CAB202 ASSIGNMENT 1

Race To Zombie Mountain (Teensy Version)

Due 3 June 2018

Davina Tan
N9741127

Tutor – Floyd Creevey

Executive Summary

This assignment required the implementation of the game 'Race To Zombie Mountain' onto the teensy. This was done by using the CAB202 Teensy, ADC and USB serial libraries. The full source code for the game has been submitted via AMS, submission reference number 38d80c43-61dd-4c7d-a449-a13e6d8e05e5.

The required functionality of porting the basic game (Part A) as set out in the specification has been implemented. Majority of the extended game (Part B) has been also been implemented. The only functionality that was not attempted from the extended game was the implementation of the curved road. Adding pixel-level collision detection and bidirectional serial communication was not implemented from demonstrate mastery (Part C). However, every other section in demonstrate mastery (Part C) was implemented.

Contents

PART A – Port Basic Game

1.1 Splash Screen

Description

The splash screen was implemented to act as a welcome screen to users who start the game. It includes the title of the game, the name of the author as well as the student number. The screen also displays an instruction on how to begin playing the game: "L or R to play". The game begins once the user presses the left or right button (SW2 or SW3).

Instructions

While playing game:

- **Joystick Up –** Accelerate
- **Joystick Down –** Brake
- **Joystick Left –** Steer Left
- **Joystick Right –** Steer Right
- **Joystick Centre –** Pause
- **Left Potentiometer DOWN –** Steer Left
- **Left Potentiometer UP –** Steer Right

While paused:

- **Right Button –** Resume
- **Joystick Up –** Send save message to putty (not implemented)

While in game over:

- **Joystick Centre –** Replay
- **Right Button -** Exit

Global Variables and Functions

Global Variables

SPLASH_SCREEN – enum type variable used for game state
new_game – bool type variable defining the game state

Functions

void process_game_state (void) – Lines 475:515
Switch statement that processes the current game state. Begins with the SPLASH_SCREEN. Calls function **reset_variables** to reset all variables at the start of the game. Contains an if statement which triggers the start of the game once either the left or right button is pressed (see debouncing section 3.2).

void update_game_state (void) – Lines 517:536
Switch statement that updates the current game state. Begins with the SPLASH_SCREEN. Calls the function **splash_screen** (see below for description).

void splash_screen (void) – Lines 542:557

Contains char variables and uses the function **draw_string** to draw the splash screen. Sets the global variable new_game to false.

Test Plan

Test Case 1 – Testing Left Button

Description & Setup	The game was loaded up onto the teensypewpew. After the splash screen displayed, the left button (SW2) was pressed.
Expected Outcome	The game starts to play.
Actual Outcome	As expected.

Test Case 2 – Testing Right Button

Description & Setup	The game was loaded up onto the teensypewpew. After the splash screen displayed, the right button (SW3) was pressed.
Expected Outcome	The game starts to play.
Actual Outcome	As expected.

1.2 Dashboard

Description

The dashboard was implemented at the top of the game to show the current condition and speed of the car. The amount of fuel remaining is also displayed on the dashboard. The dashboard is separated from the game with a border. The dashboard is also never obscured by any obstacles/scenery.

Global Variables and Functions

Global Variables

condition – int type variable set at 3 to set the initial condition of the car
current_speed – int type variable set at 0 to set the initial current speed
fuel_remaining – double type variable set at 100 to set the initial fuel remaining
dashboard_string – char type variable to draw the dashboard string

Functions

void draw_game (void) – Lines 890:911
draw_dashboard() is called.

void draw_dashboard (void) – Lines 913:919
Rounds the fuel remaining and current speed into integer variables using **round** and displays them on the screen using **sprint** and **draw_string**. The function **draw_line** is used to draw the border separating the dashboard from the game.

void update_fuel_remaining (void) – Lines 987:1015
Uses if and else if statements to update the fuel remaining reflected in the dashboard depending on the speed.

void update_car (void) – Lines 1227:1249
Calls **accelerate_car, decelerate_car, decelerate_car2, accelerate_car2** which all update the current speed reflected in the dashboard (see section 1.5 and 2.2).

void update_collision (void) – Lines 1464:1478
Updates the condition reflected in the dashboard.

Test Plan

Test Case 1 – Testing Condition and Speed updates

Description & Setup	After starting the game, the joystick was held in the up position until the car collided with an obstacle on the road.
Expected Outcome	The car's speed (S on the dashboard) increases as the joystick is held down. When the car collides with an obstacle, the car respawns on the road and the condition (H on the dashboard) decreases by one.
Actual Outcome	As expected.

Test Case 2 – Testing Fuel Remaining (Decrease over time)

Description & Setup	After starting the game, the joystick was held in the up position and the left potentiometer was used to steer the car and avoid obstacles and scenery for a few seconds.
Expected Outcome	After a few seconds, it can be observed that the fuel remaining decreases slowly over time.
Actual Outcome	As expected.

Test Case 3 – Testing Fuel Remaining (Respawn)

Description & Setup	After starting the game, the joystick was held in the up position and the left potentiometer was used to steer the car and avoid obstacles and scenery for a few seconds. After it was observed that the fuel remaining had decreased, the car was steered into an obstacle and respawned on the road.
Expected Outcome	When the car respawns on the road, the fuel remaining fills up to the maximum (100).
Actual Outcome	As expected.

1.3 Paused View

Description

A pause functionality was implemented which is activated by pressing the **joystick centre** and suspends the game until the user chooses to resume playing by pressing the **right button**. The total distance travelled by the car and elapsed time since the start of the game (accurate to the nearest 1/100th of a second) is displayed. While the game is in pause, the elapsed time pauses until the game resumes which allows the elapsed time to continue to accumulate.

Global Variables and Functions

Global Variables

overflow_counter – uint32_t type variable that stores the time using Timer 1
elapsed_time – double type variable used to store the elapsed time
time_string – char type variable to draw the time string
distance_string – char type variable to draw the distance string
PAUSE_SCREEN – enum type variable used for the game state
game_paused – bool type variable to check is the game is paused

Functions

void setup_timers (void) – Lines 260:273
Sets up the timers and turns on interrupts. (Timer 1 for the game).

ISR (TIMER1_OVF_vect) – Lines 280:284
Interrupt service routine for Timer 1. Increases while the game is not paused.

double current_time (void) – Lines 286:289
Controls the game time.

void process_game_state (void) – Lines 475:515
Switch statement that processes the current game state. PAUSE_SCREEN dims the backlight using PWM (see section 3.5). Contains an if statement which resumes the game if the right button is pressed (see debouncing section 3.2).

void update_game_state (void) – Lines 517:536
Switch statement that updates the current game state. Calls the function **pause** (see below for description).

void pause (void) – Lines 559:568
Draws the pause screen. Uses **snprintf**, **sprintf** and **draw_string** to draw the elapsed time and distance travelled.

void process (void) – Lines 574:657
Contains an if statement. Logic is: if the game is not paused, update the time and the distance.

void update_distance (void) – Lines 935:957
Updates the distance travelled.

Test Plan

Test Case 1 – Testing 10 Seconds Paused

Description & Setup	After starting the game, the centre joystick was pressed immediately to pause. After waiting approximately 10 seconds, the right button was pressed to resume the game. Immediately after the game was resumed, the centre joystick was pressed again to pause.
Expected Outcome	The time difference between the first pause screen and second pause screen has less than a second difference, even though the game has been paused for 10 seconds. This successfully tests that the game timer pauses while on the pause screen.
Actual Outcome	As expected.

Test Case 2 – Testing Distance is proportional to Time

Description & Setup	After starting the game, the up joystick was held down to accelerate the car to speed 10 while avoiding obstacles and scenery. After approximately 5 seconds, the game was paused by pressing the centre joystick. The game was then resumed by pressing the right button. After the game was resumed, the left and right joystick switches were used to avoid obstacles and scenery while keeping the speed at 1 for 5 seconds. The game was then paused again.
Expected Outcome	It can be observed after the first pause that the distance accumulated is much larger than the distance accumulated on the second pause. This tests to see if moving the car at a faster speed would have a larger distance travelled in a shorter amount of time than moving at a slower speed.
Actual Outcome	As expected.

Test Case 3 – Testing 0 Speed and Distance

Description & Setup	After starting the game, the joystick was held in the down position for approximately five seconds. The centre joystick button was then pressed to pause the game and view the distance travelled.
Expected Outcome	It can be observed that after the game is paused, the distance travelled is 0.0.
Actual Outcome	As expected.

1.4 Race Car, Horizontal Movement (non-collision)

Subject to constraints in Use ADC (section 3.1) and More Realistic Steering (section 2.3) – Test cases uses potentiometer for steering instead of left and right switches, however left and right switches can still be used for steering in game.

Description

The sprite of the race car is 6 bits wide and 8 bits high. When the current speed of the race car is zero, the car will not be able to move to the left or right. When the current speed of the car is greater than zero then the car will be able to move left or right dependent on the joystick switch the user presses. The left and right joystick switches were chosen to function as left and right movement as they deemed to be the most suitable keys. The car is also constrained so it is not able to overlap any borders as well as collision objects.

Global Variables and Functions

Global Variables

car – Sprite type variable for the car
current_speed – double type variable set at 0 to set the initial current speed
car_image – uint8_t type variable used for car bitmap

Functions

void process (void) – Lines 574:657
Calls **update_car** (see below).

int main (void) – Lines 660:689
Calls **process** if it is a new game, if the game is not over, and if the game state is currently PLAYING_GAME.

void setup (void) – Lines 789:806
Calls **setup_car** (see below).

void setup_car (void) – Lines 808:812
Initialises the car sprite using **sprite_init** at the bottom middle of the screen.

void draw_game (void) – Lines 890:911
Draws the car sprite using **sprite_draw**.

void update_car (void) – Lines 1227:1249
See details of the beginning of the function in section 1.5 and 2.2. Calls **update_steering** (see below) if the current speed is non-zero. Calls **check_offroad** (see below).

void check_offroad (void) – Lines 1251:1260
Rounds the current x coordinate of the car sprite using **round** and checks to see if the car is off the road or on the road. Limits the car's speed to 3 if the car is off the road.

void update_steering (void) – Lines 1314:1425

Rounds the current x coordinate of the car sprite using **round**. Uses an if statement to only allow horizontal movement while the car is constrained within the screen. Controls the car's left and right movement using the left and right joystick switches in functions **right_debounce** and **left_debounce** (see section 3.2 debouncing). Also controls the horizontal movement through implementation of ADC (see section 3.1).

Test Plan

Test Case 1 – Testing Car in Middle of the Road, Speed Zero

Description & Setup	After starting the game, the joystick was held down to brake and keep the speed at zero. The potentiometer was used to try and steer the car left and right while the brake was held down.
Expected Outcome	The car is not able to move left or right as the brake is held down and the speed stays at zero.
Actual Outcome	As expected.

Test Case 2 – Testing Car Next to Left Edge of Screen, Speed Non-Zero

Description & Setup	After starting the game, the joystick (up and down to accelerate and brake) and left potentiometer (scrolled down) were used to manoeuvre the car sprite to the left edge of the screen. While the speed continued to be greater than zero: 1. The left joystick switch was held to test if the car would move off the screen. 2. The left potentiometer was scrolled down (left horizontal movement) to test if the car would move off the screen.
Expected Outcome	The car is constrained successfully, and no part of the car leaves the edge of the screen. Test cases for car not overlapping scenery, obstacles, and fuel depots in section 1.10 (Collision).
Actual Outcome	As expected.

Test Case 3 – Testing Car Next to Right Edge of Screen, Speed Non-Zero

Description & Setup	After starting the game, the joystick (up and down to accelerate and brake) and left potentiometer (scrolled up) were used to manoeuvre the car sprite to the right edge of the screen. While the speed continued to be greater than zero: 1. The right joystick switch was held to test if the car would move off the screen. 2. The left potentiometer was scrolled up (right horizontal movement) to test if the car would move off the screen.
Expected Outcome	The car is constrained successfully, and no part of the car leaves the edge of the screen.
Actual Outcome	As expected.

1.5 Acceleration and Speed

Subject to constraints in Accelerator and Brake (section 2.2)

Description

Pressing the **joystick up** and **joystick down** switches will function as the accelerate and decelerate controls. These switches were chosen as they were deemed to be the most suitable for the task. At the beginning of the game, the speed of the car is zero and as the **joystick up** switch is held, the speed increases and thus moves forward. The car will not be able to move left or right or forward when the speed of the car is zero. While the car is travelling on the road, the maximum speed for the car is ten and while the car is travelling off the road, the maximum speed for the car is three. The speed is accurately reflected by the dashboard at all times.

Global Variables and Functions

Global Variables

sprite_id car – sprite type variable for the car
current_speed – double type variable set at 0 to set the initial current speed
car_image – uint8_t type variable used for car bitmap
dashboard_string – char type variable to draw the dashboard string

Functions

void update_game_state (void) – Lines 517:536
Switch statement that calls **draw_game** (see below) while the case is PLAYING_GAME.

void process (void) – Lines 574:657
Calls **update_car** (see below).

int main (void) – Lines 660:689
Calls **process** if it is a new game, if the game is not over, and if the game state is currently PLAYING_GAME. Calls **update_game_state** if it is a new game and if the game is not over.

void draw_game (void) – Lines 890:911
Calls **draw_dashboard** (see below).

void draw_dashboard (void) – Lines 913:919
Rounds the current speed using **round** and displays it in the dashboard using **sprint** and **draw_string**.

void update_car (void) – Lines 1227:1249
Calls **accelerate_car** and **decelerate_car** (see below) if the joystick is switch up or down using **up_debounce** and **down_debounce** (see section 3.2 for debouncing). Also has a constraint set so the car can only decelerate if the current speed is greater than zero.

void accelerate_car (void) – Lines 1262:1276
Uses **round** to store the current x coordinate of the car. Uses if statements to check if the car is in boundaries and if the current speed is within specifications (max 3 speed off road, max 10 speed on road). Accumulates speed depending on the car's location (see section 2.2 for details).

void decelerate_car (void) – Lines 1293:1295

Decreases the speed of the car.

Test Plan

Test Case 1 – Testing Car Stationary

Description & Setup	After starting the game, the user waited a few seconds and then the joystick down switch was held to brake and keep the speed at zero.
Expected Outcome	When the game starts, the speed of the car is initially zero. After the speed accelerates to 1 automatically (see section 2.2 Accelerator), the joystick down switch changes the speed back to zero. This tests that the car's speed stays at zero while the joystick down switch is held. (speed is observed in dashboard)
Actual Outcome	As expected.

Test Case 2 – Testing Car Accelerating from moving at an intermediate speed to going very fast on the road

Description & Setup	After starting the game, the joystick up switch was held until the speed reached 5. The left potentiometer was used to avoid obstacles and scenery while staying on the road. The joystick up switch was continued to be held in place.
Expected Outcome	The speed of the car reaches 10 and does not exceed 10. This tests that the accelerate control works. (speed is observed in dashboard)
Actual Outcome	As expected.

Test Case 3 – Testing Car Decelerating from going very fast to moving at a slow speed on the road

Description & Setup	After starting the game, the joystick up switch was held until the speed reached 10. The left potentiometer was used to avoid obstacles and scenery while staying on the road. The joystick down switch was continued to be held in place for a couple of seconds.
Expected Outcome	The speed of the car decreases from 10 to 1. This tests that the decelerate control works. (speed is observed in dashboard)
Actual Outcome	As expected.

Test Case 4 – Testing Car Decelerating from going very fast on road to going to slow off road

Description & Setup	After starting the game, the joystick up switch was held until the speed reached 10. The left potentiometer was used to avoid obstacles and scenery while staying on the road. The left potentiometer was then used to steer off the road while the joystick up switch was still held.
Expected Outcome	The speed drops from 10 to 3 once the car sprite steers off the road. This tests the maximum speed of the car off the road. (speed is observed in dashboard)
Actual Outcome	As expected.

1.6 Scenery and Obstacles

Description

The implementation of scenery and obstacles is an important factor for the game. This is because to produce the animation of the car moving forward, the scenery and obstacles scroll into view from the top and out of view down the bottom. The scenery and obstacles will not obscure the border or the dashboard and will just smoothly scroll underneath both. The speed of the objects scrolling down the window is proportional to the current speed of the car therefore creating the impression of the car moving forwards. The scenery implemented into the game will only appear off the road and involve trees, houses, and people. The obstacles implemented into the game will only appear on the road and involve roadblocks and small cars. The initial vertical position of the objects are randomised and the horizontal positions will continuingly be randomised throughout the game. There will always be at least 5 objects altogether on the screen at all times.

Global Variables and Functions

Global Variables

tree – sprite type variable for tree
tree2 – sprite type variable for tree2
house – sprite type variable for house
house2 – sprite type variable for house2
person – sprite type variable for person
person2 – sprite type variable for person2
roadBlock – sprite type variable for roadBlock
roadBlock2 – sprite type variable for roadBlock2
smallCar – sprite type variable for smallCar
smallCar2– sprite type variable for smallCar2

tree_image – uint8_t type variable used for tree and tree2 bitmap
house_image – uint8_t type variable used for house and house2 bitmap
person_image – uint8_t type variable used for person and person2 bitmap
roadBlock_image – uint8_t type variable used for roadBlock and roadBlock2 bitmap
smallCar_image – uint8_t type variable used for smallCar and smallCar2 bitmap

current_speed – int type variable set at 0 to set the initial current speed

Functions

void process (void) – Lines 574:657
update_tree, update_tree2, update_house, update_house2, update_person, update_person2, update_roadBlock, update_roadBlock2, update_smallCar, update_smallCar2, were all called.

void setup (void) – Lines 789:806
setup_tree, setup_tree2, setup_house, setup_house2, setup_person, setup_person2, setup_roadBlock, setup_roadBlock2, setup_smallCar, setup_smallCar2, were all called. The function **srand** was also used to create a random seed for each game.

void setup_tree (void) – Lines 814:818
Initialises the tree sprite using **sprite_init** at a random x coordinate using **rand_offroad_x** (see below) and a random y coordinate using **rand**.

void setup_tree2 (void) – Lines 820:824
Same description as **setup_tree** (as above).

void setup_house (void) – Lines 826:830
Same description as **setup_tree** (as above).

void setup_house2 (void) – Lines 832:836
Same description as **setup_tree** (as above).

void setup_person (void) – Lines 838:842
Same description as **setup_tree** (as above).

void setup_person2 (void) – Lines 844:848
Same description as **setup_tree** (as above).

void setup_roadBlock (void) – Lines 850:854
Initialises the roadBlock sprite using **sprite_init** at a random x coordinate using **rand_onroad_x** (see below) and a random y coordinate using **rand**.

void setup_roadBlock2 (void) – Lines 856:860
Same description as **setup_roadBlock** (as above).

void setup_smallCar (void) – Lines 862:866
Same description as **setup_ roadBlock** (as above).

void setup_smallCar2 (void) – Lines 868:872
Same description as **setup_ roadBlock** (as above).

void draw_game (void) – Lines 890:911
The tree, house, house2, person, person2, roadBlock, smallCar and smallCar2 sprites were drawn using **sprite_draw**.

double update_speed (Sprite sprite) – Lines 959:985
Uses if and else if statements to determine the sprite's movement proportional to the current speed of the car. Returns the sprite's movement.

void rand_offroad_x (void) – Lines 1021:1039
Uses **rand** to choose a random x coordinate off the road.

void rand_onroad_x (void) – Lines 1041:1047
Uses **rand** to choose a random x coordinate on the road.

void update_tree (void) – Lines 1091:1100
Uses **round** the grab the current y coordinate of the tree. If the tree scrolls down off the screen then the tree respawns at the top at a random x coordinate off the road using **rand_offroad_x**. Updates the tree's movement using **update_speed** (see above).

void update_tree2 (void) – Lines 1105:1114
Same description as **update_tree** (as above).

void update_house (void) – Lines 1105:1114
Same description as **update_tree** (as above).

void update_house2 (void) – Lines 1105:1114
Same description as **update_tree** (as above).

void update_person (void) – Lines 1105:1114
Same description as **update_tree** (as above).

void update_person2 (void) – Lines 1105:1114
Same description as **update_tree** (as above).

void update_roadBlock (void) – Lines 1105:1114
Uses **round** the grab the current y coordinate of the roadBlock. If the roadBlock scrolls down off the screen then the roadBlock respawns at the top at a random x coordinate on the road using **rand_onroad_x**. Updates the roadBlock's movement using **update_speed** (see above).

void update_roadBlock2 (void) – Lines 1105:1114
Same description as **update_roadBlock** (as above).

void update_smallCar (void) – Lines 1105:1114
Same description as **update_roadBlock** (as above).

void update_smallCar2 (void) – Lines 1105:1114
Same description as **update_roadBlock** (as above).

Test Plan

Test Case 1 – Testing Car Stationary, Scenery/Obstacles Initial Random Spawn Location

Description & Setup	After starting the game, the joystick down switch was held to brake and keep the car's speed at zero. Using the joystick (up and down switches) and the left potentiometer (horizontal movement), the car was then driven into a fuel depot to quickly end the game. In the game over screen, the joystick centre switch was pressed to restart the game. After entering the game again, the joystick down switch was held to brake and keep the car's speed at zero again.
Expected Outcome	In the first game, it can be observed that all scenery/obstacle have been spawned at random x and y locations on the screen. After restarting the game, it can also be observed that every scenery/obstacle has spawned at a new x and y location on the screen from the first game.
Actual Outcome	As expected.

Test Case 2 – Testing Car stationary to moving at an intermediate speed, then at a very fast speed, Obstacles/Scenery stationary, moving at an intermediate speed, then at a very fast speed

Description & Setup	After starting the game, the joystick down switch was held to keep the car's speed at zero. After a couple of seconds, the joystick up switch was held to accelerate the car's speed to 5. After a couple of seconds, the joystick up switch was used to increase the car's speed to 10. The left potentiometer was used to navigate the car left and right to avoid obstacles/scenery.
Expected Outcome	As the car's speed is zero, no obstacles and scenery move down the screen. When the car begins to increase its speed to 5, the scenery/obstacles begin to move down the screen at equal speeds. After the car increases its speed to 10, the scenery/obstacles begin to move at faster speeds down the screen, all moving at equal speeds.
Actual Outcome	As expected.

Test Case 3 – Testing Scenery/Obstacle scrolling in from top of the window, scrolling down to the bottom of the window, and reappearing from the top

Description & Setup	After starting the game, the joystick (up and down switches) and potentiometer (horizontal movement) was used to play the game for approximately 10-15 seconds while avoiding obstacles/scenery.
Expected Outcome	After playing for 10-15 seconds, the scenery/obstacles can be clearly seen smoothly scrolling in from the top of the screen down to the bottom of the screen. As the objects scroll down the bottom of the screen, it can be observed that after a very short amount of time, the same object that scrolled down the screen, respawns and scrolls down from the top of the screen at a different x location.
Actual Outcome	As expected.

Note: tree2 and roadBlock2 were not implemented into the final game as screen deemed to be too small to have that many number of sprites without it being significantly difficult to succeed.

1.7 Fuel Depot

Description

A fuel depot will occasionally spawn next to the road with an equal chance of appearing on either side. The fuel depots appear at random intervals but will spawn enough so the player will always have a chance to refuel. The fuel depots scroll into and out of the window as the scenery and obstacles do.

Global Variables and Functions

Global Variables

fuel – sprite type variable for fuel
fuel_remaining – double type variable initially set as 100
fuel_image – uint8_t type variable for the fuel bitmap

Functions

void process (void) – Lines 574:657
update_fuel is called (see below).

void setup (void) – Lines 789:806
setup_fuel is called (see below).

void setup_fuel (void) – Lines 874:878
Initialises the fuel sprite using **sprite_init** at the top of the screen with a random x coordinate using **rand_fuel_x** (see below).

void draw_game (void) – Lines 890:911
The fuel sprites was drawn using **sprite_draw**.

double update_speed (Sprite sprite) – Lines 959:985
Uses if and else if statements to determine the sprite's movement proportional to the current speed of the car. Returns the sprite's movement.

void rand_fuel_x (void) – Lines 1049:1062
Uses **rand** to choose a random x coordinate on either side of the road.

void rand_fuel_y (void) – Lines 1064:1085
Uses if and else if statements constrained to the fuel remaining to determine a 'random' interval at which the fuel depot spawns at. Uses the function **rand**.

void update_fuel (void) – Lines 1204:1215
Uses **round** the grab the current y coordinate of the fuel. If the fuel scrolls down off the screen then the fuel respawns at a random interval using **rand_fuel_y** (as above) at the top of the screen. It also spawns at a random x coordinate on either side of the road using **rand_fuel_x**. Updates the fuel's movement using **update_speed** (see above).

Test Plan

Test Case 1 – Testing Car Stationary, Fuel Depot Random Spawn Intervals

Description & Setup	After starting the game, the joystick (up and down switches) and left potentiometer (horizontal movement) were used to play the game until the initial fuel depot scrolled off the screen and scrolled back onto the screen from the top. The game was played until the fuel depot respawned 2-3 times.
Expected Outcome	It can be observed that as the fuel depot scrolls off the bottom of the screen, it does not scroll back onto the screen from the top straight away unlike the other obstacles/scenery.
Actual Outcome	As expected.

Test Case 2 – Testing Car stationary to moving at an intermediate speed, then at a very fast speed, Fuel depot stationary, moving at an intermediate speed, then at a very fast speed

Description & Setup	After starting the game, the joystick down switch was held to keep the car's speed at zero. After a couple of seconds, the joystick up switch was held to accelerate the car's speed to 5. After a couple of seconds, the joystick up switch was used to increase the car's speed to 10. The left potentiometer was used to navigate the car left and right to avoid obstacles/scenery.
Expected Outcome	As the car's speed is zero, the fuel depot does not move down the screen. When the car begins to increase its speed to 5, the fuel depot begins to move down the screen at equal speeds with the obstacles/scenery. After the car increases its speed to 10, the fuel depot begins to move at faster speeds down the screen, still moving at equal speeds with the obstacles/scenery.
Actual Outcome	As expected.

Test Case 3 – Testing Fuel Depot spawning on either side of the road

Description & Setup	After starting the game, the joystick (up and down switches) and left potentiometer (horizontal movement) were used to play the game until the initial fuel depot scrolled off the screen and scrolled back onto the screen from the top. The game was played until the fuel depot respawned 2-3 times.
Expected Outcome	It can be observed that the fuel depot respawns randomly on either side of the road.
Actual Outcome	As expected.

1.8 Fuel

Part of Specification in Fuel Level Increases Gradually (section 2.4)

Description

The fuel is a feature that has been implemented into the game for the user to refill the car. The car must have fuel to move and if the car runs out of fuel then the game is over. The game starts with a full tank and the fuel is consumed at a rate proportional to the speed of the car. The car's fuel tank will be refilled to maximum if the car parks next to a fuel depot for three seconds. The fuel updates will be reflected on the dashboard at all times.

Global Variables and Functions

Global Variables

car – sprite type variable for car
fuel – sprite type variable for fuel
fuel_remaining – double type variable initially set as 100
fuel_image – uint8_t type variable for the fuel bitmap
begin_refuel_amount – float type variable set at 0
refuel_stages – int type variable set at 7
stage_fuel – float type variable for set at 0
refuel_start_time – double type variable set at 0
not_refuelling – bool type variable set as true

Functions

double current_time (void) – Lines 286:289
Controls the game time.

void process (void) – Lines 574:657
Calls **update_fuel_remaining** (see below) if the game is not paused. Contains if and if else statements which control the refuelling action. If the car has 'collided' with the fuel hitbox using **fuel_collided** (see below), if the car is not refuelling, and if the car is parked, then begin the refuel using **begin_refuel** and **refuel** (see below) with **current_time** (as above) as the parameter.

void draw_dashboard (void) – Lines 913:919
Uses **round** to place the fuel remaining in an integer. Uses **sprintf** and **draw_string** to keep the dashboard updated with the fuel remaining.

void update_fuel_remaining (void) – Lines 987:1015
Uses if and else if statements to determine the amount of fuel being used (subtracted) depending on the current speed of the car. If the condition of the car is 0 then goes to the GAME_OVER state.

bool fuel_collided (void) – Lines 1554:1581
Function to test if the car has entered the collision box of the fuel depot. Uses **round**. Returns the bool variable 'collided'.

void begin_refuel (double time) – Lines 1583:1589
Returns if the fuel remaining is at maximum. Otherwise, separates the fuel remaining into sections and updates the global variable refuel_start_time.

void refuel (double time) – Lines 1591:1604

Uses current game time and the global variable refuel_start_time to refuel the fuel gradually in approximately 3 seconds.

Test Plan

Test Case 1 – Testing Car Stationary then Car Moving at a Constant Speed

Description & Setup	After starting the game, the joystick down switch was held to keep the car's speed at 0 for a few seconds. The joystick up switch was then held to increase the car's speed to 10 and kept at 10 for approximately 5 seconds. The joystick up switch was then let go and the game was played while the car's speed stayed on 1.
Expected Outcome	As the car's speed is 0, the fuel remaining stays at 100. When the car's speed increases to 10 for approximately 5 seconds, it can be observed that the fuel remaining decreases at a constant rate. When the car's speed stays at 1, it can be observed further that the fuel remaining decreases at a constant but much slower rate.
Actual Outcome	As expected.

Test Case 2 – Testing Refill at Fuel Depot

Description & Setup	After starting the game, the joystick (up and down switches) and left potentiometer (horizontal movement) were used to play the game until the car was able to park next to a fuel depot safely and the joystick down switch is held down to keep the car parked as it refuels.
Expected Outcome	It can be observed that as the car stays parked next to the fuel depot, the fuel remaining displayed on the dashboard increases gradually to maximum (100) over approximately 3 seconds.
Actual Outcome	As expected.

1.9 Distance Travelled

Description

The total amount of distance travelled during the game can be seen in the paused view. The value is initially set to zero and increases at a rate proportional to the speed of the car. The distance can never be reset to zero during a game. Once the car has travelled a very long way (100) it reaches Zombie Mountain and the player wins the game by passing through a finish line. After 100 total distance travelled, a finishing line will scroll into view and once the car is fully over the line, a victorious game over screen is displayed.

Global Variables and Functions

Global Variables

finishLine – sprite type variable for the finishLine
distance_travelled – double type variable initially set at 0
finishLine_image – char type variable for the finishLine image
distance_string – char type variable to draw the distance string
game_paused – bool type variable set to true

Functions

void pause (void) – Lines 559:568
Draws the pause screen. Uses **sprintf** and **draw_string** to draw the distance travelled.

void process (void) – Lines 574:657
Calls **update_distance** (see below) when the game is not paused. Calls **update_finishLine** and **finishLine_collision** (see below).

void setup (void) – Lines 789:806
Calls **setup_finishLine** (see below).

void setup_finishLine (void) – Lines 880:884
Initialises the finishLine sprite using **sprite_init** at the top middle of the screen.

void draw_game (void) – Lines 890:911
Draws the finishLine sprite using **sprite_draw**.

void update_distance (void) – Lines 935:957
Uses if and else if statements to determine the amount of distance travelled, proportional to the current speed of the car.

double update_speed (Sprite sprite) – Lines 959:985
Uses if and else if statements to determine the sprite's movement proportional to the current speed of the car. Returns the sprite's movement.

void update_finishLine (void) – Lines 1217:1221
If the maximum distance has been reached then scroll the finishLine sprite down the screen using **update_speed** (as above).

void finishLine_collision (void) – Lines 1540:1548

Uses **round** to take the current y coordinates of the car and finishLine sprites. Uses an if statement to see if the car sprite has crossed the finishLine sprite and triggers the GAME_OVER game state.

void do_game_over (void) – Lines 1610:1632

Displays "YOU WON" on the screen if the finishLine sprite is crossed by the car sprite (see section Game Over Dialogue 1.11 for more details).

Test Plan

Test Case 1 – Testing Car Stationary for 5 seconds

Description & Setup	After starting the game, the joystick down switch was held to keep the car's speed at 0 for 5 seconds. The centre joystick switch was then pressed to put the game in paused view.
Expected Outcome	In paused view it can be observed that even though approximately 5 seconds has passed, the distance is 0.
Actual Outcome	As expected.

Test Case 2 – Testing Car Moving at Constant Speed

Description & Setup	After starting the game, the joystick up switch was held to increase the car's speed to 10 and kept at 10 for approximately 5 seconds. The left potentiometer was used to horizontal movement to avoid obstacles/scenery. The centre joystick switch was pressed to pause after the car travelled with speed 10 for approximately 5 seconds.
Expected Outcome	As the car travels for approximately 5 seconds with very fast speed (10), it can be observed that the distance accumulated is close to 20. This successfully tests that distance accumulates at a rate proportional to the speed of the car.
Actual Outcome	As expected.

Test Case 3 – Testing Car Crossing Finish Line

Description & Setup	After starting the game, the joystick (up and down switches) and left potentiometer (horizontal movement) were used to play the game and successfully reach the finish line at 100 distance.
Expected Outcome	After the car crosses the finish line, a victorious form of the game over dialogue is displayed along with the elapsed time and distance travelled (see section Game Over Dialogue 1.11 for more details).
Actual Outcome	As expected.

1.10 Collision

Description

The game utilises bounding box collision detection. This is the most important implementation of the game because the main objective is to get to Zombie Mountain and avoid objects in the way. If the car hits any object (scenery or obstacle) then the condition is reduced to reflect the damage and updated on the dashboard. The car will instantly spawn in a safe position on the road with the current speed equal to zero and a refilled tank of fuel. If the condition of the car reaches zero then the car is destroyed and the game over screen displays. If the car collides with a fuel depot then the car is blown up and the game over screen displays.

Global Variables and Functions

Global Variables

car – sprite type variable for car
tree – sprite type variable for tree
house – sprite type variable for house
house2 – sprite type variable for house2
person – sprite type variable for person
person2 – sprite type variable for person2
roadBlock – sprite type variable for roadBlock
smallCar – sprite type variable for smallCar
smallCar2 – sprite type variable for smallCar2

car_image – char type variable for the car
tree_image – char type variable for tree
house_image – char type variable for house and house2
person_image – char type variable for person and person2
roadBlock_image – char type variable for roadBlock
smallCar_image – char type variable for smallCar and smallCar2

GAME_OVER – enum type variable for the game state
condition – int type variable set at 3 to set the initial condition of the car
game_over – bool type variable set as false

Functions

void process (void) – Lines 574:657

Calls **sprites_collided** (see below) to test if the car has collided with another sprite. If the car has collided with a scenery object or obstacle, then calls **update_collision** (see below). If the car has collided with the fuel depot then sets the game state to GAME_OVER.

void draw_dashboard (void) – Lines 913:919

The functions **sprintf**, and **draw_string** are used to display the condition of the car.

bool sprites_collided (Sprite sprite1, Sprite sprite2, int sprite1_height, int sprite1_width, int sprite2_height, int sprite2_width) – Lines 1431:1462

Uses the parameters and **round** to find the current 'border' position of each sprite. Uses if statements to see if the borders have overlapped (collided). Returns bool variable collided.

void update_collision (void) – Lines 1464:1478

Decreases the condition by 1 if the condition is greater than 0. If the condition is 0, sets the game state to GAME_OVER. Calls **car_safe_spot** (see below) to respawn the car. Sets the current speed to 0 and fuel remaining to maximum (100).

bool car_safe_spot (void) – Lines 1504:1538

Contains a while loop that continues until the car is in a safe position on the road (not colliding with any scenery/obstacle). Calls **rand_onroad_x** to spawn the car in a new x location on the road. Calls **sprites_collided** with if and else if statements to check if the car has collided with anything. Returns the bool variable safe.

void do_game_over (void) – Lines 1610:1632

Displays "GAME OVER" on the screen if the condition reaches 0 or if the car sprite has collided with the fuel sprite (see section Game Over Dialogue 1.11 for more details).

Test Plan

Test Case 1 – Testing Head-on Collision with an Obstacle (on road)

Description & Setup	After starting the game, as the car initially spawns on the road, the joystick up switch was held down until the car collided with an obstacle head-on.
Expected Outcome	When the car collides with an obstacle head-on, the car respawns at a safe location on the road, the current speed is set to 0, and the fuel remaining is set to 100. The condition of the car is decreased by 1.
Actual Outcome	As expected.

Test Case 2 – Testing Left Side Collision with a Scenery Object (off road)

Description & Setup	After starting the game, the joystick (up and down switches) and left potentiometer (scrolled up to move right) were used to move the car sprite safely off the road on the right side of the screen. The user then waited for a scenery sprite to scroll down to the right of the car sprite. The left potentiometer was scrolled up to move the car right and have a left-side collision with the scenery sprite.
Expected Outcome	When the car collides with a scenery sprite on the left side, the car respawns at a safe location on the road, the current speed is set to 0, and the fuel remaining is set to 100. The condition of the car is decreased by 1.
Actual Outcome	As expected.

Test Case 3 – Testing Right Side Collision with the Fuel Depot

Description & Setup	After starting the game, the joystick (up and down switches) and left potentiometer (horizontal movement) were used to play the game, while avoiding obstacles, until a fuel depot spawned on the left side of the road. The user then waited for the fuel depot sprite to scroll down to the left of the car sprite. The left potentiometer was scrolled down to move the car left and have a right-side collision with the fuel depot.
Expected Outcome	When the car collides with a fuel depot on the right side, the game over screen is displayed with the elapsed time and distance travelled (see section Game Over Dialogue 1.11 for more details).
Actual Outcome	As expected.

1.11 Game Over Dialogue

Description

When the game ends, a screen which will have either a victorious or game over dialogue, will display. The elapsed time and distance travelled will also be displayed on the screen. The screen will also prompt the user to allow the user to play again if they would like to and the program will wait for an affirmative or negative response via key presses. If the user chooses to play again, then a new game will commence and counters will reset. If the user chooses to decline then the program will end.

Global Variables and Functions

Global Variables

overflow_counter – double type variable set at 0 to set the initial elapsed time
condition – int type variable set at 3 to set the initial condition
current_speed – double type variable set at 0 to set the initial current speed
distance_travelled – double type variable set at 0 to set the initial distance travelled
game_over – bool type variable set as false
new_game – bool type variable set as true
GAME_OVER – enum type variable to set the game state
END_GAME – enum type variable to set the game state

Functions

void process_game_state (void) – Lines 475:515
Contains a switch statement controlling the game state. When the game state is GAME_OVER, if the user presses the joystick centre switch, **centre_debounce** (see debouncing section 3.2), then change the game state to SPLASH_SCREEN (restarts the game). If the user presses the right button, **rightSwitch_debounce** (see debouncing section 3.2) to change the game state to END_GAME (shows end game screen). When the game is restarted, and game state is SPLASH_SCREEN, calls **reset_variables** (see below).

void update_game_state (void) – Lines 517:536
Contains a switch statement that updates the game state. When the game state is GAME_OVER, call **do_game_over** (see below). When the game state is END_GAME, call **do_end_Game** (see below).

void do_game_over (void) – Lines 1610:1632
Uses an if statement to determine if the game has been won or lost depending on the condition. Displays either “YOU WON” or “GAME OVER”. Uses **snprintf**, **sprintf** and **draw_string** to display the elapsed time, distance travelled. Also prompts the user to play again or exit.

void reset_variables (void) – Lines 1634:1642
Resets global variables to initial values.

void do_end_game (void) – Lines 1644:1651
Uses **draw_string** to draw end game text “Thanks for playing!”.

Test Plan

Test Case 1 – Testing Player Wins, Restarting the Game

Description & Setup	After starting the game, the joystick (up and down switches) and left potentiometer (horizontal movement) were used to control the car and avoid obstacles/scenery/fuel depots until the distance accumulated reached 100 and the car crossed the finish line. On the game over screen the centre joystick switch was pressed. On the splash screen the left button was pressed.
Expected Outcome	After the car crosses the finish line, the game over screen displays with "YOU WON". The elapsed time and distance travelled are displayed underneath. At the bottom of the screen, "Centre to play" and "Right Switch Exit" also display. When the centre joystick switch is pressed, the splash screen displays. After pressing the left button on the splash screen, the game restarts with all variables reset.
Actual Outcome	As expected.

Test Case 2 – Testing Player Does Not Win, Ends Game

Description & Setup	After starting the game, the joystick (up and down switches) and left potentiometer (horizontal movement) were used to drive the car into a fuel depot. On the game over screen the right button was pressed.
Expected Outcome	After the car collides with the fuel depot, the game over screen displays with "GAME OVER". The elapsed time and distance travelled are displayed underneath. At the bottom of the screen, "Centre to play" and "Right Switch Exit" also display. When the right button is pressed, the end game screen displays "Thanks for playing".
Actual Outcome	As expected.

PART B – Extend Game

2.1 Curved Road (NOT IMPLEMENTED)

2.2 Accelerator and Brake

Further developed from Acceleration and Speed (section 1.5)

Description

The accelerator and brake features introduce more realistic acceleration, deceleration, and braking. Specifications include:

- Speed decreases linearly when the brake is pressed from 10 to 0 in approximately 2 seconds.
- Speed increases linearly when the accelerator is pressed
  - from 1 to 10 in approximately 5 seconds on the road
  - from 1 to 3 in approximately 5 seconds off the road
- Speed decreases linearly when both the brake and accelerator are not pressed, and the speed is greater than 1
  - from 10 to 1 in approximately 3 seconds on the road
  - from 3 to 1 in approximately 3 seconds off the road
- Speed increases linearly when both the brake and accelerator are not pressed, and the speed is less than 1
  - from 0 to 1 in approximately 2 seconds on the road
  - from 0 to 1 in approximately 3 seconds off the road

Global Variables and Functions

Global Variables

current_speed – double type variable used for the current speed of the car

Functions

void update_car (void) – Lines 1227:1249

Calls **accelerate_car** (see below) if joystick up switch is pressed using **up_debounce** (see debouncing section 3.2). Calls **decelerate_car** (see below) if joystick down is pressed using **down_debounce** (see debouncing section 3.2) and the current speed is greater than zero. Checks if current speed is greater than or less than 1 (no buttons pressed) and calls either **decelerate_car2** or **accelerate_car2** (see below).

void accelerate_car (void) – Lines 1262:1276

Uses **round** to store the current car's x position. Uses if statements to check if the car is off or on the road and increases the speed respectively.

void accelerate_car2 (void) – Lines 1278:1291

Uses **round** to store the current car's x position. Uses if statements to check if the car is off or on the road and increases the speed respectively.

void decelerate_car (void) – Lines 1293:1295

Decreases the speed.

void decelerate_car2 (void) – Lines 1297:1311

Uses **round** to store the current car's x position. Uses if statements to check if the car is off or on the road and decreases the speed respectively.

Test Plan

Test Case 1 – Testing Acceleration from 1 to 10 on road, Braking from 10 to 0

Description & Setup	After starting the game, the joystick up switch was held while the car stayed on the road, to accelerate from 1 to 10. The left potentiometer was used to steer the car and avoid obstacles. After the speed reached 10, the centre joystick switch was pressed to pause the game and view the time elapsed. The right button was pressed to resume the game and the joystick down switch was held down to brake until the speed reached 0. The centre joystick switch was then pressed again to pause the game for a second time.
Expected Outcome	It can be observed that the time elapsed on the first pause is approximately 5 seconds. On the second pause the time elapsed was approximately 2 seconds more than the last pause.
Actual Outcome	As expected.

Test Case 2 – Testing Acceleration from 1 to 3 off road, Deceleration from 3 to 1 off road

Description & Setup	After starting the game, the left potentiometer was used to steer the car off the road. The centre joystick switch was pressed to pause the game and the time elapsed was observed. The right button was pressed to resume the game and the joystick up switch was held while the car stayed off the road, to accelerate from 1 to 3. After the speed reached 3, the centre joystick switch was pressed to pause the game and the time elapsed was observed. The right button was pressed to resume the game and no buttons/switches were pressed. After the car decelerated to 1, the centre joystick switch was then pressed again to pause the game and the time elapsed was observed.
Expected Outcome	It can be observed that the time elapsed on the second pause is approximately 5 seconds more than the time elapsed on the first pause. On the third pause, it can be observed that the time elapsed is approximately 3 seconds more than the second pause.
Actual Outcome	As expected.

Test Case 3 – Testing Acceleration from 0 to 1 on road, Acceleration from 1 to 10 on road, Deceleration from 10 to 1 on road

Description & Setup	After starting the game, the centre joystick switch was pressed to pause the game and the time elapsed was observed. The right button was pressed to resume the game and no buttons/switches were pressed. When the speed increased to 1, the centre joystick switch was pressed to pause the game and the time elapsed was observed. The joystick up switch was held while the car stayed on the road, to accelerate from 1 to 10. After the speed reached 10, the centre joystick switch was pressed to pause the game and the time elapsed was observed. The right button was pressed to resume the game and no buttons were pressed. After the car decelerated to 1, the centre joystick switch was then pressed again to pause the game and the time elapsed was observed.
Expected Outcome	It can be observed that the time elapsed on the second pause is approximately 2 seconds more than the time elapsed on the first pause. On the fourth pause, it can be observed that the time elapsed is approximately 3 seconds more than the third pause.
Actual Outcome	As expected.

Test Case 4 – Testing Acceleration from 0 to 1 off road

Description & Setup	After starting the game, the left potentiometer was used to steer the car off the road. The down joystick switch was held to decelerate the car's speed to 0. The centre joystick switch was pressed to pause the game and the time elapsed was observed. The right button was pressed to resume the game and no buttons/switches were pressed. When the speed increased to 1, the centre joystick switch was pressed to pause the game and the time elapsed was observed.
Expected Outcome	It can be observed that the time elapsed on the second pause is approximately 3 seconds more than the time elapsed on the first pause.
Actual Outcome	As expected.

2.3 More Realistic Steering

Further developed from Race Car, Horizontal Movement (non-collision) (section 1.4)

Description

This feature allows more realistic steering through restrictions on the horizontal movement of the car. The speed at which the car can move horizontally is proportional to the current speed of the car. Therefore, the higher the speed, the faster the car can move horizontally.

Global Variables and Functions

Global Variables

car – Sprite type variable for the car
current_speed – double type variable set at 0 to set the initial current speed
car_image – uint8_t type variable used for car bitmap

Functions

void process (void) – Lines 574:657
Calls **update_car** (see below).

void update_car (void) – Lines 1227:1249
Calls **update_steering** (see below) if the current speed is non-zero.

void update_steering (void) – Lines 1314:1425
Rounds the current x coordinate of the car sprite using **round**. Uses an if statement to only allow horizontal movement while the car is constrained within the screen. Controls the car's horizontal movement using the left and right joystick switches in functions **right_debounce** and **left_debounce** (see section 3.2 debouncing). Uses if and else if statements to check what the current speed of the car is, then decreases (moves left) or increases (moves right) the horizontal speed respectively. Also controls the horizontal movement through implementation of ADC (see section 3.1).

Test Plan

Test Case 1 – Testing Car at a Very Fast Speed Moving Left, Decelerating to Speed 1 Moving Left

Description & Setup	After starting the game, the joystick up switch was held to accelerate the car to a very fast speed. The left potentiometer was scrolled down to move the car left. The joystick up switch was let go and the car decelerated while it continued to move left.
Expected Outcome	It can be observed that the car moves left faster while the speed is very fast, and it can also be observed that as the accelerator (joystick up) is let go, the car starts to move to the left slower as the speed decreases.
Actual Outcome	As expected.

Test Case 2 – Testing Car at Speed 1 Moving Right, Accelerating to a Very Fast Speed Moving Right

Description & Setup	After starting the game, the left potentiometer was scrolled up to move the car right. The joystick up switch was then held down to increase the car's speed while the car continued to move to the right.
Expected Outcome	It can be observed that the car moves right slowly while the speed is 1, and it can also be observed that as the accelerator (joystick up) is held, the car starts to move to the right faster as the speed increases.
Actual Outcome	As expected.

2.4 Fuel Level Increases Gradually

See section 1.8 for implementation of fuel level increasing gradually

PART C – Demonstrate Mastery

3.1 Use ADC

Description

ADC (left potentiometer) was added into the program to implement an easier option for steering. By turning the left potentiometer downwards, the car will be able to steer left. On the other hand, turning the left potentiometer upwards will allow the car to be steered right.

Global Variables and Functions

Global Variables

left_adc – int type variable set at 0 to initialise the left potentiometer

Functions

void process (void) – Lines 574:657

Sets left_adc using **adc_read(0)**.

int main (void) – Lines 660:689

Calls **adc_init** to initialise ADC.

void new_lcd_init (uint8_t contrast) – Lines 754:775

Initialises the LCD display.

void update_steering (void) – Lines 1314:1425

Uses if statements to check if the left potentiometer is scrolled down or up and updates the car's horizontal movement respectively.

Test Plan

Test Case 1 – Testing Left Potentiometer Scrolled Down (left horizontal movement)

Description & Setup	After starting the game, and the speed of the car accelerates to 1, the left potentiometer was scrolled down.
Expected Outcome	It can be observed that the car moves left.
Actual Outcome	As expected.

Test Case 1 – Testing Left Potentiometer Scrolled Up (right horizontal movement)

Description & Setup	After starting the game, and the speed of the car accelerates to 1, the left potentiometer was scrolled up.
Expected Outcome	It can be observed that the car moves right.
Actual Outcome	As expected.

3.2 De-bounce All Switches

Description

All switches were debounced in an efficient manner using the algorithm developed in Portfolio Topic 9.

Global Variables and Functions

Global Variables

centre_counter – volatile uint8_t type variable used for the joystick centre switch
rightSwitch_counter – volatile uint8_t type variable used for the right button
start_counter – volatile uint8_t type variable used for both left and right buttons
up_counter – volatile uint8_t type variable used for the joystick up switch
down_counter – volatile uint8_t type variable used for the joystick down switch
left_counter – volatile uint8_t type variable used for the joystick left switch
right_counter – volatile uint8_t type variable used for the joystick right switch

centre_closed – volatile uint8_t type variable used for the joystick centre switch
rightSwitch_ closed – volatile uint8_t type variable used for the right button
start_ closed – volatile uint8_t type variable used for both left and right buttons
up_ closed – volatile uint8_t type variable used for the joystick up switch
down_ closed – volatile uint8_t type variable used for the joystick down switch
left_ closed – volatile uint8_t type variable used for the joystick left switch
right_ closed – volatile uint8_t type variable used for the joystick right switch

Functions

void setup_timers (void) – Lines 260:273
Sets up the timers and turns on interrupts. (Timer 0 for debouncing).

ISR (TIMER0_OVF_vect) – Lines 311:388
Interrupt service routine for Timer 0. Updates the counters by left-shifting counters, using bitwise AND and OR with the masks. Uses if statements to check if the counters are equal to the bit masks. Determines if the buttons/switches are open or closed.

bool centre_debounce (void) – Lines 394:403
Determines if the switch is debounced.

bool rightSwitch_debounce (void) – Lines 405:414
Determines if the switch is debounced.

bool start_debounce (void) – Lines 416:425
Determines if the switch is debounced.

bool up_debounce (void) – Lines 427:436
Determines if the switch is debounced.

bool down_debounce (void) – Lines 438:447
Determines if the switch is debounced.

bool left_debounce (void) – Lines 449:458
Determines if the switch is debounced.

bool right_debounce (void) – Lines 460:469
Determines if the switch is debounced.

Test Plan

Test Case 1 – Testing Centre Switch

- *See Paused View section 1.3 to test the centre switch to pause*

Test Case 2– Testing Right Button

- *See Paused View section 1.3 to test the right button to resume*

Test Case 3– Testing Start (Left or Right) Button

- *See Splash Screen section 1.1 to test either the left or right button starts the game*

Test Case 4– Testing Up Switch

- *See Acceleration and Speed section 1.5 and Accelerator and Brake section 2.2 to test the up switch accelerates.*

Test Case 5– Testing Down Switch

- *See Acceleration and Speed section 1.5 and Accelerator and Brake section 2.2 to test the down switch decelerates.*

Test Case 6– Testing Left Switch

- *See Race Car, Horizontal Movement (non-collision) section 1.4 to test left movement*

Test Case 7– Testing Right Switch

- *See Race Car, Horizontal Movement (non-collision) section 1.4 to test right movement*

3.3 Direct Screen Update

Description

Direct control of the LCD display was used to bypass the CAB202_teensy screen buffer for implementation of a flashing effect when the car collides with an obstacle/scenery object.

Global Variables and Functions

Global Variables

crash – int type variable used for the crash

Functions

void process (void) – Lines 574:657

Calls **crash_collision** (see below).

void update_collision (void) – Lines 1464:1478

Sets crash as 25.

void crash_collision (void) – Lines 1481:1502

Decreases crash if crash is greater than 0. Bypasses the screen by directly writing uint8_t variable byte_to_write to the screen. Uses two for loops to visit each column and row of the output bitmap.

Test Plan

Test Case 1 – Testing Screen Flashes when Car Collides with a Sprite

Description & Setup	After starting the game, the joystick up switch was held until the car collided with an obstacle on the road.
Expected Outcome	When the car collides with an obstacle the screen flashes approximately 3 times and then the game resumes as normal.
Actual Outcome	As expected.

3.4 Timers and Volatile Data

Description

Multiple timers and interrupts were used in the program to implement elapsed time (section 1.3), debouncing (section 3.2) as well as PWM (section 3.5).

Global Variables and Functions

Global Variables

overflow_counter – uint32_t type variable that stores the time using Timer 1
elapsed_time – double type variable used to store the elapsed time

centre_counter – volatile uint8_t type variable used for the joystick centre switch
rightSwitch_counter – volatile uint8_t type variable used for the right button
start_counter – volatile uint8_t type variable used for both left and right buttons
up_counter – volatile uint8_t type variable used for the joystick up switch
down_counter – volatile uint8_t type variable used for the joystick down switch
left_counter – volatile uint8_t type variable used for the joystick left switch
right_counter – volatile uint8_t type variable used for the joystick right switch

centre_closed – volatile uint8_t type variable used for the joystick centre switch
rightSwitch_ closed – volatile uint8_t type variable used for the right button
start_ closed – volatile uint8_t type variable used for both left and right buttons
up_ closed – volatile uint8_t type variable used for the joystick up switch
down_ closed – volatile uint8_t type variable used for the joystick down switch
left_ closed – volatile uint8_t type variable used for the joystick left switch
right_ closed – volatile uint8_t type variable used for the joystick right switch

Functions

void setup_timers (void) – Lines 260:273
Sets up the timers and turns on interrupts. (Timer 1 for the game, Timer 0 for debouncing).

ISR (TIMER1_OVF_vect) – Lines 280:284
Interrupt service routine for Timer 1. Increases while the game is not paused.

double current_time (void) – Lines 286:289
Controls the game time.

ISR (TIMER0_OVF_vect) – Lines 311:388
Interrupt service routine for Timer 0 (see details debouncing section 3.2).

void setup_PWM (void) – Lines 738:747
Uses Timer 4 to drive the Output Compare Register 4A, on pin C7 (backlight).

void set_duty_cycle (int duty_cycle)
Sets bits of Output Compare Register 4A.

Test Plan

See Paused View section 1.3 for Timer 1 test plan.

See Debouncing section 3.2 for Timer 0 test plan.

See PWM section 3.5 for Timer 4 test plan.

3.5 PWM (Pulse Width Modulation)

Description

PWM is implemented in an appropriate manner to implement special effects regarding the backlight. If the car collides with an obstacle/scenery object then the backlight dims while the screen flashes (see direct screen update section 3.3). The backlight also dims while the game is in paused view.

Global Variables and Functions

Global Variables

None

Functions

void process_game_state (void) – Lines 475:515

Switch statement that processes the current game state. If the game state is PLAYING_GAME then calls **set_duty_cycle** (see below) with 0 as the parameter. If the game state is PAUSE_SCREEN then calls **set_duty_cycle** with 700 as the parameter.

int main (void) – Lines 660:689

Calls **setup_PWM** (see below).

void setup_PWM (void) – Lines 738:747

Uses Timer 4 to drive the Output Compare Register 4A, on pin C7 (backlight).

void set_duty_cycle (int duty_cycle)

Sets bits of Output Compare Register 4A.

void crash_collision (void) – Lines 1481:1502

Calls **set_duty_cycle** (as above) with 750 as the parameter.

Test Plan

Test Case 1 – Testing Backlight Dims in Paused View

Description & Setup	After starting the game, the joystick centre switch was pressed to enter paused view. The right button was then pressed to resume the game.
Expected Outcome	In paused view it can be observed that the backlight dims. When the game is resumed, it can be observed that the backlight brightens to maximum.
Actual Outcome	As expected.

Test Case 1 – Testing Backlight Dims during Flash when Car Collides with a Sprite

Description & Setup	After starting the game, the joystick up switch was held until the car collided with an obstacle on the road.
Expected Outcome	When the car collides with an obstacle the backlight dims as the screen flashes and when the game resumes the backlight is set back to its maximum brightness.
Actual Outcome	As expected.

3.6 Pixel-level Collision Detection (NOT IMPLEMENTED)

3.7 Bidirectional Serial Communication and Access to File System (NOT IMPLEMENTED)