Race To Zombie Mountain Game V2 Report

Supported by ZDK, programmed in C language, compiled in Cygwin and ported into TeensyPewPew

Kevin Duong – n9934731

DUE DATE – 03 JUNE 2018

Executive Summary

This report was made to analyse the game ‘Race to Zombie Mountain’, which is now ported into the ‘TeensyPewPew’ microcontroller. Key findings involve testing the portable Teensy to demonstrate and explain the features for each case, using switches, LEDs and a joystick on the TeensyPewPew to create a dynamic experience for the player. Each feature is created with the use of functions and support from ZDK and cab202_teensy, and are listed so to indicate the functions used for the specific feature used in the game to operate. After each case, the findings conclude the Teensy’s hardware capabilities of performing the basic game ports on the PC. It also takes into account the ADC, switch de-bouncing to perform particular tasks, timers and volatile data and more. The game works through understanding the concepts of each controls and functions, with the help of reading each case to realise what each feature does, and what each of the function and variables is used for that feature as well.

Table of Contents

Introduction

Previously, a report was made to analyse the game ‘Race To Zombie Mountain’, which could run on any PC with the addition of using the Cygwin console to compile and run the source code. There is a new method of playing the game, which involves playing it on a microcontroller driven gamepad called ‘TeensyPewPew’. The microcontroller consists of many features that makes the game more interactive and enjoyable by using switches, potentiometer, and LEDs rather than a keyboard. This report documents such features from TeensyPewPew and functions to make the game work, using both kits of the ZDK and cab202_teensy files as the frameworks for the game.

Part A – Port Basic Game

Splash Screen

The player is greeted with a splash screen, welcoming them to the game. It displays the title of the game 'WELCOME TO RTZM', following the name of the student, their ID number and an instruction to start the game. The game will start once the player presses the middle joystick.

Global variables / Functions used:

- bool menu = true; (line 25)
  - Enables the menu to launch first in the game before playing.
- void splash_screen(); (line 667)
  - The menu of the game, displaying string of text.

<table>
<tr><th>Test Case</th><th>Test description and setup</th><th>Expected outcome</th><th>Actual outcome</th></tr>
<tr><td>SW1</td><td rowspan="5">The game must be running and should start up with a splash screen. A button should allow the user to proceed to the game.</td><td>The game should begin and exit the splash screen by pressing the centre of the joystick.</td><td>The game now starts, and the splash screen had ended.</td></tr>
<tr><td>SW2</td><td rowspan="4">Nothing should be triggered by pressing this button.</td><td rowspan="4">As stated before, the game does not have any effect.</td></tr>
<tr><td>SW3</td></tr>
<tr><td>POT0</td></tr>
<tr><td>POT1</td></tr>
</table>

Dashboard

A heads up display of player's status is located on the top of the LCD, separated from the border. The H stands for health, which is the condition of the player and has a value of 0-100. The S stands for speed, the current speed of the race car and has a value of 0-10. The last letter F stands for fuel, which is the amount of fuel remaining and contains the value from 0-100.

Global variables / Functions used:

- int health = 100; (line 29)
  - Health value for player.
- int speed = 0; (line 30)
  - Speed value for player.
- float fuel = 0; (line 31)
  - Fuel value for player.
- void player_hud(); (line 694)
  - The dashboard that displays the player's status in game.

Test Case	Test description and setup	Expected outcome	Actual outcome
Dashboard	When starting the game, the player can proceed to view the dashboard by pressing the centre joystick.	The game should include a dashboard to always reflect the player's status; the player is introduced to the dashboard on top of the LCD.	As a result, the dashboard is displaying correctly, showing three conditions of the player.
Pause game	Press the SW3 button to pause the game while playing the game.	By staying in the pause menu, the dashboard should still be available once the player resumes playing.	

Paused View

Players can pause the game during the session. By pressing SW3, the game is suspended, and a new screen will prompt the title 'GAME PAUSED', with the 'DISTANCE' and 'TIME' labels just below the title. The pause menu records the distance travelled and time elapsed during gameplay, and will hold onto the statistics until the game resumes, by pressing SW2 to exit the pause screen.

Global variables / Functions used:

- bool pause = false; (line 25)
  - Boolean statement to determine if pause mode is true or false.
- float distance = 0; (line 32)
  - Distance value of the game travelled by player.
- int seconds = 0; (line 36)
  - Time value in seconds.
- int delay_count = 0; (line 37)
  - Counter to increase seconds integer.
- void pausemenu(); (line 739)

  - The pause screen of the game, displaying player statistics of time played and distance travelled.
- void timer(); (line 727)
  - Elapsed time of in game.
- void distance travelled();
  - Distance recorded in game.

Test Case	Test description and setup	Expected outcome	Actual outcome
Elapsed Time	Record the amount of time pass in the game and return to the pause menu to check if working.	We only want the time to elapse **only** in the game and not in the pause menu.	The time is static and does not change. It only changes when the player resumes the game for a sufficient time to make the time change.
Distance travelled	Progress in the game for an amount of time to accumulate distance travelled, then return to the pause menu to validate. Also test this by being stationary.	The distance should only be recorded in game and not in the menu. We also want to make sure that distance isn't covered when player is stationary.	Distance is covered as we expected when the player drives. It also does not increase value when we are not moving.

Race car, horizontal movement (non-collision)

The player commandeers a vehicle, which is located at the bottom middle of the LCD. Given the conditions of movement, if the speed is 0, then the player cannot move laterally at any side of the map and can do so if speed is accumulated to be greater than 0. Players can move their car by toggling the left and right joysticks that results in moving left and right respectively. They can only move within the boundaries of the border and are not able to overlap any objects along the way.

Global variables / Functions used:

- Sprite player (line 52);
  - Sprite ID for player.
- static uint8_t player_image[]; (line 53)
  - bitmap image of the player model (car).
- void setup_player(void); (line 64)
  - Setup bits that will be used for controls, established player sprite.
- void controls(void); (line 772)
  - Movement controls set bits to PINB 1 and PIND 0 (left and right).
  - Border boundaries set.

Test Case	Test description and setup	Expected outcome	Actual outcome
Car in middle of road (Speed zero)	Move the vehicle to the middle of the road by pushing the joystick to the left and right.	The car should already be in the middle of the lower road, even when not moving.	The player is already in the middle of the road when the game starts. This is shown in line 77 of the source code which initiates the sprite to spawn in such position.
Car next to left border (Speed zero)	Move the vehicle to the left border of the road by pushing the joystick to the left.	The car should not be moving as its speed is zero, hence it will not move at all.	The car cannot move when its speed is less than 1. Line 783 indicates that if speed is 0, then no lateral motion will be enforced.
Car next to right border (Speed zero)	Move the vehicle to the right border of the road by pushing the joystick to the right.		
Car in middle of road (Non- speed zero)	Move the vehicle in the middle of the road by pushing the joystick to the left and right.	The player should be able to move to the middle of the road if they use their joystick to control the car,	The player initially is in the middle of the road, but can move horizontally elsewhere and return back to the middle of the road.
Car next to left border (Non-speed zero)	Move the vehicle to the left border of the road by pushing the joystick to the left.	The player should be able to move to the left side of the map should they push their left joystick towards it.	The player can move next to the left/right border. It cannot however pass through any border shown in an if statement in line 794 and 800.
Car next to right border (Non-Speed zero)	Move the vehicle to the right border of the road by pushing the joystick to the right.	The player should be able to move to the left side of the map should they push their left joystick towards it.	

Acceleration and speed

To go faster in the game, the player can use their joystick to either increase or decrease their speed, which is respectively followed by the up and down joystick. With the speed initially 0, the player cannot decrease their speed any further, as the speed is in the range of 0 – 10, going slow to fast for that matter. When the player leaves the road, their speed drops down to 3 and cannot exceed the speed until they enter the road again. Players can see their speed which is reflected on the dashboard.

Global variables / Functions used:

- int speed = 0; (line 30)
  - Speed value for player.
- void player_hud() (line 693)
  - Speed is reflected onto the dashboard.
- void controls(void); (line 772)
  - Speed controls set bits to PIND 1 and PINB 7 (increase and decrease speed)

Test Case	Test description and setup	Expected outcome	Actual outcome
Car stationary (Accelerate)	Do not move the car.	We should see the car still being stationary and not maintain speed.	The car is stationary and remains to be so. It will accelerate when the player increases its speed.
Car moving intermediate speed (Accelerate)	Set the car speed to 5 by pushing the joystick up.	The car should be at an intermediate speed.	As stated before, the car can reach an intermediate speed indicated by the S:5 in the dashboard.
Car going very fast (Accelerate)	Set the car speed to 10 by pushing the joystick up.	The car should going very fast.	As stated before, the car can go very fast indicated by the S:10 in the dashboard.
Car stationary (Decelerate)	Do not move the car.	We should see the car still being stationary and not maintain speed.	The car is stationary and remains to be so. It cannot decelerate any further as it is at speed 0.
Car moving intermediate speed (Decelerate)	Decelerate the car to a speed of 5 by pushing the joystick down.	The car should not be able reach intermediate speeds when decelerating the car.	As so, the car can only decelerate to the intermediate speed if it is above 5.
Car going very fast (Decelerate)	Decelerate the car to a speed of 10 by pushing the joystick down.		As so, the car can only decelerate to the intermediate speed if it is above 10.

Scenery and obstacles

The game is filled with obstacles and scenery that start approaching to the player when they start to progress in the game. These objects spawn at the top of the border and make their way through the bottom of the border when the player is moving. They spawn randomly across the map; the scenery spawn outside of the road, while obstacles spawn on the road. The speed of the objects is proportional to the speed of the player, and when the player stops, so do the objects themselves.

Global variables / Functions used:

- int seed = 1; (line 34)
  - Number value to create a spawn choice.
- int spriteCount = 5; (line 82)
  - Number of sprites per spawn.
- uint8_t zombie_image[]; (line 85)
  - bitmap image of zombie.
- uint8_t sign_image[]; (line 94)
  - bitmap image of construction sign.
- uint8_t blockade_image[]; (line 103)
  - bitmap image of water-filled barrier.
- Sprite obstacle:2:3, (line 110, 142, 174)
  - Obstacle sprite to group sprites on road.
- void create_obstacle(); 2; 3, (line 111, 143, 175)
  - Setup obstacle size, positioning on the left side of map.
- uint8_t tree_image[]; (line 207)
  - bitmap image of tree.
- uint8_t house_image[]; (line 218)
  - bitmap image of house.
- Sprite sceneryLeft:2,3 (line 230, 257, 284)
  - Scenery sprite to group sprits outside of road at left side.
- void create_sceneryLeft();2,3 (line 231, 258, 285)
  - Setup scenery size, positioning.
- Sprite sceneryRight:2,3 (line 311, 338, 365)
  - Scenery sprite to group sprits outside of road at right side.
- void create_sceneryRight();2,3 (line 312, 339, 366)
  - Setup scenery size, positioning on the right side of map.
- void create_sprites(); (line 468)
  - Initialise sprites in the beginning of game.
- void spawn_sprites(); (line 505)
  - Draw sprites to be visible in game.
  - Set sprite speed based on player speed.

Test Case	Test description and setup	Expected outcome	Actual outcome
Scenery/obstacle scrolling in at top of window			
Car stationary	Do not move the car.	The scenery and obstacles should also be stationary at the top of the window.	No objects or obstacles move on their own. They are still at their respective positions.
Car moving intermediate speed	Accelerate the car to a speed of 5 by pushing the joystick up.	The scenery/obstacles should now scroll down even faster at a speed proportional to the player and	All objects move at a speed proportional to the player's intermediate speed.
Car going very fast	Accelerate the car to a speed of 10 by		All objects move at a speed proportional

	pushing the joystick up.	start approaching down to the bottom.	to the player's very fast speed.
Scenery/obstacle in middle of window			
Car stationary	Do not move the car.	The scenery and obstacles should also be stationary at the middle of the window.	No objects or obstacles move on their own. They are still at their respective positions.
Car moving intermediate speed	Accelerate the car to a speed of 5 by pushing the joystick up.	The scenery/obstacles should now scroll down even faster at a speed proportional to the player and start approaching down to the bottom very quickly.	All objects move at a speed proportional to the player's intermediate speed.
Car going very fast	Accelerate the car to a speed of 10 by pushing the joystick up.		All objects move at a speed proportional to the player's very fast speed.
Scenery/obstacle scrolling out at bottom of window			
Car stationary	Do not move the car.	The scenery and obstacles should also be stationary at the bottom of the window.	No objects or obstacles move on their own. They are still at their respective positions.
Car moving intermediate speed	Accelerate the car to a speed of 5 by pushing the joystick up.	The scenery/obstacles should now scroll down even faster at a speed proportional to the player and be hidden away from the game.	All objects move at a speed proportional to the player's intermediate speed.
Car going very fast	Accelerate the car to a speed of 10 by pushing the joystick up.		All objects move at a speed proportional to the player's very fast speed.

Fuel depot

As the player progress in the game, they will gradually lose their fuel tank and will need to refuel their car, by driving near a fuel depot in either any side of the road. Indicated by the letter 'F' for fuel, these stations can appear randomly just like objects and obstacles and their behaviour is also like them as well.

Global variables / Functions used:

- uint8_t fuelDepot_img[]; (line 393)
  - bitmap image of fuel depot.
- Sprite fuelDepotLeft (line 405)
  - Sprite ID for fuelDepot spawning on left side of road.
- Sprite fuelDepotRight (line 406)
  - Sprite ID for fuelDepot spawning on the right side of road.

- void create_fuelDepotLeft/Right(); (line 408, 422)
  - Setup size, positioning for two fuel depots.
- void create_sprites(); (line 468)
  - Initialise fuel sprites in the beginning of game.
- void spawn_sprites(); (line 505)
  - Draw fuel depot to be visible in game.
  - Set fuel depot speed based on player speed.

Include in test plan:

- Test cases for meaningful combinations of (Car stationary; Car moving intermediate speed; Car going very fast) vs. (Depot at top of window; Depot in middle of window; Depot at bottom of window).

<table>
<tr><th>Test Case</th><th>Test description and setup</th><th>Expected outcome</th><th>Actual outcome</th></tr>
<tr><td colspan="4">Fuel Depot scrolling in at top of window</td></tr>
<tr><td>Car stationary</td><td>Do not move the car.</td><td>The fuel depot should also be stationary at the top of the window.</td><td>The fuel depot does not move on its own. It is still at its respective position.</td></tr>
<tr><td>Car moving intermediate speed</td><td>Accelerate the car to a speed of 5 by pushing the joystick up.</td><td rowspan="2">The fuel depot should now scroll down even faster at a speed proportional to the player and start approaching down to the bottom.</td><td>The fuel depot moves at a speed proportional to the player’s intermediate speed.</td></tr>
<tr><td>Car going very fast</td><td>Accelerate the car to a speed of 10 by pushing the joystick up.</td><td>The fuel depot moves at a speed proportional to the player’s very fast speed.</td></tr>
<tr><td colspan="4">Fuel Depot in middle of window</td></tr>
<tr><td>Car stationary</td><td>Do not move the car.</td><td>The fuel depot should also be stationary at the middle of the window.</td><td>The fuel depot does not move on its own. It is still at its respective position.</td></tr>
<tr><td>Car moving intermediate speed</td><td>Accelerate the car to a speed of 5 by pushing the joystick up.</td><td rowspan="2">The fuel depot should now scroll down even faster at a speed proportional to the player and start approaching down to the bottom very quickly.</td><td>The fuel depot moves at a speed proportional to the player’s intermediate speed.</td></tr>
<tr><td>Car going very fast</td><td>Accelerate the car to a speed of 10 by pushing the joystick up.</td><td>The fuel depot moves at a speed proportional to the player’s very fast speed.</td></tr>
<tr><td colspan="4">Fuel Depot scrolling out at bottom of window</td></tr>
</table>

Car stationary	Do not move the car.	The fuel depot should be stationary at the bottom of the window.	The fuel depot does move on its own. It is still at its respective position.
Car moving intermediate speed	Accelerate the car to a speed of 5 by pushing the joystick up.	The fuel depot should now scroll down even faster at a speed proportional to the player and be hidden away from the game.	The fuel depot moves at a speed proportional to the player's intermediate speed.
Car going very fast	Accelerate the car to a speed of 10 by pushing the joystick up.		The fuel depot moves at a speed proportional to the player's very fast speed.

Fuel

The player starts with a full tank which can be consumed at a rate proportional to the speed of the car, indicated by the fuel value on the dashboard. Refuelling requires the player to park their car next to any fuel stations on the edge of the road, which will take 3 seconds to restore the fuel tank to its maximum. If the player loses all fuel and is not next a fuel station, then the game will end.

Global variables / Functions used:

- float fuel = 100; (line 31)
  - Fuel value for player.
- int refuel_timer = 0; (line 39)
  - Timer to count refuel.
- int refuel_count = 0; (line 40)
  - Timer to count refuel.
- void player_hud() (line 693)
  - Fuel value is reflected onto the dashboard.
- Bool refuel_left (Sprite player, Sprite fuelDepotLeft); (line 612)
  - Collision Boolean statement for player and left fuel depot
- Bool refuel_right (Sprite player, Sprite fuelDepotRight); (line 631)
  - Collision Boolean statement for player and right fuel depot
- void fuel_consumption(); (line 716)
  - Fuel rate decrease based on speed.
- void refuel_function(); (line 849)
  - Function to be able to make player refuel their car at any fuel depot.

Test Case	Test description and setup	Expected outcome	Actual outcome
Car stationary	Do not move the car.	The player should still have their fuel	The fuel value has not decreased in any way while

		remain the same when not moving.	stationary. This reflected on the dashboard.
Car moving at constant speed	Move the car at a reasonable speed.	The fuel value should decrease in the duration of the car moving.	The fuel does decrease when speed is greater than 0. The decrease rate depends on the player's speed; the higher the speed, the quicker the fuel decreases.
Refill at a fuel depot	Drive up to a fuel depot, being adjacent to one while on the road.	The player should have their car successfully fuelled up at the duration of 3 seconds.	The car is adjacent to the fuel depot and cannot move for 3 seconds until it is fuelled up. The dashboard indicates it is full and the car can now move freely.

Distance Travelled

Travelling in the game will record how far the player has gone from the beginning of the game. They will be able to see this in the pause screen, which displays the distance follow by the value of the distance in metres. The distance statistic will reset when the player restarts the game. Speed will make a difference in the distance travelled, as proportionate takes effect. To finish the game, the player must travel at a certain distance of 5km to reach their destination, where a finishing point will spawn and be available to reach - this will finish the game.

Global variables / Functions used:

- float distance = 0; (line 32)
  - Initial distance value of player.
- void distance_traveled(); (line 707)
  - Distance accumulates when player progresses.
- void pausemenu(); (line 739)
  - The distance value can be viewed on the pause menu.
- uint8_t ZM_image[]; (line 435)
  - bitmap image of the proposed zombie mountain finishing line.
- Sprite finishline1:2:3, (lines 445, 452, 459)
  - Sprite ID for finishing lines.
- uint8_t finishline_width = 8; (line 441)
- uint8_t finishline_height = 3; (line 442)
- int finishline_y = 2; (line 443)
- void spawn_finishline1,2,3(); (lines 446, 453, 460)

  - Sprite size, and positioning.
- void create_sprites(); (line 468)
  - Initialise finishing line sprites in the beginning of game.
- void spawn_sprites(); (line 505)
  - Draw finishing lines to be visible in game.
  - Set finishing line speed based on player speed.

Test Case	Test description and setup	Expected outcome	Actual outcome
Car stationary	Do not move the car.	The distance value should still be the same, whether the car was initially stationary or was once moving but then stopped.	The distance travelled does indeed remain the same, which can be checked in the pause menu.
Car moving at constant speed	Increase the speed of the car to make it move by pushing the joystick up.	The distance travelled should accumulate how long the player has gone. Distance also accumulates at a rate proportional to the speed of the car.	At a constant speed, the distance travelled has recorded how long the car has been moving, which can be seen in the pause menu.

Collision

Despite the sceneries and obstacles that come down the map, the player will be damaged if they collide at any one of the objects. If the player hits a scenery or object, they take damage and lose 10 HP as reflected so in the dashboard. This will reposition the player to another location on the map where they can resume properly with a fuel tank and starting stationary speed. If the player collides onto a fuel station or takes a lot of damage to the point where their health is 0, then the game is over.

Global variables / Functions used:

- int health = 100; (line 29)
  - Health value of player.
- Bool collided (Sprite player, Sprite sprite); (line 592)
  - Collision detection Boolean statement for player and object
- void damage_function(); (line 816)
  - Colliding onto any scenery, obstacle will result in losing 10 health value.
  - Player will respawn back to original position, and obstacles will be wiped out and will eventually appear again once the player moves.

Test Case	Test description and setup	Expected outcome	Actual outcome
Scenery			

Head-on collision	Try and hit a scenery object by driving forward to one.	The player should take damage and lose 10 health.	The player does indeed lose 10 health value. In addition, the player also respawns back to the middle of the road with a full fuel tank.
Left side collision	Try and hit a scenery object by colliding one on the left side of the car,		
Right side collision	Try and hit a scenery object by colliding one on the right side of the car,		
Obstacle			
Head-on collision	Try and hit an obstacle object by driving forward to one.	The player should take damage and lose 10 health.	The player does indeed lose 10 health value. In addition, the player also respawns back to the middle of the road with a full fuel tank.
Left side collision	Try and hit an obstacle object by colliding one on the left side of the car,		
Right side collision	Try and hit an obstacle object by colliding one on the right side of the car,		
Fuel Depot			
Head-on collision	Try and hit a fuel depot object by driving forward to one.	The player should lose all health upon colliding the fuel depot.	The player instantly loses and the game is now over.
Left side collision	Try and hit a fuel depot object by colliding one on the left side of the car,		
Right side collision	Try and hit a fuel depot by colliding one on the right side of the car,		

Game Over Dialogue

The game ends in several ways: one where the player wins the game by reaching to their destination, or loses by receiving heavy damages or running out of fuel. When the player wins, they will be greeted with a victory title and their stats of the game. This also applies to when losing the game as well. The player can either replay the game or finish shown by the bottom message in the LCD which tells the player to press ‘SW2’ to restart. This will reset the game counters and start a new game. If the player quits, then the program will end.

Global variables / Functions used:

- bool game_over = false; (line 27)
  - To indicate whether the game has ended or not.
- void end_game();
  - The game over screen; an if statement is applied to determine if the player wins or lose.
- void process (void); (line 877)
  - Gameover is activated if player loses all health or finishes the game.
- void fuel_consumption(); (line 716)
  - Game is over when fuel is empty.

Include in test plan:

- Test cases for (Player wins; Player does not win).
- Indicate in report clearly how you restart the game.

Test Case	Test description and setup	Expected outcome	Actual outcome
Player wins	You must reach 5km (5000) to finish the game.	If the player successfully reaches their destination, then they will be greeted with a 'You win!' message.	After reaching 5km, the winning title is displayed, with the distance travelled and time elapsed, followed by a message to play again by pressing SW2.
Player does not win	You must attempt to lose all your health, run out of fuel, or collide onto a fuel depot.	If the player successfully reaches their destination, then they will be greeted with a 'Game Over' message	As a result, the losing title is displayed, with the distance travelled and time elapsed, followed by a message to play again by pressing SW2

Part B – Extend Game

Curved Road

Accelerator and brake

More realistic steering

The player's movement is defined by the speed in which the player is moving, indicated by the speed on the dashboard. This will affect the steering control of the vehicle when the speed is increased. The faster the player goes, the faster the car's lateral motion.

Global variables / Functions used:

- int speed = 0; (line 30)
  - speed value of player.
- void controls (void); (line 771)
  - lateral motion is met with the value of speed

Test Case	Test description and setup	Expected outcome	Actual outcome
Realistic steering	Horizontal movement while also increasing the speed of the car.	The player should experience a change of speed on the horizontal movement the faster it goes, and when it is also slow.	By changing the speed, the lateral motion of the car also changes. The x movement is multiplied by the speed shown in line 787.

Fuel level increases gradually

As the player wishes to refuel their car, they can do so in a way that it takes 3 seconds depending how much fuel they have, or it can do partial top-ups in less than 3 seconds if their fuel capacity is high. This also reflects onto their dashboard, as the fuel counter increases in the duration of refuelling.

Global variables / Functions used:

- float fuel = 100; (line 31)
  - Fuel value for player.
- int refuel_timer = 0; (line 39)
  - Timer to count refuel.
- int refuel_count = 0; (line 40)
  - Timer to count refuel.
- void refuel_function(); (line 849)
  - Conditions states that fuel value increments, while refuel timer counts to 3 seconds.

Test Case	Test description and setup	Expected outcome	Actual outcome
Fuel level increases gradually	Locate a fuel depot and refuel your vehicle.	If the player has a low amount of fuel, then it should take at least 3 seconds to	The fuel counter increments in the duration of refuelling the

		fuel up. However, if their fuel is high and should not take longer than 3 seconds, then it should fill up before the timer ends.	vehicle and is reflected on the dashboard. The fuel begins to increment when getting in contact with a fuel depot, and line 860 says if the timer reaches 3 seconds OR fuel reaches 100, then the player is free to go.

Part C – Demonstrate Mastery

Use ADC

De-bounce all switches

Direct screen update

Timers and volatile data

Program memory OR PWM

Pixel-level collision detection

Bidirectional serial communication and access to file system

Conclusion

The TeensyPewPew game pad comes with features and abilities to that of the PC version of the game RTZM. Even so, the Teensy can play such game by having the PC ported onto it, which behaves the same way. It is in many ways unique for its switches, LEDs, an LCD screen, joysticks and two adc wheels in the back, making it compelling and more enjoyable to play on compare to other devices so far.