



# CAB202 ASSIGNMENT 1

Race To Zombie Mountain

Due 22 April 2018  
(Extended Date) Due 29 April 2018

Davina Tan  
N9741127

Tutor – Floyd Creevey

## Executive Summary

This assignment required the implementation of the game 'Race To Zombie Mountain' by using the CAB202 ZDK character-based graphics library. The full source code for the game has been submitted via AMS, submission reference number d9331c16-aabe-49e2-a3f5-08293ad0ecb7.

Majority of the required functionality of the basic game (Part A) as set out in the specification has been implemented. The only functionality that was not implemented was the function to pause the game for three seconds as the car refuels. The car does refuel to maximum when 'passing by' the fuel station, however it does come to a stop. An extension of the game (Part B) was not attempted.

# Contents

Executive Summary .....	1
Introduction .....	4
Splash Screen .....	5
Description.....	5
Global Variables and Functions.....	5
Test Plan.....	6
Border .....	7
Description.....	7
Global Variables and Functions.....	7
Test Plan.....	7
Dashboard.....	9
Description.....	9
Global Variables and Functions.....	9
Test Plan.....	10
Race Car, Horizontal Movement (Non-collision) .....	12
Description.....	12
Global Variables and Functions.....	12
Test Plan.....	13
Acceleration and Speed .....	15
Description.....	15
Global Variables and Functions.....	15
Test Plan.....	16
Scenery and Obstacles .....	19
Description.....	19
Global Variables and Functions.....	19
Test Plan.....	22
Fuel Depot.....	25
Description.....	25
Global Variables and Functions.....	25
Test Plan.....	26
Fuel .....	29
Description.....	29

Global Variables and Functions.....	29
Test Plan.....	30
Distance Travelled.....	32
Description.....	32
Global Variables and Functions.....	32
Test Plan.....	33
Collision .....	35
Description.....	35
Global Variables and Functions.....	35
Test Plan.....	37
Game Over Dialogue .....	40
Description.....	40
Global Variables and Functions.....	40
Test Plan.....	41

## Introduction

This assignment required the game 'Race to Zombie Mountain' to be created by using the CAB202 ZDK character-based graphics library. The game starts with a splash screen with the title and explains the instructions and goals while waiting for the user to press on any key to begin.

Furthermore, it consists of obstacles and scenery smoothly scrolling down the window to create the illusion of movement and the user controls a car which can move left or right to dodge objects. By allowing the condition of the car to remain intact, and not letting the car run out of fuel, the player will eventually reach Zombie Mountain and win the game. This report will explain how each subsection of the game was implemented and will include a high-level description, all the global variables used as well as all the functions used.

# 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 instructions and controls were also displayed so the user would know how to play the game. The game begins once the user presses any key.

## Global Variables and Functions

### **Global Variables**

**splash\_screen\_image** – char type variable used for sprite image

**new\_game** – bool type variable defining the game state

### **Functions**

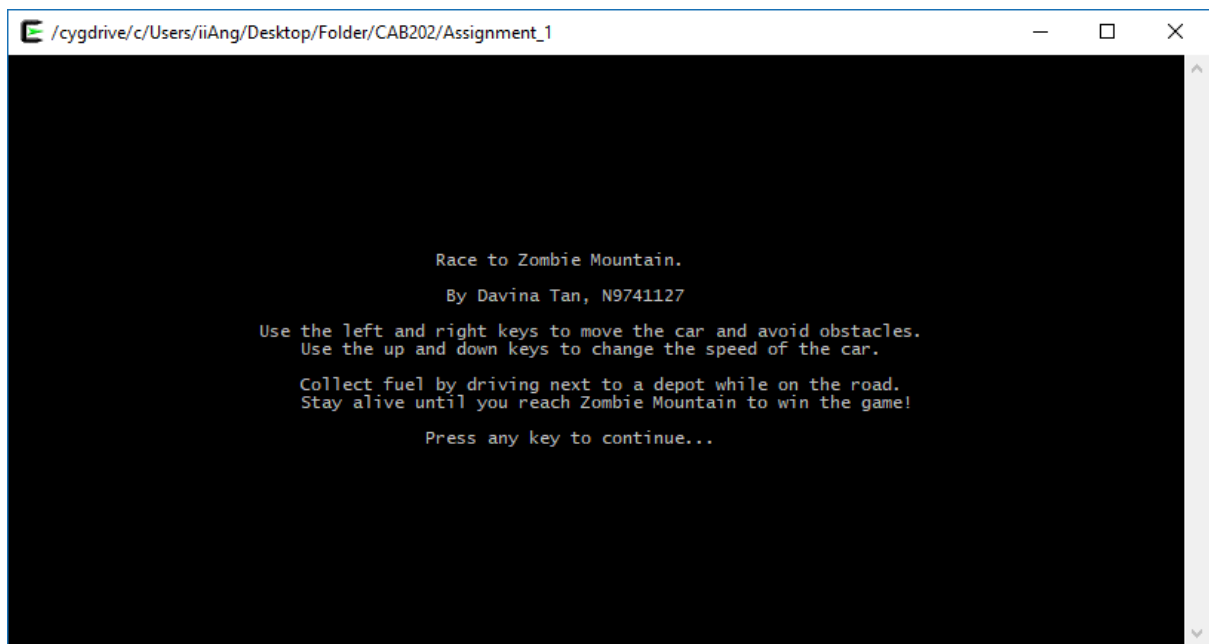
#### **void splash\_screen (void) – Lines 230:248**

When the function is called, the screen is cleared and the sprite\_id named splash\_box is created and drawn. The X and Y positions of the sprite are relative to the screen by using the functions screen\_width() and screen\_height(). The height of the splash\_box is the height of the splash\_screen\_image character. The width of the splash\_box is derived by using strlen() of the splash\_screen\_image character and dividing it by the splash\_screen\_image height. After the sprite is drawn, show\_screen() is called to show the sprite on the screen. Wait\_char() is called afterwards to wait for the user to press a key to proceed to the next screen. New\_game is set to false so in the functions in the main function do not keep repeating.

#### **int main (void) – Lines 335:355**

This function begins by setting up the screen by calling the function setup\_screen(). It then includes an if statement and while loop to process the game. The logic for the code is: If there is a new game, setup the game, show the screen and then show the splash screen. While the game is not over, process the game and if the global bool variable update\_screen is true, then show the screen. This logic is implemented by calling the functions setup(), show\_screen() and splash\_screen(). The main function ends by returning zero.

## Test Plan



Splash screen showing program name, author, student number, game controls and instructions.

# Border

## Description

A border was added to the game to act as boundaries for the car's movement as well as set limitations for where objects could spawn. The border was created out of the character '\*' and surrounds the edges of the terminal window. The border will adapt to the size of the window when the program begins to run.

## Global Variables and Functions

### Global Variables

None

### Functions

#### void draw\_game(void) – Lines 436:456

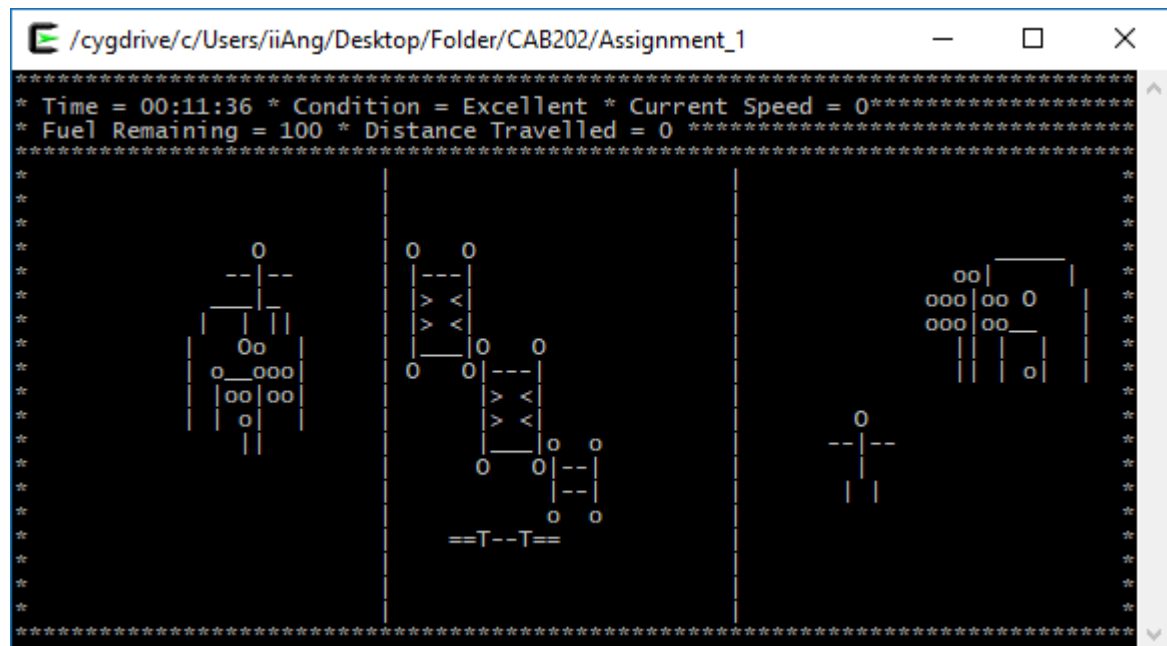
The function draw\_borderDashboard() is called.

#### void draw\_borderDashboard(void) – Lines 458:480

At the beginning of the function, local int variables left, right, top, bot and local char variable border\_char are created for each parameter of the draw\_line() function. Left and top are both set to zero. Right uses the screen\_width() function and minuses one. Bot uses the screen\_height() function and minuses one. After the local variables are called, the draw\_line() function is called four times to draw four lines on each side of the screen.

The dashboard is also drawn by using the draw\_line() function three times and adding one, two and three to the top variable. This fills the dashboard area with the same character as the border.

## Test Plan



Test case 1 – 80 x 24 (minimum dimensions)





# Dashboard

## Description

The dashboard was implemented at the top of the game to show the elapsed time since the start of the game accurate to the nearest 1/100<sup>th</sup> of a second (millisecond). It also displays the current speed and condition of the car. The amount of fuel remaining as well as the total distance travelled since the start of the game is also displayed on the dashboard. The dashboard is separated by the same character used for the border of the game. The dashboard is also never obscured by any obstacles/scenery.

## Global Variables and Functions

### **Global Variables**

**game\_timer** – timer\_id type variable to initialise timer

**milliseconds** – int type variable set at 0 to set the initial milliseconds

**seconds** – int type variable set at 0 to set the initial seconds

**minutes** – int type variable set at 0 to set the initial minutes

**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

**distance\_travelled** – double type variable set at 0 to set the initial distance travelled

**current\_condition** – char type variable set at “Excellent”

### **Functions**

#### **void setup (void) – Lines 188:209**

The global variable game\_timer was created using the function create\_timer().

#### **void update\_timer (void) – Lines 213:228**

This function involves three if statements. The first if statement checks if the game\_timer has expired and resets it then adds one to the global int variable milliseconds. The second if statement increases the global int variable seconds by one and resets milliseconds to zero once milliseconds has reached one hundred. The last if statement increases the minutes by one and resets seconds to zero once seconds has reached sixty.

#### **void process (void) – Lines 251:332**

The function draw\_game() is called.

#### **void draw\_game (void) – Lines 436:456**

The function draw\_road() and draw\_borderDashboard() is called. The function to draw the road is called before all the sprites are drawn so the road is ‘underneath’ the car when the car drives over the road.

### **void draw\_borderDashboard (void) – Lines 458:480**

At the end of the function, the function draw\_formatted() is called five times for each specification needed. The time, condition and current speed are on the first line of the dashboard therefore the local variable (from the Border subsection) top adds one. For the fuel remaining and distance travelled, top adds two so they wrap underneath. This is to meet the minimum screen requirements so the fuel remaining and distance travelled do not go off the screen.

### **void draw\_road (void) – Lines 484:494**

This function begins with four local int variables being declared as the top, bottom, left and right coordinates for the road. The functions screen\_height() and screen\_width() are used in the calculations for the coordinates. A local char variable is also named to choose the character to draw the road with. The road is drawn function draw\_line() called twice, the parameters being the local variables declared earlier in the function.

### **void update\_car (int key) – Lines 496:544**

Changes the value of current speed depending on which key is pressed (see subsection acceleration and speed).

### **void update\_distance (void) – Lines 736:759**

Changes the value of the distance travelled (see subsection distance travelled).

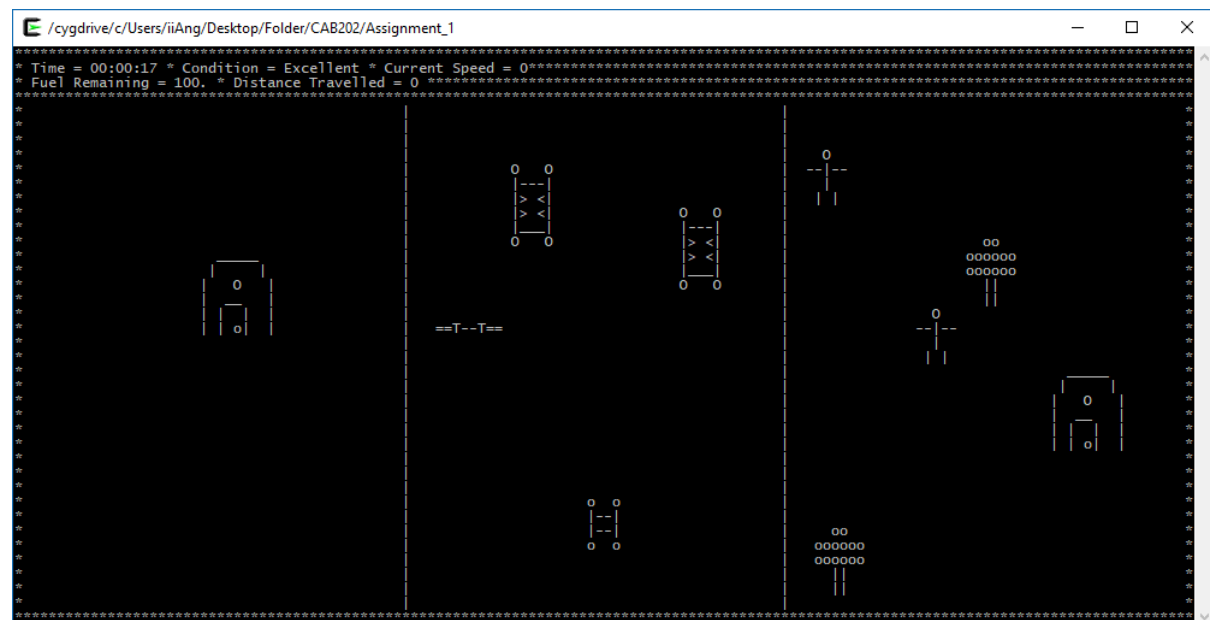
### **void update\_fuel\_remaining (void) – Lines 779:804**

Changes the value of the fuel remaining (see subsection fuel).

### **void update\_collision (void) – Lines 856:876**

Changes the value of the current condition (see subsection collision).

## **Test Plan**



**Start of game** – The game timer starts straight after the user presses a key on the splash screen even if the car does not move and no distance has been made. Fuel can be seen full (100).



# Race Car, Horizontal Movement (Non-collision)

## Description

The sprite of the race car is four units wide and four units 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 one unit left or right dependent on the key the user presses. The left and right arrow keys 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\_id type variable for the car

**current\_speed** – int type variable set at 0 to set the initial current speed

**car\_image** - char type variable used for car image

### **Functions**

#### **void setup (void) – Lines 188:209**

The function setup\_car() is called.

#### **void process (void) – Lines 251:332**

The function update\_car(key) is called.

#### **void setup\_car (void) – Lines 358:362**

This function begins by declaring two local int variables for the initial x and y spawn location of the car. These are calculated by using the screen\_height() and screen\_width() functions so the car will always spawn in the middle of the screen even when the screen dimensions change. The car sprite is created using sprite\_create(). The parameters are the two local int variables, the car width and height which were defined at the beginning of the program (**lines 14:15**) as well as the global char car\_image.

#### **void draw\_game (void) – Lines 436:456**

Using the function sprite\_draw() the car sprite is drawn.

#### **void update\_car (int key) – Lines 496:544**

Within the function, there are embedded if statements to constrain the movement of the car relative to the speed. The logic of the code is: if the up or down key is pressed, increase or decrease the speed (see embedded if statements in subsection acceleration and speed). The input parameter of the function is int key and for the two if statements, KEY\_UP and KEY\_DOWN were used to control the current speed (see subsection acceleration and speed).

The logic for the last if statement in the function is: if the current speed is greater than zero, allow the car to move left and right. KEY\_LEFT and KEY\_RIGHT were chosen to move the car left and right. The if statement also involved calculating that the car's current x coordinate was not obscuring the left and right borders before allowing the car to move using the function sprite\_move(). The sprite was only moved one unit to the left or right via each key press.

## Test Plan

[illegible]

## Car middle of road with speed zero

```

C:\cygdrive\c\Users\iiAng\Desktop\Folder\CAB202\Assignment_1
Time = 00:14:46 * Condition = Excellent * Current Speed = 3
Fuel Remaining = 98.4 * Distance Travelled = 16

```

Car next to right border with speed non-zero -

Set-up actions involve pressing the up arrow and pressing the right arrow until the car hits the right border. If the right arrow is pressed more times, the car will not move to the right. The car is still able to change speeds by pressing the up and down arrow in this position.



# Acceleration and Speed

## Description

Pressing the up and down arrow keys will function as the accelerate and decelerate keys. These keys 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 up arrow key is pressed, 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.

## Global Variables and Functions

### **Global Variables**

**sprite\_id car** – sprite type variable for the car

**current\_speed** – int type variable set at 0 to set the initial current speed

**car\_image** - char type variable used for car image

### **Functions**

#### **void process (void) – Lines 251:332**

The function update\_car(key) is called.

#### **void draw\_borderDashboard (void) – Lines 458:480**

The function draw\_borderDashboard (void) draws the current speed which is changed through this function.

#### **void update\_car (int key) – Lines 496:544**

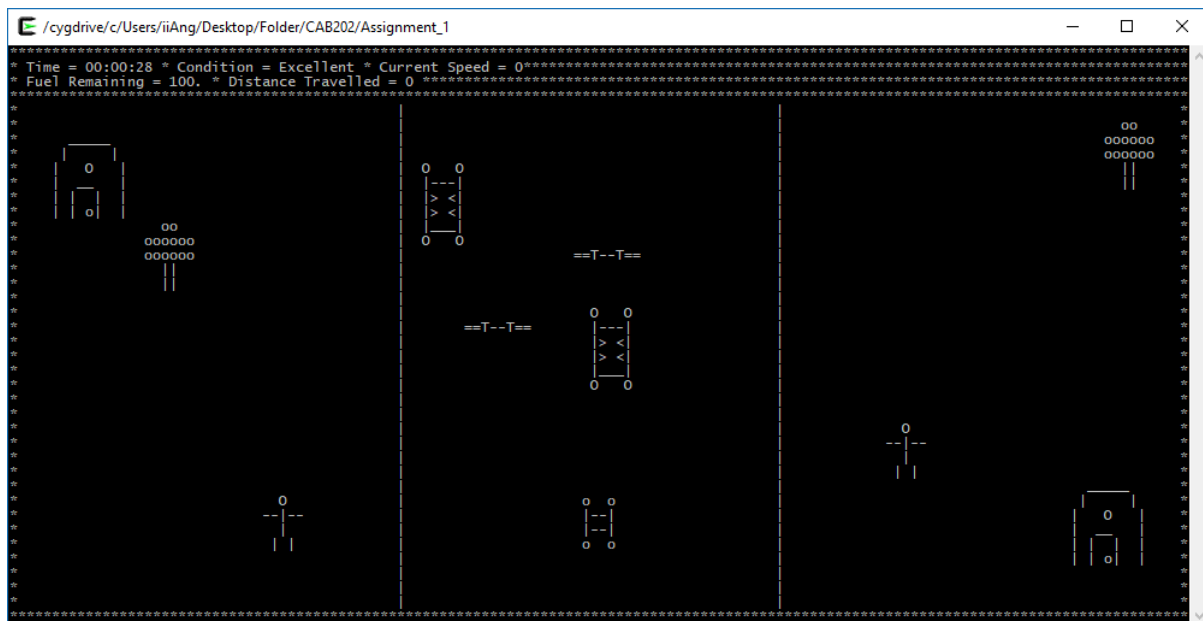
The function begins with the local int variable car\_x being declared by using the round() function to round the current x coordinates of the car using the function sprite\_x(). By using calculations, the function screen\_width() and the car's current x location the maximum speed of the car off road was controlled using two if statements. The logic for the code was: if the car is off the road but inside the border, and the speed is greater than three, change the speed to three. The road takes up the middle third of screen therefore by calculating the sections in terms of thirds, it was relatively easy to make an 'invisible border' that separated the road from off the road.

The second set of embedded if functions involved the KEY\_UP and KEY\_DOWN keys to accelerate and decelerate the car. The logic of this code was: if the up key is pressed, then increase the current speed by one and if the car is on the road then only increase the current speed if it is less than ten. If the car is off the road then only increase the current speed if it is less than three. The code involved using the function screen\_width() and the local int variable car\_x to calculate if the car was on the road. The same logic and method of code was implemented for decelerating as well.

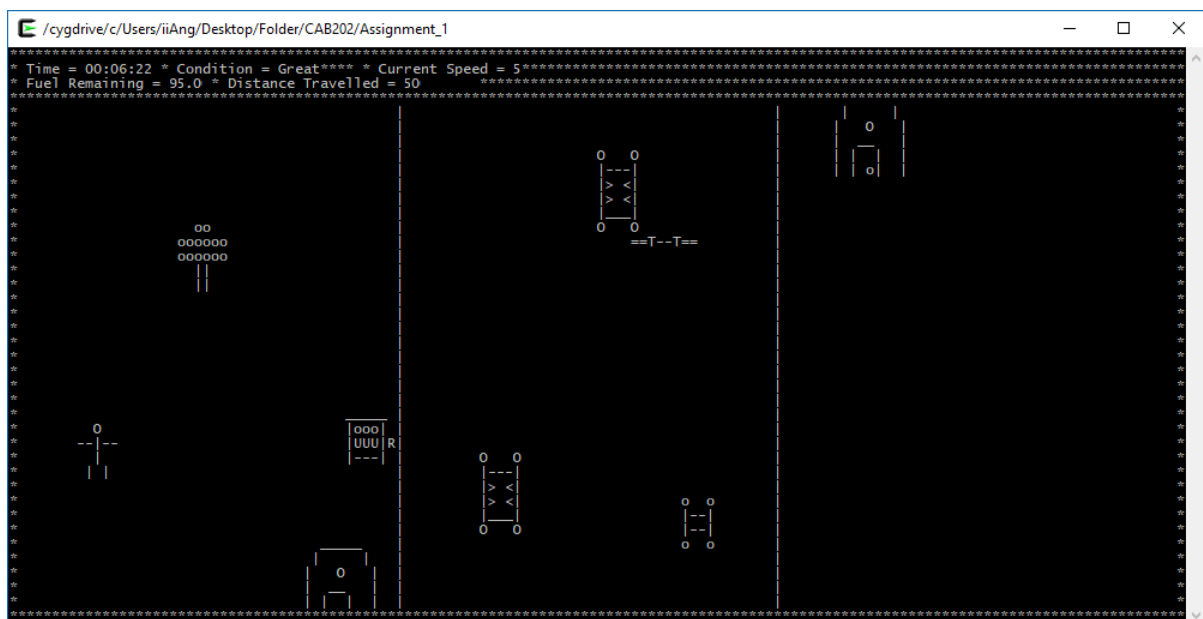


## Test Plan

**Test Case 1** – Testing that different current speeds affect the same distance travelled in different amounts of time and that it is reflected on the dashboard.



**BEFORE SHOT** - Car stationary (before accelerating)



**AFTER SHOT 1** - Car moving intermediate speed (current speed is 5/10). When the distance travelled is 50, the time taken is 6.22 seconds.



**Test Case 2** – Testing the the current speed will not increase more than the maximum speed for the area the car is currently on

```

C:/cygdrive/c/Users/iiAng/Desktop/Folder/CAB202/Assignment_1
Time = 00:08:09 * Condition = Excellent * Current Speed = 10
Fuel Remaining = 94.9 * Distance Travelled = 51

```

**BEFORE SHOT** – The car is going at maximum speed on the road (current speed is 10). If the up arrow key is pressed, the current speed will not increase further.

[illegible]

**AFTER SHOT** – The car has moved off the road and has dropped to the maximum speed off the road (current speed is now 3). If the up arrow key is pressed, the current speed will not increase further until the car moves back onto the road.

# 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 buses. The initial vertical position of the objects are randomised and the horizontal positions will continually 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

**bus** – sprite type variable for bus

**bus2** – sprite type variable for bus2

**tree\_image** – char type variable for tree and tree2

**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 and roadBlock2

**bus\_image** – char type variable for bus and bus2

**current\_speed** – int type variable set at 0 to set the initial current speed

### **Functions**

#### **void setup (void) – Lines 188:209**

The functions: `setup_tree()`, `setup_tree2()`, `setup_house()`, `setup_house2()`, `setup_person()`, `setup_person2()`, `setup_roadBlock()`, `setup_roadBlock2()`, `setup_bus()` and `setup_bus2()` are called.

#### **void process (void) – Lines 251:332**

The functions: `update_tree()`, `update_tree2()`, `update_house()`, `update_house2()`, `update_person()`, `update_person2()`, `update_roadBlock()`, `update_roadBlock2()`, `update_bus()` and `update_bus2()` are called.

#### **void setup\_tree (void) – Lines 364:368**

Declares two local int variables for the initial x and y spawn location of the tree. The initial x variable is made by calling the function rand\_offroad\_x() – see function description below. The initial y variable is made by using the rand() function to choose a random number between zero and the max screen height possible. The max screen height possible is calculated by using the function screen\_height() and taking away the height of the tree and one unit for the border. The height of the tree was defined at the beginning of the program (**line 17**). The tree sprite is created using sprite\_create(). The parameters are the two local int variables, the tree width and height which were defined at the beginning of the program (**lines 16:17**) as well as the global char tree\_image.

#### **void setup\_tree2 (void) – Lines 370:374**

Function description same as setup\_tree() as above.

#### **void setup\_house (void) – Lines 376:380**

Function description same as setup\_tree() as above. House height and width defined (**lines 18:19**) and global char house is used instead.

#### **void setup\_house2 (void) – Lines 382:386**

Function description same as setup\_house() as above.

#### **void setup\_person (void) – Lines 388:392**

Function description same as setup\_tree() as above. Person height and width defined (**lines 20:21**) and global char person is used instead.

#### **void setup\_person2 (void) – Lines 394:398**

Function description same as setup\_person() as above.

#### **void setup\_roadBlock (void) – Lines 400:404**

Function description same as setup\_tree() as above. RoadBlock height and width defined (**lines 22:23**) and global char roadBlock used instead. The initial x variable is made by calling the function rand\_onroad\_x() instead – see function description below.

#### **void setup\_roadBlock2 (void) – Lines 406:410**

Function description same as setup\_roadBlock() as above.

#### **void setup\_bus (void) – Lines 412:416**

Function description same as setup\_roadBlock() as above. Bus height and width defined (**lines 24:25**) and global char bus used instead.

#### **void setup\_bus2 (void) – Lines 418:422**

Function description same as setup\_bus() as above.

#### **void draw\_game (void) – Lines 436:456**

Using the function sprite\_draw(), the sprites: tree, tree2, house, house2, person, person2, roadBlock, roadBlock2, bus and bus2 are drawn.

#### **int rand\_offroad\_x (void) – Lines 546:564**

This function creates multiple local int variables including a minimum and maximum x position of both 'off road' sections of the window. The function screen\_width() is used in the calculations. A local variable int number for each section is calculated by using a formula which chooses a random number between two numbers. After the two random numbers have been chosen, both are put into a local int array called offroad\_array. A local int called random\_choice is made by using the rand() function again to choose between zero and one. Therefore, the chances of each side being chosen is 50/50. The final local int variable is the random position chosen by the using the random\_choice result as the index of the offroad\_array. The function returns the random position.

#### **int rand\_onroad\_x (void) – Lines 566:572**

This function creates three local int variables: a maximum and minimum x position of the 'on road' section and the calculated random x position. The maximum and minimum variables are calculated using the screen\_width() function. The final random x position is calculated using a formula which chooses a random number between two numbers. The function returns the final random x position.

#### **void update\_speed (sprite\_id sprite) – Lines 612:634**

This function updates the rate of which the sprites move which is proportional to the current speed. It takes a sprite\_id as the parameter and updates the movement of that specific sprite using if and else if statements. The logic of the code is: if the current speed is (1-10), then move the sprite this many pixels. The sprite\_move() function is used to move the sprite accordingly to the current speed. As the sprites just scroll downwards, the amount of pixels x it will move to will just be zero. However the amount of pixels y it moves is calculated by the maximum speed (defined at **line 31**) divided by ten, multiplied by the current speed.

#### **void update\_tree (void) – Lines 636:644**

This function begins with creating a local int variable by using the round() function to round the current y coordinates of the tree using the function sprite\_y(). An if statement is made to ensure that the tree sprite only moves to the top of the window by using the function sprite\_move\_to() when the tree's y position is greater than the maximum screen height (using screen\_height()) plus the tree's height (defined at **line 17**). The function will move the sprite to the top of the window (y coordinate will be zero) and choose a new random x location to spawn it by using the function rand\_offroad\_x(). Otherwise, if the tree is within the boundaries of the terminal window, the function update\_speed() (see function above) will be called using tree as the parameter.

#### **void update\_tree2 (void) – Lines 646:654**

Function description same as update\_tree() as above.

#### **void update\_house (void) – Lines 656:664**

Function description same as update\_tree() as above. The house height (defined at **line 19**) is used instead.

#### **void update\_house2 (void) – Lines 666:674**

Function description same as update\_house() as above.

#### **void update\_person (void) – Lines 676:684**

Function description same as update\_tree() as above. The person height (defined at **line 21**) is used instead.

### **void update\_person2 (void) – Lines 686:694**

Function description same as update\_person() as above.

### **void update\_roadBlock (void) – Lines 696:704**

Function description same as update\_tree() as above. The roadBlock height (defined at **line 23**) is used instead. The function rand\_onroad\_x() is used to find the new x position that the sprite moves to instead.

### **void update\_roadBlock2 (void) – Lines 706:714**

Function description same as update\_roadBlock() as above.

### **void update\_bus (void) – Lines 716:724**

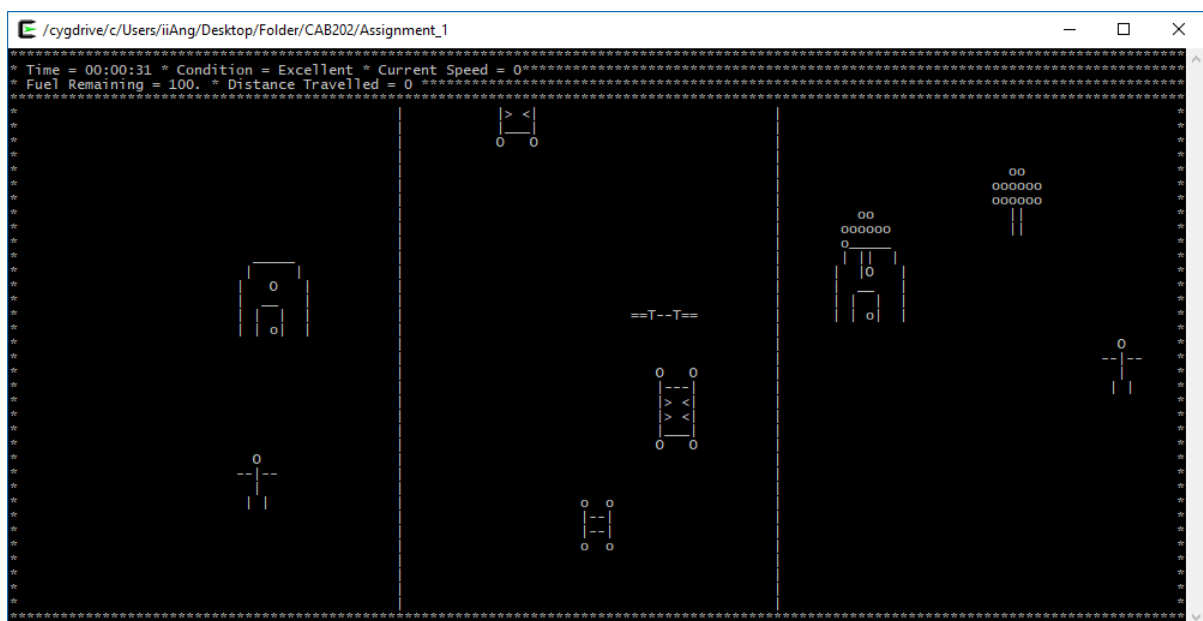
Function description same as update\_roadBlock() as above. The bus height (defined at **line 25**) is used instead.

### **void update\_bus2 (void) – Lines 726:734**

Function description same as update\_bus() as above.

## **Test Plan**

**Test Case 1** – Testing that scenery and obstacles move relative to the car's speed (current speed) with the car moving at an intermediate speed (5/10).



**At zero seconds** – Car is stationary so scenery and objects are stationary. It can be seen that the bus sprite at the top of the road is not obscuring the dashboard.

```

/cygdrive/c/Users/iiAng/Desktop/Folder/CAB202/Assignment_1
* Time = 00:06:34 * Condition = Excellent * Current Speed = 5
Fuel Remaining = 97.4 * Distance Travelled = 26

```

At six seconds – Car is moving at an intermediate speed, scenery and objects have moved down the window.

```

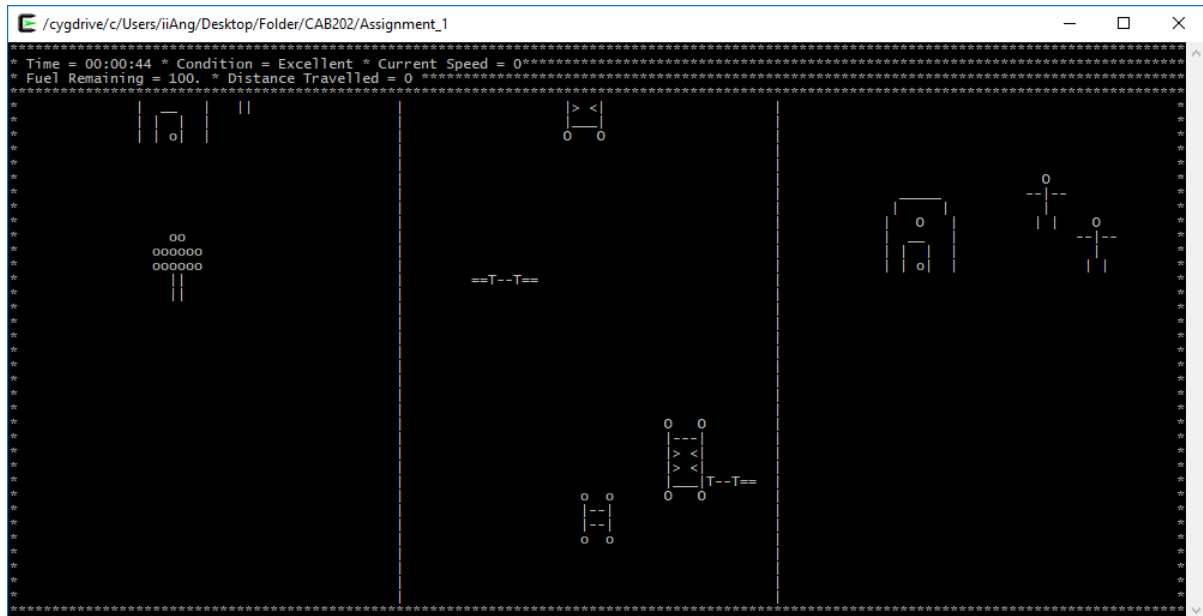
/cygdrive/c/Users/iiAng/Desktop/Folder/CAB202/Assignment_1
* Time = 00:09:63 * Condition = Excellent * Current Speed = 5
Fuel Remaining = 94.7 * Distance Travelled = 53

```

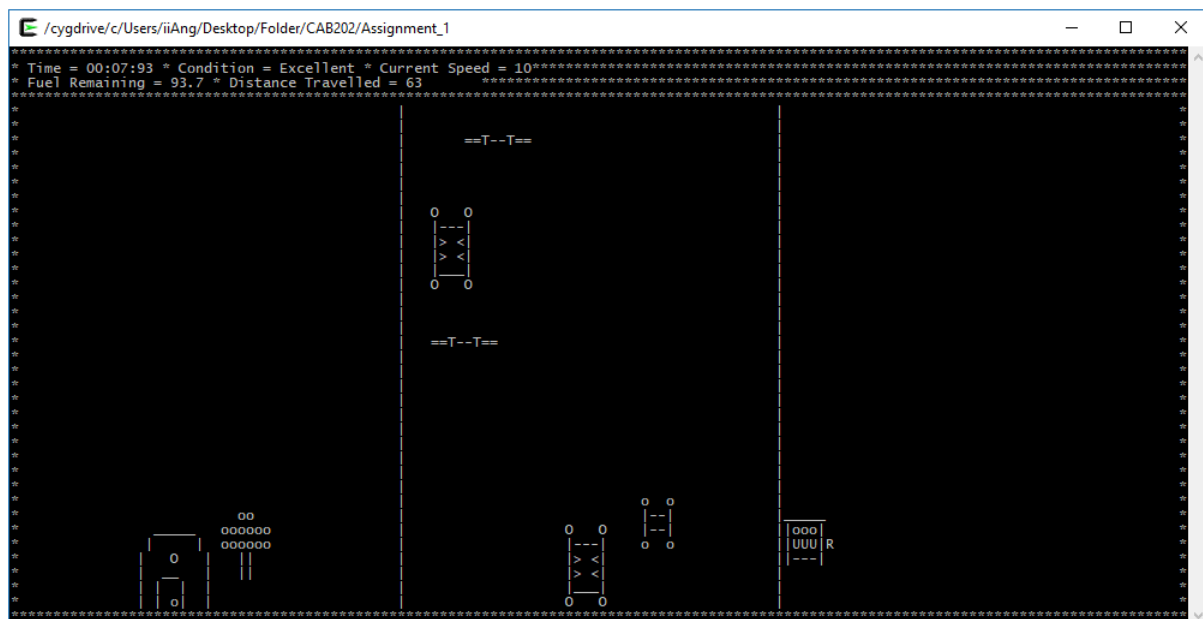
At nine seconds – Car still moving at an intermediate speed, scenery and objects have moved further down the window. Some sprites have now disappeared below the screen but are not obscuring the border.



**Test Case 2** – Testing that scenery and obstacles move relative to the car's speed (current speed) with the car moving very fast (10/10).



At zero seconds – Car is stationary so scenery and objects are stationary.



At seven seconds – Car moves at a very fast speed, scenery and objects have moved further down the window quicker than going at an intermediate speed. The house and tree sprites to the left have travelled a long distance in a shorter amount of time.

# Fuel Depot

## Description

A fuel depot will occasionally spawn next to the road with an equal chance of appearing on either side. The fuel depot is six units wide and four units high. 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** – char type variable for the fuel image

### **Functions**

#### **void setup (void) – Lines 188:209**

The function setup\_fuel() is called.

#### **void process (void) – Lines 251:332**

The functions update\_fuel() and update\_fuel\_remaining() are called.

#### **void setup\_fuel (void) – Lines 430:434**

This function starts by declaring two local int variables that determine the fuel's x and y spawn coordinates. The x coordinate is created by calling the function rand\_fuel\_x() (see function below) and the y coordinate is set to zero. The fuel sprite is created using the function sprite\_create() and the parameters include the two local int variables, the height and width of the fuel (defined at **lines 28:29**), as well as the char variable fuel image.

#### **void draw\_game (void) – Lines 436:456**

The fuel sprite is drawn using the function sprite\_draw() and fuel as the parameter.

#### **int rand\_fuel\_x (void) – Lines 574:587**

This function chooses which side of the road to spawn at with equal probability. First of all, two local int variables are made to store the x position of each side of the road. The two variables are then stored into an int array and another local int variable called random\_choice chooses a number between 1 and 0 by using the rand() function. The result of random\_choice is used as the index of the array hence choosing one side out of the two equally and storing it into a new local int variable. The function returns the new local int variable as the new random x coordinate of the fuel's position.

#### **int rand\_fuel\_y (void) – Lines 589:610**

This function chooses a random y coordinate below the screen at which the fuel depot reaches and then moves to the top of the window (see update\_fuel() function below). The function begins with declaring two local int variables named max and min that are both initialised as zero. If and else if statements are written to determine when the fuel depot should move to the top relative to how much fuel is remaining. The logic of the code is: the lower the fuel remaining, the higher the chance of the fuel moving to the top. The max and min variables were altered depending on how much fuel

was remaining and then a local int variable was calculated at the end of the function using the formula that chooses a random number between two numbers (the function rand() is used). The result of that local int variable is returned at the end of the function.

#### **void update\_speed(void) – Lines 612:634**

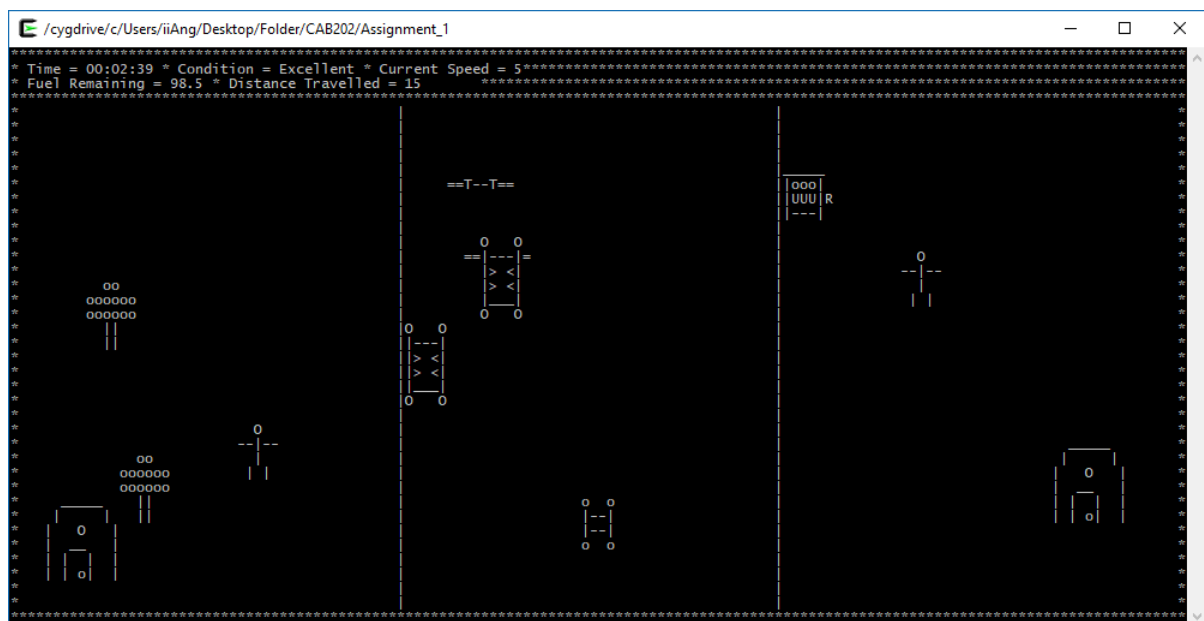
This function is called in the function update\_fuel() to move the fuel depot relative to the current speed (see subsection scenery and obstacles for in-depth description of the function).

#### **void update\_fuel(void) – Lines 767:777**

This function begins with creating a local int variable by using the round() function to round the current y coordinates of the fuel depot using the function sprite\_y(). There are two main if statements in this function that control the movement and the intervals between spawning. The first if statement says that if the fuel is greater than zero, move the fuel using the function update\_speed() and fuel as the parameter. The second if statement states that if the fuel's current y coordinate is greater than the maximum screen height (by using the function screen\_height()) plus the fuel's height (defined at **line 29**) plus the random extra value taken from the function rand\_fuel\_y(), then move the fuel depot to the top of the screen by making the new y coordinate zero.

### **Test Plan**

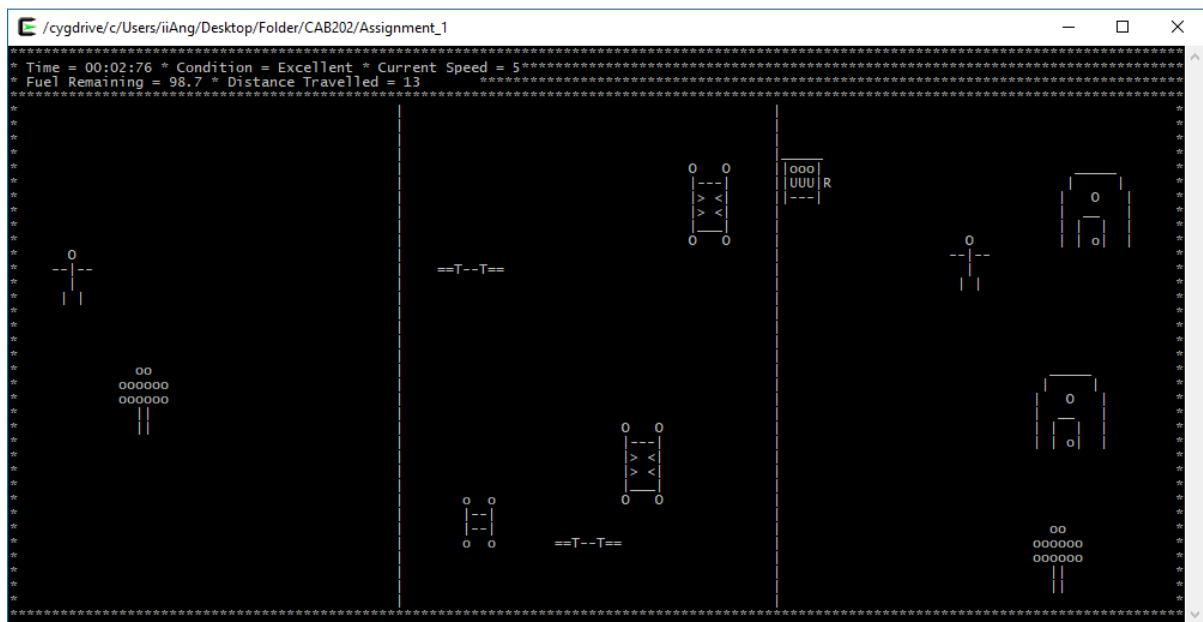
**Test Case 1** – Testing that the fuel depot moves relative to the car's speed (current speed) with the car moving at an intermediate speed (5/10).



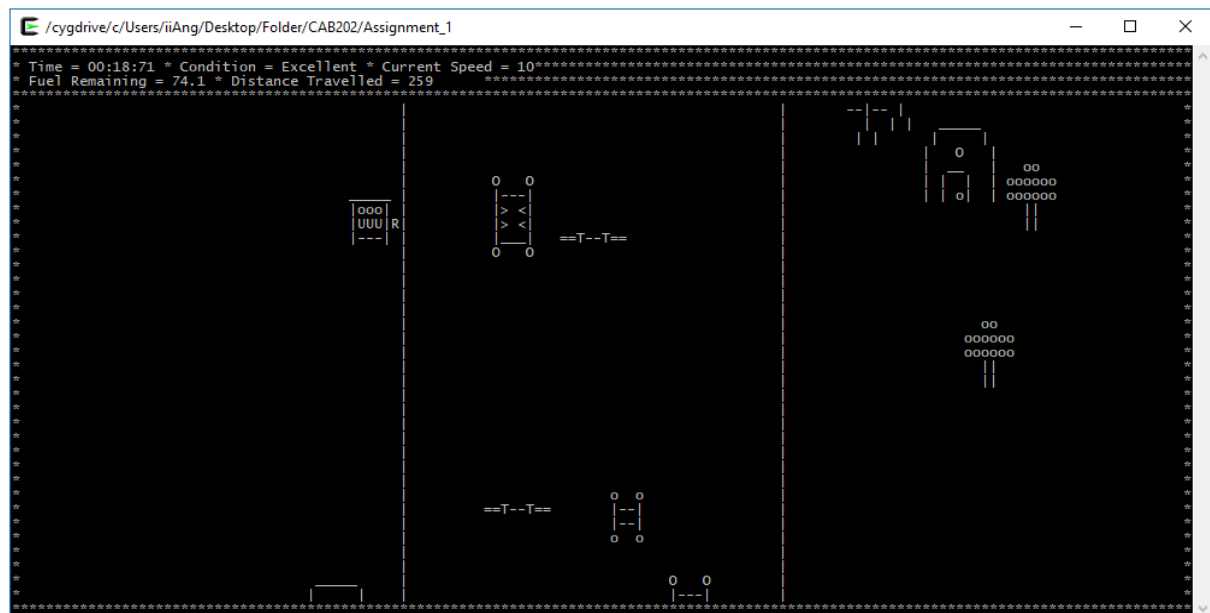
**At two and a half seconds** – Car is moving at an intermediate speed, the fuel depot is at the top of the window.



## Test Case 2 – Testing that the fuel depot has a chance of appearing on the left and right of the road.



Beginning of new game – Fuel depot has spawned on the right side of the road



At nearly nineteen seconds – Fuel depot has spawned on the left side of the road

# Fuel

## Description

The fuel is a feature that has been implemented into the game for the user to refill the car and have the chance of losing by running out of fuel. The car must have fuel in order 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 used 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\_id type variable for the car

**fuel** – sprite\_id type variable for the fuel depot

**fuel\_remaining** – double type variable set at 100 to set the initial fuel remaining

**fuel\_image** – char type variable for the fuel image

### Functions

#### void process (void) – Lines 251:332

The functions update\_fuel\_remaining() and fuel\_refill\_collision() are called.

#### void update\_fuel\_remaining (void) – Lines 779:804

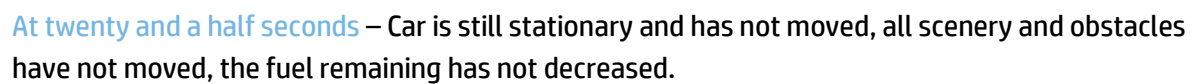
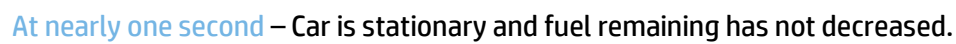
This function updates the fuel remaining and calculates the consumption rate relative to the speed of the car. The function contains an if, embedded if and else if statements. The logic of the code is: if the fuel remaining is greater than one, and if the current speed is (1 – 10) then the fuel remaining will minus equal  $1/100000 \times$  the current speed.

#### void fuel\_refill\_collision (void) – Lines 839:854

This function tests to see if the car has collided with the 'refill collision box' and if it has, to refill the car to maximum fuel. The function starts by creating local int variables that use the round() function and the sprite\_x()/sprite\_y() functions to find the current x and y coordinates of both the car and fuel sprites. Afterwards, there are embedded if statements to calculate if the car has entered the 'refill collision box'. The logic for this code is: if the y coordinate of the car and fuel is the same, and if the car has entered the 'refill collision box' which is a pixel larger than the collision box of the fuel depot, then refill the fuel. The car only refills fuel next to the fuel depot when the car is on the road beside it. The car will not refill fuel when beside the fuel depot but off the road.

*The requirement of parking the car for three seconds before refilling to the fuel to maximum was not satisfied. However, if the car does 'pass by' the 'refill collision box' then the fuel will refill to the maximum. (See test case 2 below)*

### Test Case 1 – Testing that the car has not moved and the fuel has not decreased



### Test Case 2 – Testing that the fuel refills at the fuel depot

```

C:/cygdrive/c/Users/iAng/Desktop/Folder/CAB202/Assignment_1
Time = 00:21:31 * Condition = Excellent * Current Speed = 1*****
Fuel Remaining = 96.0 * Distance Travelled = 40*****

      |   |
      | o |
      |__|
      | o |
      |   |

=====

          O      O
          ---
          v      ^
          ^      v
          O      O

==T--T==

                    OOOO
                    UUUU R
                    ---

O      O
--    --
|      |
--    --
O      O

==T--T==

              [_____]
              |       |
              |   O   |
              |_____|

=====

```

**Fuel depot approaching** – The car approaches the fuel depot on the side of the road, the fuel is below maximum.

[illegible]

**Next to fuel depot** – The car can be seen next to the fuel depot but still on the road. The fuel remaining has been filled up and can be seen on the dashboard.



# Distance Travelled

## Description

The dashboard reflects the total amount of distance travelled during the game. 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 (1000) it reaches Zombie Mountain and the player wins the game by passing through a finish line. After 1000 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\_id type variable for the finishLine

**distance\_travelled** – double type variable initially set at 0

**finishLine\_image** – char type variable for the finishLine image

### Functions

#### **void setup (void) – Lines 188:209**

The function setup\_finishLine() is called.

#### **void process (void) – Lines 251:332**

The functions update\_distance(), update\_finishLine() and finishLine\_collision() are called.

#### **void setup\_finishLine (void) – Lines 424:428**

This function begins by declaring two local int variables for the initial x and y spawn location of the finish line. These are calculated by using the screen\_height() and screen\_width() functions so the car will always spawn in the middle of the screen even when the screen dimensions change. The car sprite is created using sprite\_create(). The parameters are the two local int variables, the finish line width and height which were defined at the beginning of the program (**lines 26:27**) as well as the global char finishLine\_image.

#### **void draw\_game (void) – Lines 436:456**

The finishLine sprite is drawn using the function sprite\_draw() with the finishLine as the parameter.

#### **void update\_distance (void) – Lines 736:759**

This function updates the distance travelled which is reflected on the dashboard always. The distance increases at a rate proportional to the speed of the car. The function contains if and else if statements. The logic of the code is: if the current speed is (1-10) then the distance travelled will increase by  $1/10000 * \text{the current speed}$ .

#### **void update\_finishLine (void) – Lines 761:765**

This short function includes one if statement. The logic is: if the distance travelled is greater than or equal to the maximum distance (defined at **line 32**) then move the finish line to scroll down the window along with the other scenery and obstacles. The movement of the finish line is done by calling the function update\_speed() and using finishLine as the parameter.

### void finishLine\_collision (void) – Lines 912:920

The function begins by creating local int variables that use the round() function and the sprite\_y() function to find the current y coordinates of both the car and finishLine sprites. To trigger the game over screen after the car has crossed the line instead of just colliding with it, an if statement was added to the function. The logic of the code is: if the y coordinate of the car is equal to the y coordinate of the finish line minus the car's height (defined at **line 15**) then call the function do\_game\_over().

## Test Plan

**Test Case 1** – Testing that the distance does not increase when the car does not move (current speed is zero)

```

C:\cygdrive\c\Users\iiAng\Desktop\Folder\CAB202\Assignment_1
Time = 00:00:28 * Condition = Excellent * Current Speed = 0
Fuel Remaining = 100. * Distance Travelled = 0

```

**At nearly one second** – Car is stationary, and distance travelled has not increased.

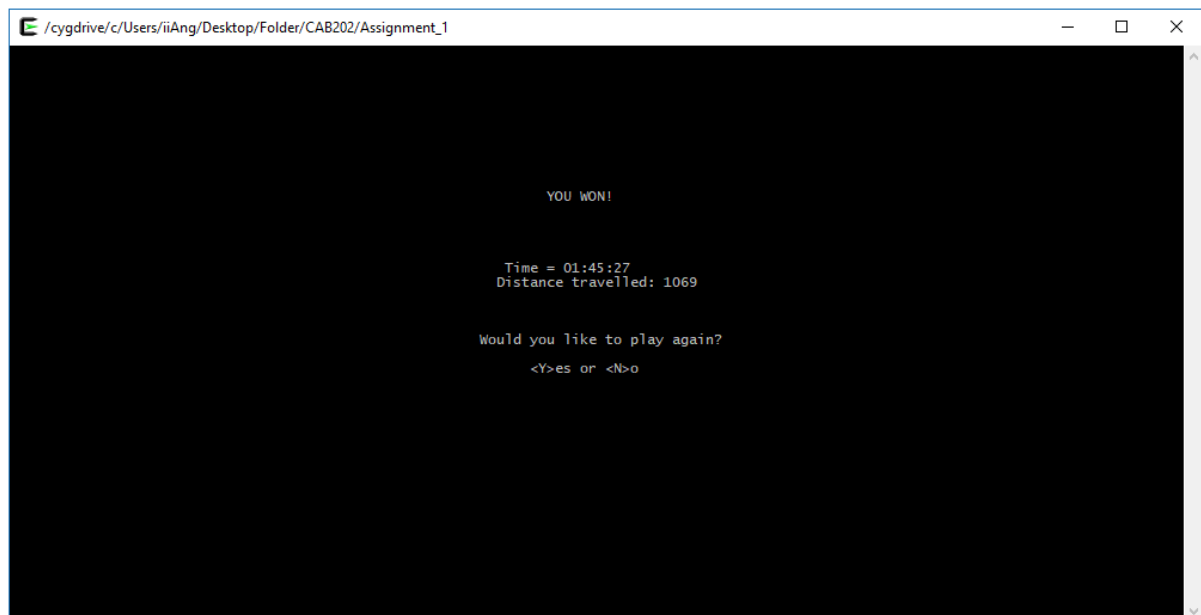
[illegible]

**At thirty seconds** – Car is still stationary, and distance travelled has not increased.

**Test Case 2** – Testing that after the car crosses the finish line, the victorious game over dialogue is displayed



**Crossing the finish line** – The dashboard displays a distance greater than 1000 and the user can see the car crossing the finish line.



**Victorious game over screen** – Victorious quote ‘You won!’ is displayed, showing the time and distance travelled.

# 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

**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

**bus** – sprite type variable for bus

**bus2** – sprite type variable for bus2

**car\_image** – char type variable for the car

**tree\_image** – char type variable for tree and tree2

**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 and roadBlock2

**bus\_image** – char type variable for bus and bus2

**current\_condition** – char type variable set at “Excellent”

**game\_over** – bool type variable set as false

### **Functions**

#### **void setup (void) – Lines 188:209**

The functions to setup all sprites are called.

#### **void process (void) – Lines 251:332**

The functions to update all sprites are called. Many if statements are made to check if any scenery or obstacle sprite has collided with the car sprite by calling the function `sprites_collided()` (see function as below). If there has been a collision, the function `update_collision()` (see function as below) is called. If the car has collided with the fuel depot by using the `sprites_collided()` function then the condition equals zero and the function `do_game_over()` is called.

### **bool sprites\_collided (sprite\_id sprite1, sprite\_id sprite2) – Lines 807:837**

This function starts by declaring a local bool variable collided is true. Local int variables defining the top, bottom, left and right coordinates of each sprite used for the parameters are calculated using the functions round(), sprite\_y(), sprite\_x(), sprite\_height() and sprite\_width(). Four if statements are made which calculate if any of the sides of one sprite has not overlapped with any of the sides of the other sprite. If the four if statements apply then collided is false and the game continues however if there is a collision the collided will be true. The function returns the bool variable collided.

### **void update\_collision (void) – Lines 856:876**

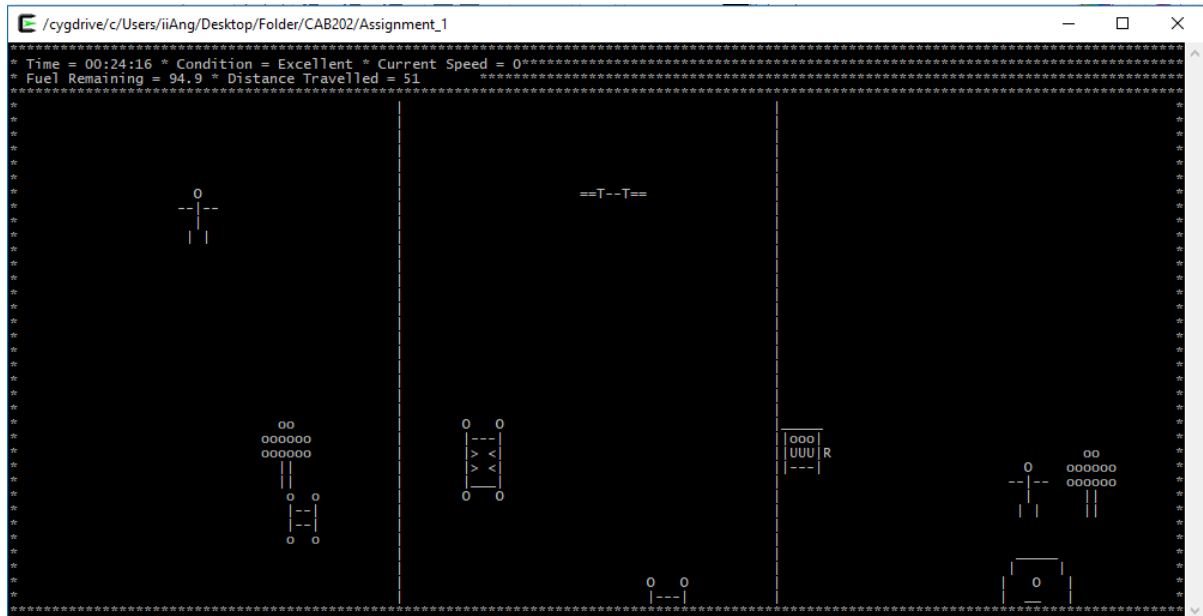
This function is called if a collision has been made and it changes the condition which is displayed on the dashboard. The function contains if, embedded if and else if statements. The logic of the code is: if the condition is greater than zero, minus a quarter (twenty five) of the full condition. If the condition is seventy five, display “Great” in the dashboard as the current condition, else if the condition is 50, display “Good” in the dashboard as the current condition, else display “Bad” as the current condition. A final if statement is stated and the logic is: if the condition equals zero, display game over screen by calling the function do\_game\_over(). After a collision, the function car\_safe\_spot() (see function as below) is called to move the car to a safe spot on the road. The current speed is set to zero and the fuel is refilled to maximum.

### **bool car\_safe\_spot (void) – Lines 878:910**

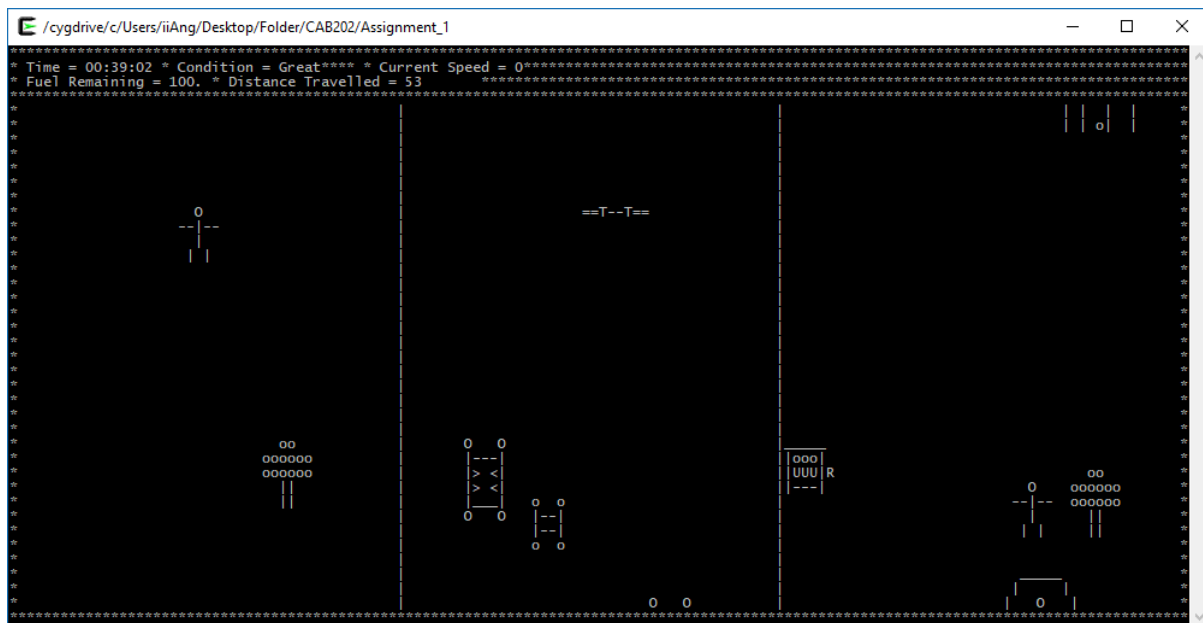
This functions begins with a local bool variable named safe being false. A while loop is used and the logic of the code is: while safe is false, continue to move the car to a random x position on the road using the function rand\_onroad\_x(). To check if the car is in a safe spot, if and else if statements are made while calling the function sprites\_collided() to check if the car has collided with any other sprite. If the car is in a new random x position on the road and has not collided with any other sprite then safe equals true and the while loop ends, and the function returns safe.

## Test Plan

### Test Case 1 – Testing of a Head- on collision with scenery

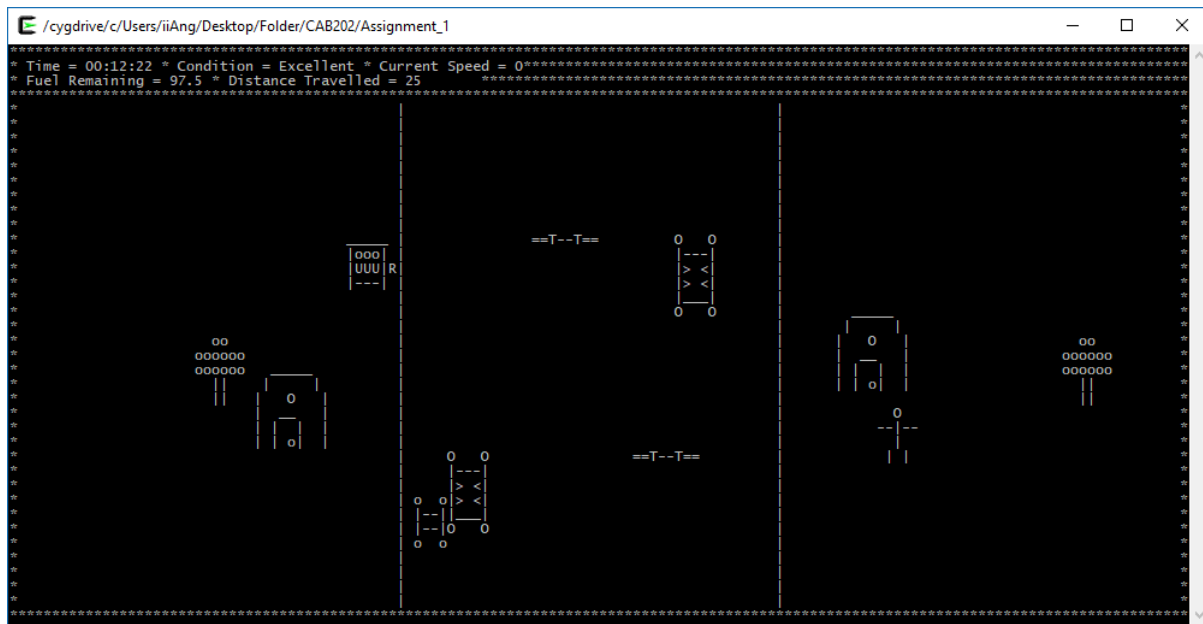


### Car approaching scenery head on



**Car hit and respawn on road** – Car has respawned on a safe spot on the road. The dashboard has reflected the damage via the condition, the current speed is set to zero and the fuel remaining has been refilled.

### Test Case 2 – Testing of a left side collision with obstacle



### Car immediately adjacent to obstacle




**Car hit and respawn on road** – Car has respawned on a safe spot on the road. The dashboard has reflected the damage via the condition, the current speed is set to zero and the fuel remaining has been refilled.

### Test Case 3 – Testing of a right side collision with fuel depot

```
/cygdrive/c/Users/iiAng/Desktop/Folder/CAB202/Assignment_1
```

```
** Time = 00:14:41 * Condition = Great** * Current Speed = 0  
** Fuel Remaining = 94.2 * Distance Travelled = 58 **  
  
| | | > <  
o o o o  
o o o o  
  
[ ] [ ]  
o o  
o o  
  
o o  
-- --  
<< >>  
o o  
  
==T--T==  
  
oo  
ooooooo  
ooooooo  
||  
||  
  
ooo  
uuu Ro o  
--- --- o  
o o  
o o  
  
o  
-- --  
| |  
  
oo
```

Car immediately adjacent to fuel depot



```

/cygdrive/c/Users/iiAng/Desktop/Folder/CAB202/Assignment_1
GAME OVER!

Time = 00:21:03
Distance travelled: 59

Would you like to play again?
<Y>es or <N>o

```

**Game over screen** – The game over is triggered when the car collides with the fuel depot. ‘Game over!’ is displayed, showing the time and distance travelled.



# Game Over Dialogue

## Description

When the game ends, a screen will display which will have either a victorious or game over dialogue. 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**

**game\_timer** – timer\_id type variable to initialise timer

**milliseconds** – int type variable set at 0 to set the initial milliseconds

**seconds** – int type variable set at 0 to set the initial seconds

**minutes** – int type variable set at 0 to set the initial minutes

**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

### **Functions**

#### **void do\_game\_over (void) – Lines 922:968**

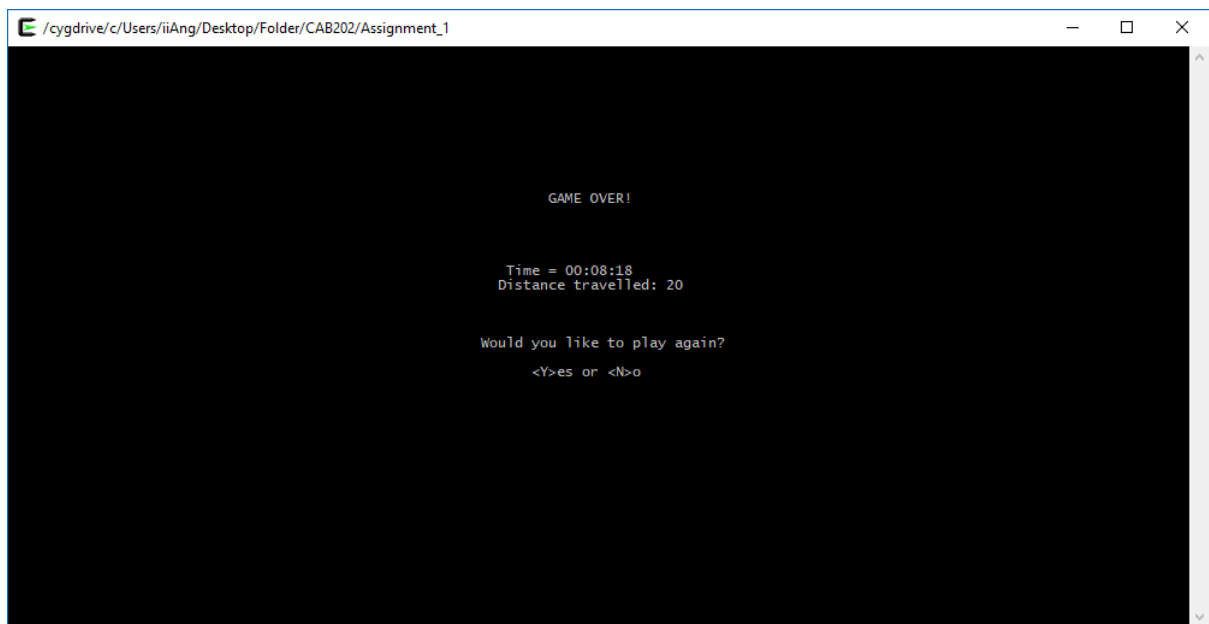
This function begins by declaring a local bool variable named valid\_input and it is set to false. The global variable is then set to true and the screen is cleared. A while loop is created and an if statement was added to make while the valid input is false and if the condition is zero then game over, or else the user wins. The function draw\_formatted() is used to write the text in specific positions which are determined using the functions screen\_width() and screen\_height(). The global variables time and distance travelled are used to display the finish elapsed time and distance. After the functions of draw\_formatted() are called, the function show\_screen() is called to display the text.

A local int variable called key is declared for the function get\_char() to wait for the user to press a key. Still in the while loop, if and else statements are created to determine if the key that the user pressed is valid. If the key that the user pressed is valid, then another if statement is made to check that if the key pressed is 'y' then make a new game, set the bool variable game\_over to false and reset all conditions. The function time\_reset() is called to reset the time. Condition is set to 100, current\_condition is set to "Excellent", current\_speed is set to 0, fuel\_remaining is set to 100 and distance\_travelled is set to 0. Main is called at the end of the if statement to restart the game.

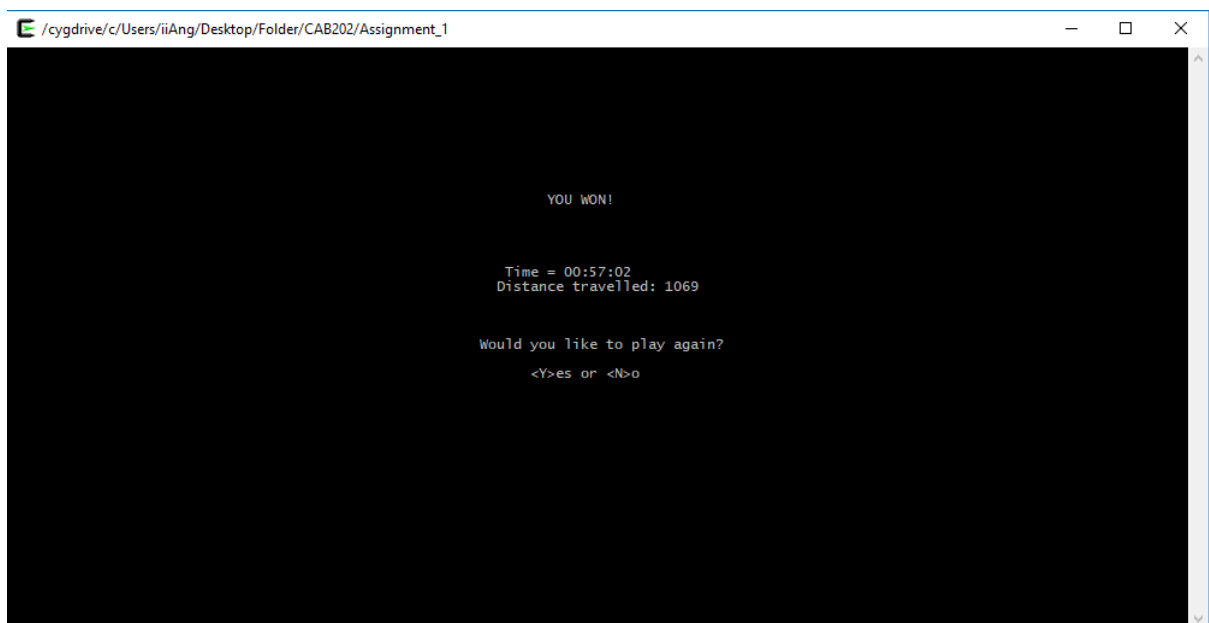
#### **void time\_reset (void) – Lines 970:973**

This function resets the time by setting the global variables milliseconds and seconds equal to zero.

## Test Plan



## Game over screen



## Victorious End Screen