zero. Each time this is done, the speed will drop in value by one. When zeroed out and on the road, the car has been set up to return back to a speed of one over a period of two seconds. Once again due to the greater resistance when off road, the cars acceleration takes roughly three seconds to return speed back to a value of one.

2.1.1 Brake or accelerate function

The following function is defined in the source file **a2_n8548625.c** spanning lines 1002 - 11011. The following global variables are first featured in this function, located on lines 213 - 214 and will be briefly defined:

- **Brake:** Initially an undefined global variable of type floating point. This holds a changing value, determined based on what function it is currently being called to be used in. This holds the amount in which the speed needs to be decreased by, increasing or decreasing based on the time interval of this breaking too occur.
- **Accelerating:** Initially an undefined global variable of type floating point. This holds a changing value, determined based on what function it is currently being called to be used in. This holds the amount in which the speed needs to be increase by, increasing or decreasing based on the time interval of this acceleration too occur.

2.2 More realistic steering

For this section Not only were the features required by 1.4 implemented, but the more advanced horizontal movement from 2.2 was also. This part sees the setup of the horizontal movement of the race car.

Test plan: When the game begins the race car is visibly displayed in the center of the road area, towards the bottom of the screen. It is constrained in the y direction, unable to visibly move up or down. It is larger than the minimum width of 3 and height of four pixels, instead spanning 9 pixels in width and a hight of nine pixels.

When the speed is shown in the dashboard to be of value zero, pressing either the Joystick in either the left or right directions, leads to no movement. With features implemented from 2.1 the player also has to hold the joystick in the down direction. Toggling between the direction the player wants to travel and the down position of the joystick. Through doing so the sprite remains static. Otherwise due to the nature of the features implemented in 2.1, where the race car will begin to accelerate if speed is smaller than one until it reaches speed of one. Whilst the speed value in the dashboard still appears as as a result of zero, this is because it is represented in integer form not as its actual floating point representation. This is due to the pixel constraints in the dashboard. By performing this toggling action the sprite will remain in its position, not moving left or right.

When the speed is greater than zero, the sprite can be shifted left or right, this is achieved through pressing the joystick in the left direction to make the car move in the left direction. Additionally by pressing the joystick in the right direction it will move in the right direction. Originally the race car was desired to move one space, but with the implementation of a more advanced horizontal movement. The horizontal movement is scaled with respect to the speed. When moving at greater speeds so too is the amount the race car can be moved left and right. Due to de-bouncing implementation for the buttons, the buttons cannot be held indefinitely, it will only register a single press. To move horizontally multiple times, the player must toggle the joystick in the desired horizontal direction, repeatedly.

The sprite has been constrained so that it will not leave the bounds of the game area. Vertical constraint is directly visible by the fact that the up, down and center joystick movements do not move the race car vertically, they just affect the speed. In addition to this the collision detection functions prevent the race car from overlapping with the scenery sprites (trees and houses), as well as the fuel depot sprite.

2.3 Fuel level increases gradually

The fuel section makes use of both the simple fuel assignment requested in 1.8, as well as the more advanced fuel top ups from section 2.3. Where a full top up takes 3 seconds, but partial top ups now take far less time.

Test plan: To test this the player must allow the vehicle to run out of different levels of petrol, then upon seeing a petrol station it can park beside remaining at a speed of zero. To do this keep pushing and then releasing the joystick from the down position. This will compensate for the switch de-bouncing and allow the race car to remain at a speed of zero, until fuel is topped up. When testing this feature, different levels of fuel depletion do lead to different time frames for each top up. This was achieved by determining the value in which the fuel level would need to increase by on each interval. Requiring greater intervals to pass when low and less when the fuel level is almost full.

2.3.1 Refueling process function

The following function is defined in the source file **a2_n8548625.c** spanning lines 778 - 801. The following global variables are first featured in this function, located on lines 55, 209 - 210 and will be briefly defined:

- **time_per_fuel:** Defined local global variable of type floating point. This holds a constant value, approximating the amount the fuel level must increase by, too ensure it is at 100% in a period of three seconds.
- **variation:** Defined local global variable of type floating point. This too holds a constant value, and returns a slightly smaller value (when compared too 'time_per_fuel') approximating the amount the fuel level must increase by.
- **initial_time:** Previously defined boolean global variable, initialised with a value of zero. As it is used it gains and stores a logical value, where true is equal to a one and false equals a value of zero. In this instance, if found to have value of zero it will result in the hold time being processed and set to the current seconds count value. Should it be equal to a value of one, it will not process anything until it returns too a zero result.
- **hold_time:** Initially an undefined global variable of type floating point. This holds a changing value, approximating the amount current second count value.
- **max_time:** Initially an undefined global variable of type floating point. This holds a changing value, storing the second count value and being reset to zero, every time the minute value is increased by one.

3 Part C: Demonstrate Mastery

3.1 De-bounce all switches

This section covers the switch de-bouncing that has been implemented. Its use case is covered greatly in sections 1.4, 1.5, 2.1, 2.2 and 3.2. de-bouncing has been implemented to set a maximum of five consecutive triggers. This constitutes one trigger in game, anything greater requires the button to be pressed again. This helps to alleviate multiple button presses, as well as instances where the player may hold down certain buttons for long periods of time. This results in smoother run times, accurate button triggers, etc. This feature has been implemented on all joystick positions (left, right, up, down, center) as well as the right and left switches (switch 2 and switch 3).

3.1.1 Setup controls function

The following function is defined in the source file **a2_n8548625.c** spanning lines 496 - 509.

3.1.2 Down switch function

The following function is defined in the source file **a2_n8548625.c** spanning lines 887 - 899.

3.1.3 Speed dependent function

The following function is defined in the source file **a2_n8548625.c** spanning lines 905 - 912.

3.1.4 Up switch function

The following function is defined in the source file **a2_n8548625.c** spanning lines 918 - 938. The following global variables are first featured in this function, located on lines 58 - 59 and will be briefly defined:

- **Decelerate_sequence:** Previously defined boolean global variable, initialised with a value of zero. As it is used it gains and stores a logical value, where true is equal to a one and false equals a value of zero. In this instance, if found to have value of zero it will cycle through the up key function operations too check for a key press. This also informs whether the race car sprite is currently decelerating, if not it can process the cars temporal forward acceleration.

- **Accelerate_sequence:**Previously defined boolean global variable, initialised with a value of zero. As it is used it gains and stores a logical value, where true is equal to a one and false equals a value of zero. In this instance, if found to have value of zero it will result in the 'Speed' variable retaining its current value. Should it be equal to a value of one, will begin too process the cars temporal forward acceleration.

3.1.5 Left switch function

The following function is defined in the source file **a2_n8548625.c** spanning lines 944 - 955. The following global variables are first featured in this function, located on lines 174, 260, 263 - 273 and will be briefly defined:

- **Race_car:** Calls the previously defined global sprite id/Sprite, initially defined in the source file **a2_n8548625.c** line 260.
- **RC_y:** Previously undefined global variable of type unsigned integer of length 8 bits (1 byte), features an overflow period of 256 bits. This variable is used to set sprite race car at a fixed y coordinate, towards the lower portion of the screen.
- **Race_car_bitmap:** Previously defined global variable of type unsigned integer of length 8 bits (1 byte), features an overflow period of 256 bits. This variable is bitmap representing the image of the race car sprite. It is an an array spanning a width of one byte and a height of nine bytes.

3.1.6 Right switch function

The following function is defined in the source file **a2_n8548625.c** spanning lines 961 - 972.

3.1.7 Center switch function

The following function is defined in the source file **a2_n8548625.c** spanning lines 978 - 996.

3.1.8 No switch pressed function

The following function is defined in the source file **a2_n8548625.c** spanning lines 1018 - 1038.

3.2 Timers and volatile data

3.2.1 Interrupt service Routine ISR0

The following function is defined in the source file **a2_n8548625.c** spanning lines 442 - 454. This interrupt service has been set up to assist with the button press de-bouncing process. It allows operations to be periodically interrupted to oversee and see if any buttons have been pressed. When pressed this result is recorded and saved. A maximum of five triggers constitutes one trigger, anything greater requires the button to be pressed again. This helps to alleviate multiple button presses, as well as instances where the player may hold down certain buttons for long periods of time. This results in smoother run times, accurate button triggers, etc

3.2.2 Interrupt service Routine ISR3

The following function is defined in the source file **a2_n8548625.c** spanning lines 464 - 469. The following Teensy interrupt variable is first featured and set in this function. This interrupt service makes use of the 16 bite timer of the Teensy. The respective timer register is assigned a prescaler assignment of 3, equating too a prescaler value of 64 for timer three. By selecting this timer, the greater overflow period can be leveraged to improve the subsequent accuracy of the output time. In addition to this, this particular prescaler ensures that the timer will overflow approximately every 0.52 seconds, allowing a more accurate second count to be equated. Whilst not exact, the output time appears far closer to measured real examples of time when compared with other 8 bit and 16 bit timer configuration.

- **cycle_count:** is used in the interrupt service routine for timer three, when the game begins, it will increase in value every time the interrupt is used.

3.2.3 Setup teensy timer function

The following function is defined in the source file **a2_n8548625.c** spanning lines 475 - 480. The following Teensy registers are first featured and set in this function:

- **TCCR0A:** The following Teensy timer register is used to set the timer to start up in normal mode, by assigning a value of zero, for timer zero.
- **TCCR0B:** The following Teensy timer register is used to set the relevant prescaler too use. For timer zero, a prescaler assignment of 4 is used, equating to0 a prescaler value of 256 for timer zero.
- **TCCR3A:** The following Teensy timer register is used to set the timer to start up in normal mode, by assigning a value of zero, for timer three.
- **TCCR3B:** The following Teensy timer register is used to set the relevant prescaler too use. For timer zero, a prescaler assignment of 3 is used, equating to0 a prescaler value of 64 for timer three.

3.2.4 Setup teensy interrupts function

The following function is defined in the source file **a2_n8548625.c** spanning lines 486 - 490. The following Teensy registers are first featured and set in this function:

- **TIMSK0:** The following Teensy timer interrupt mask register, is specified and set to equal a value of one. this results in turning on the relevant overflow interrupt for timer zero.
- **TIMSK3:** The following Teensy timer interrupt mask register, is specified and set to equal a value of one. this results in turning on the relevant overflow interrupt for timer three.

3.2.5 Setup teensy function

The following function is defined in the source file **a2_n8548625.c** spanning lines 515 - 523.

3.2.6 ISR constants function

The following function is defined in the source file **a2_n8548625.c** spanning lines 529 - 539. The following global variables are first featured in this function, located on lines 198, 199 - 200 and will be briefly defined:

- **bit_count[ ]:**Previously undefined global array of type unsigned integer of length 8 bits (1 byte), features an overflow period of 256 bits. This array gains seven specific bytes, where when a button is pressed it gains a value of one and when not pressed a zero. this is updated on a frequent basis. applying a left bit shift before updating with the relevant value.
- **history[ ]:**Previously undefined global array of type unsigned integer of length 8 bits (1 byte), features an overflow period of 256 bits. This array gains seven specific values, storing the triggered values of the buttons in the history array.

3.2.7 now function

The following function is defined in the source file **a2_n8548625.c** spanning lines 665 - 669. The following global variables are first featured in this function, located on lines 39 -40 and will be briefly defined:

- **elapsed_time:** local floating point global variable, gains and returns a new value, every time the now function is called. Where this variable is the result of the bellow formula:

$$\boldsymbol{elapsed\_time = (CC \cdot OP + T) \cdot P/F} \tag{2}$$

  where: CC = cycle count, OP = overflow period, T = clock selection too use, P = prescaler and F = frequency
- **PRESCALER:** This variable remains defined as originally set out, due to the '#define' specifier, the variable has no typing. This variable stores the prescaler value for 16 bit timer three, where overflow period 3 is in use.

- **FREQ:** This variable remains defined as originally set out, due to the '#define' specifier, the variable has no typing. This variable stores the frequency value for 16 bit timer three, where overflow period 3 is in use.

3.2.8 time logic function

The following function is defined in the source file **a2_n8548625.c** spanning lines 675 - 681.

3.2.9 time function

The following function is defined in the source file **a2_n8548625.c** spanning lines 687 - 706.

3.2.10 ISR process function

The following function is defined in the source file **a2_n8548625.c** spanning lines 852 - 868. The following are primarily local global variables (used as function inputs), with the exception of one global variable, first featured in this function, located on line 201. All will be briefly defined:

- **array_no:** Previously undefined local global variable of type unsigned integer of length 8 bits (1 byte), features an overflow period of 256 bits. Gains an assignment value in the ISR process function. This assignment value determines which of the 7 buttons are in use or too use.

- **pin:** Previously undefined local global variable of type unsigned integer of length 8 bits (1 byte), features an overflow period of 256 bits. This holds the relevant input register for the specified button to access.

- **pin_no:** Previously undefined local global variable of type unsigned integer of length 8 bits (1 byte), features an overflow period of 256 bits. This holds the relevant pin identifier, for the specific pin to use in the previously entered input register.

- **bit_mask:** Previously defined global variable of type unsigned integer of length 8 bits (1 byte), features an overflow period of 256 bits. This variable stores a constant value for the bit mask, used for button press comparisons.