Race To Zombie Mountain Game Report

Supported by ZDK, programmed in C language & compiled in Cygwin

Kevin Duong – n9934731

DUE DATE – 22 APRIL 2018

Executive Summary

This report was made to analyse the creation of a game called 'Race to Zombie Mountain', a file source written in C and compiled and played in Cygwin. Key findings involve testing the game to demonstrate and explain the features for each case, dictated by the screenshots captured at the time of the gameplay. These features build up the game with various conditions, controls, and objects that provide players a challenge and opportunity to play and win the game. Each feature is created with the use of functions and support from ZDK, 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 use of splash screens, setup of the border and dashboard, spawn implementation for scenery and zombies, and a functioning condition for health, speed, distance travelled, fuel, time, and collision. As such, the game works through understanding the concepts of the game 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

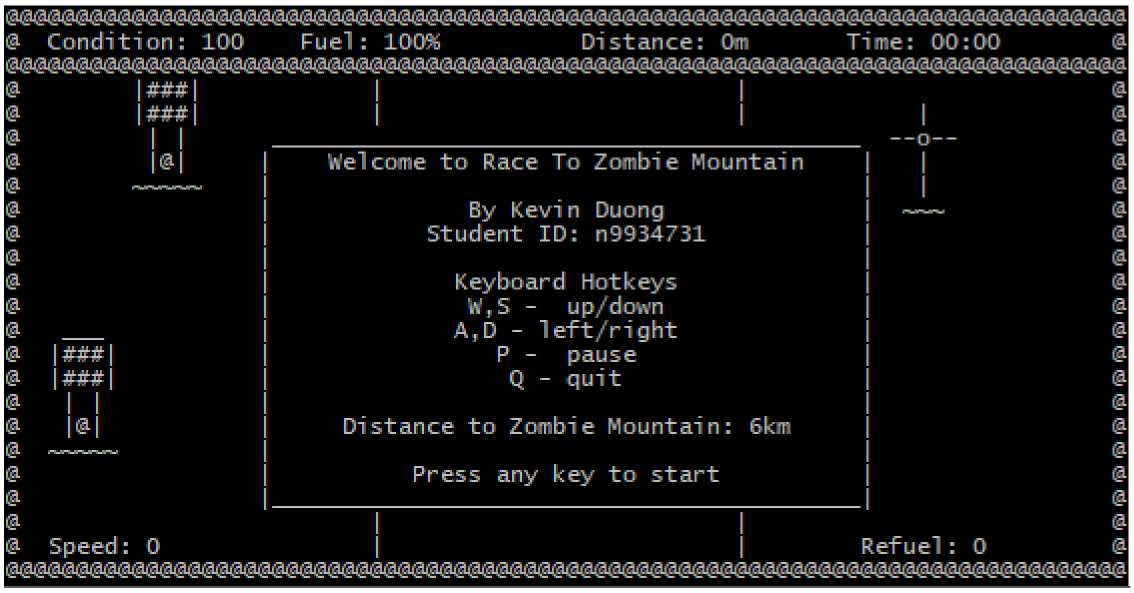
Introduction

‘Race To Zombie Mountain’ is a survival horror top-down scrolling view race-car game that puts you in a position where you must drive to ‘Zombie Mountain’ and survive against a hordes of zombies. The game is played on Cygwin, which is a Unix command line interface that can compile and run a C source file, in the essence that ‘Race to Zombie Mountain’ is a source file written in the C programming language. This report will outline the various features of the game, documenting each case of the gameplay; how it plays, how it works, and that of the methods implemented into making the game.

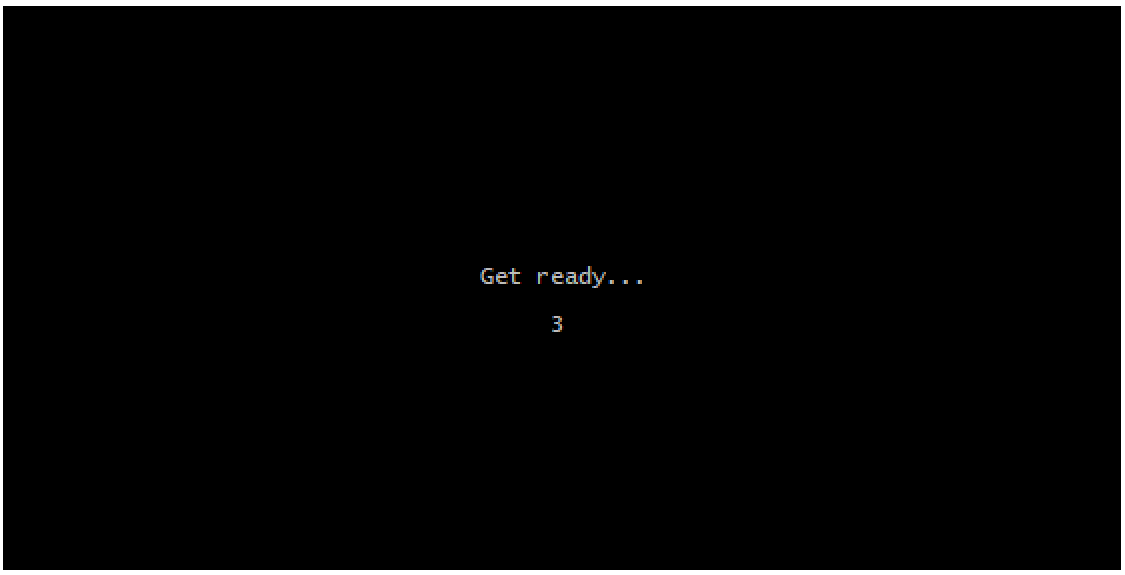
Part A – Basic Game

Splash Screen – Welcome

Players are introduced to welcome screen as soon as they execute the file on their program. The splash screen introduces the player with a title of the game, and information regarding to the game's controls and objective.



```
@  Condition: 100     Fuel: 100%          Distance: 0m        Time: 00:00

                   Welcome to Race To Zombie Mountain

                           By Kevin Duong
                        Student ID: n9934731

                          Keyboard Hotkeys
                           W,S -  up/down
                          A,D - left/right
                            P -  pause
                            Q - quit

                    Distance to Zombie Mountain: 6km

                         Press any key to start

@  Speed: 0                                                   Refuel: 0
```



```
Get ready...

     3
```

Figure 1 - The welcome screen of RTZM and Countdown.

Functions used:

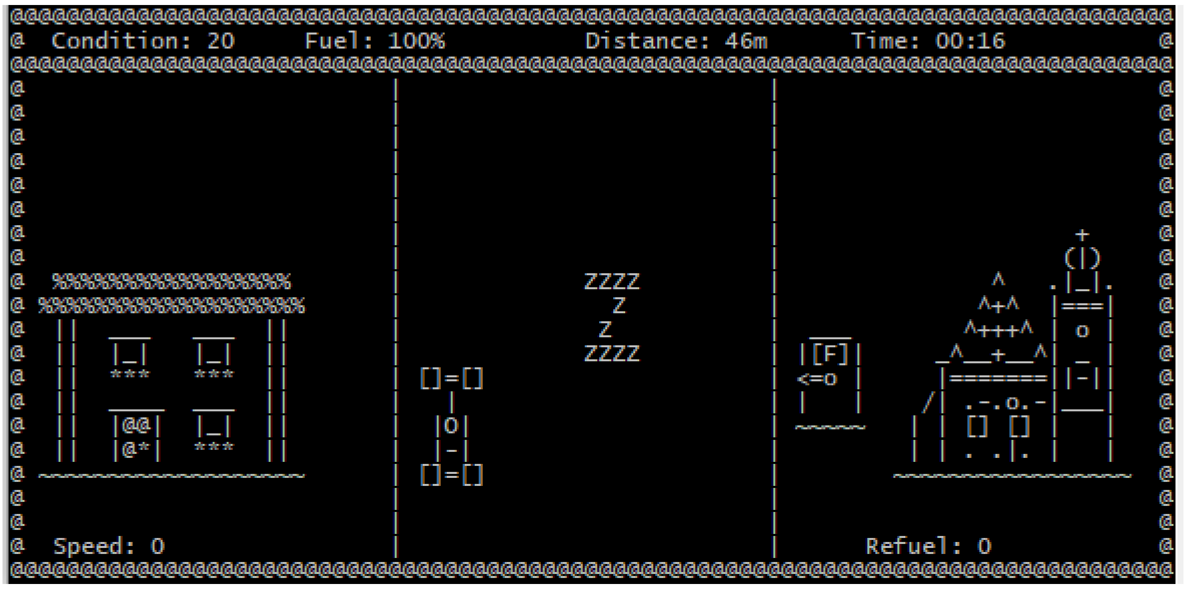
- void main_menu( ) *'a char display which contains information for splash screen'*
- void countdown() *'after a key is pressed, a countdown of 3 seconds begin'*

Boolean used:

- bool menu = true;

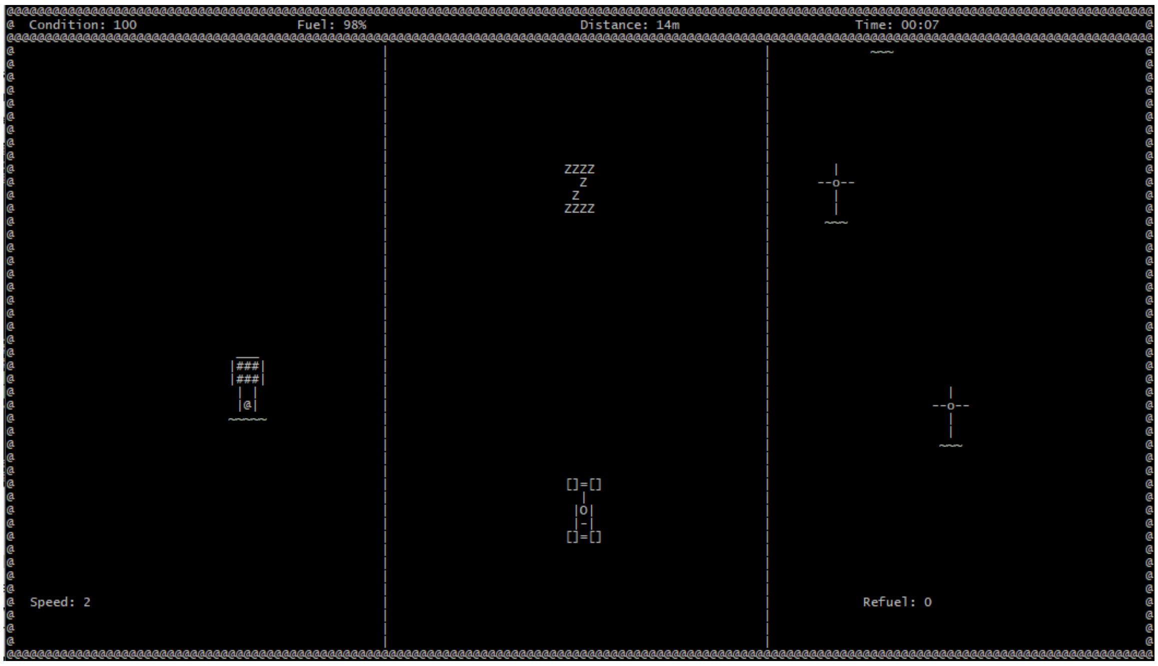
Game Border

The screen border is visible to keep the player within the game at all times. Its intent is to also hold the player's hud containing the information relevant to the game, without interfering the game or the player. The border is also responsive to any window size, regardless of the user's decision to play it on.



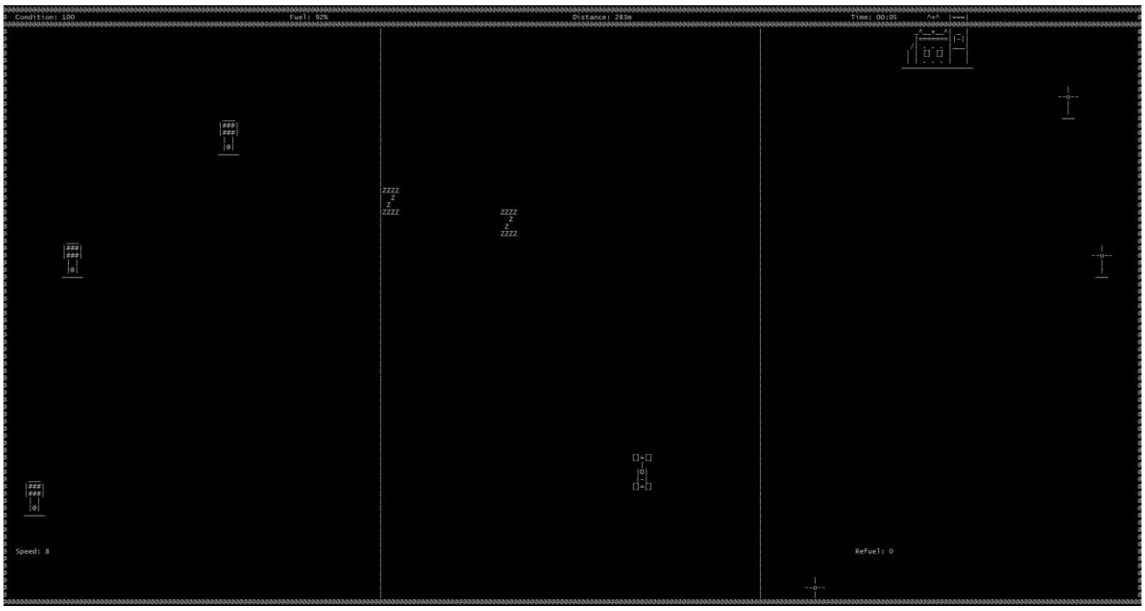
```
Condition: 20     Fuel: 100%          Distance: 46m       Time: 00:16
Speed: 0                                                  Refuel: 0
```

Figure 2 - Game window minimum size character of 80x24



```
Condition: 100     Fuel: 98%     Distance: 14m     Time: 00:07
Speed: 2                                           Refuel: 0
```

Figure 3 - Game window size character of 150x50



```
Condition: 100          Fuel: 92%          Distance: 283m          Time: 00:05
Speed: 8                                                           Refuel: 0
```

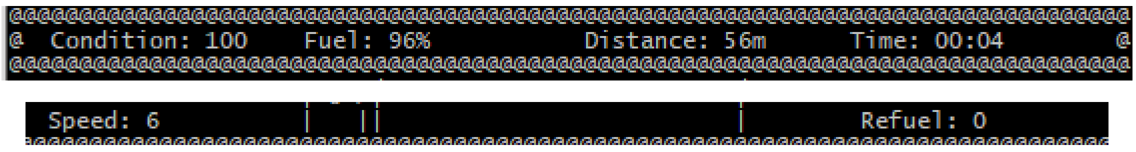
Figure 4 - Game window size character of 270x83

Functions used:

- void setup_map( )

Dashboard GUI

The player's hud (heads up display) which displays text information that is important to the player, that is the condition of the race car, the fuel the car has left, seconds to count refuel, distance travelled, speed of the car, and time. Speed and refuel count are displayed on the bottom of the screen.



```
@  Condition: 100    Fuel: 96%        Distance: 56m      Time: 00:04       @
 Speed: 6                                             Refuel: 0
```

Figure 5- Player's HUD displaying important information.

Functions used:

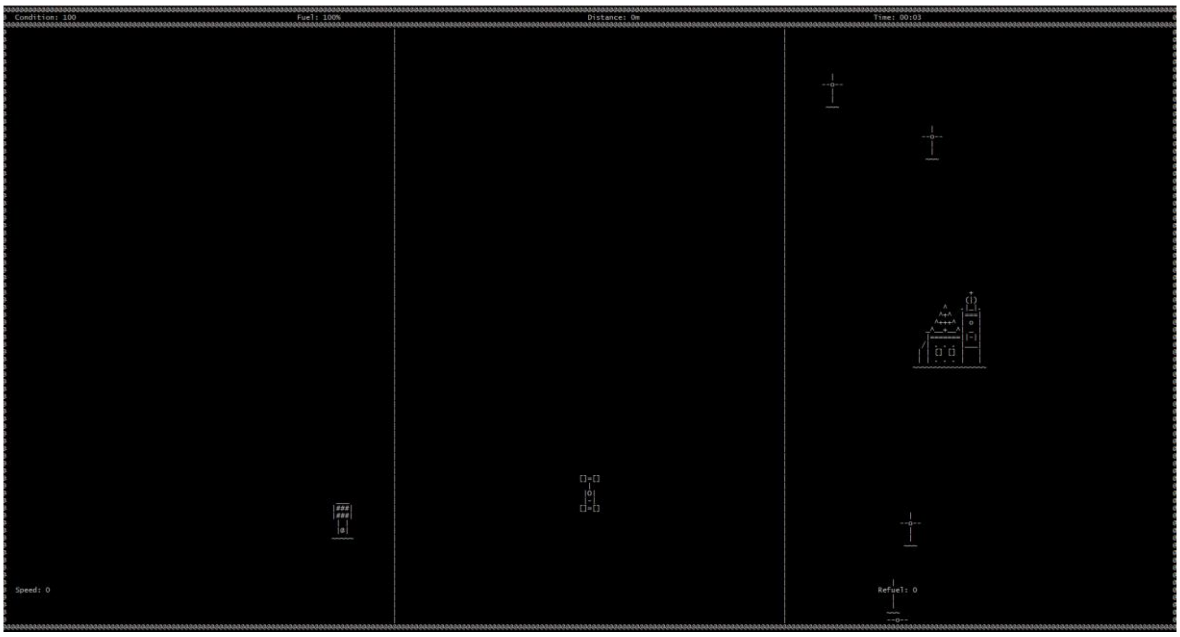
- void player_hud( ) '*Holds the global variables together to display in the HUD*'

Global Variables:

- int health = 100; *'Condition'*
- int fuel = 100;
- int distance = 0;
- time_t minutes = 0;
- time_t seconds = 0;
- int speed = 0;
- time_t refuel_timer = 0;

Race car, horizontal movement (non-collision)

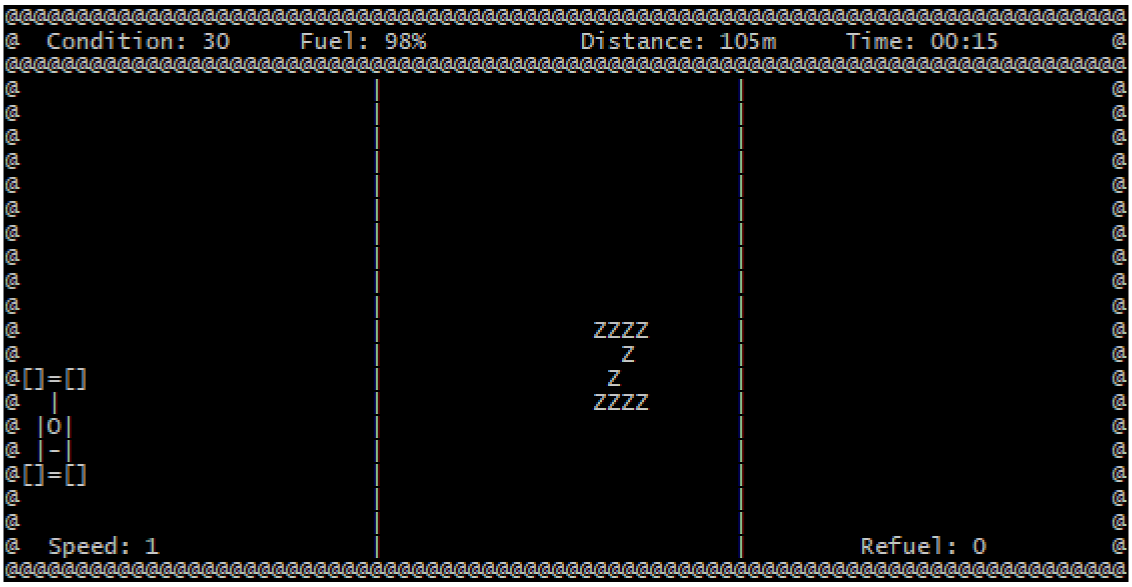
To reach Zombie Mountain in time, the player must commandeer a race car that can go up to extraordinary speeds and get to any destination quickly. The car is a 5x5 sprite creation using key letters and special characters to make the look and feel of the race car. As the game starts, the car is spawned on the lower middle of the road, and can only go left or right as this is a top-down scrolling view game. Keys are binded in terms of the car's horizontal movements; the player can go left or right pressing the key 'A' and 'D' respectively which is also shown on the splash screen to tell players the controls (Figure 1). However, once the car has achieved speed, only then the car is able to move horizontally across the game.



```
Condition: 100        Fuel: 100%        Distance: 0m        Time: 00:03

Speed: 0                                                    Refuel: 0
```

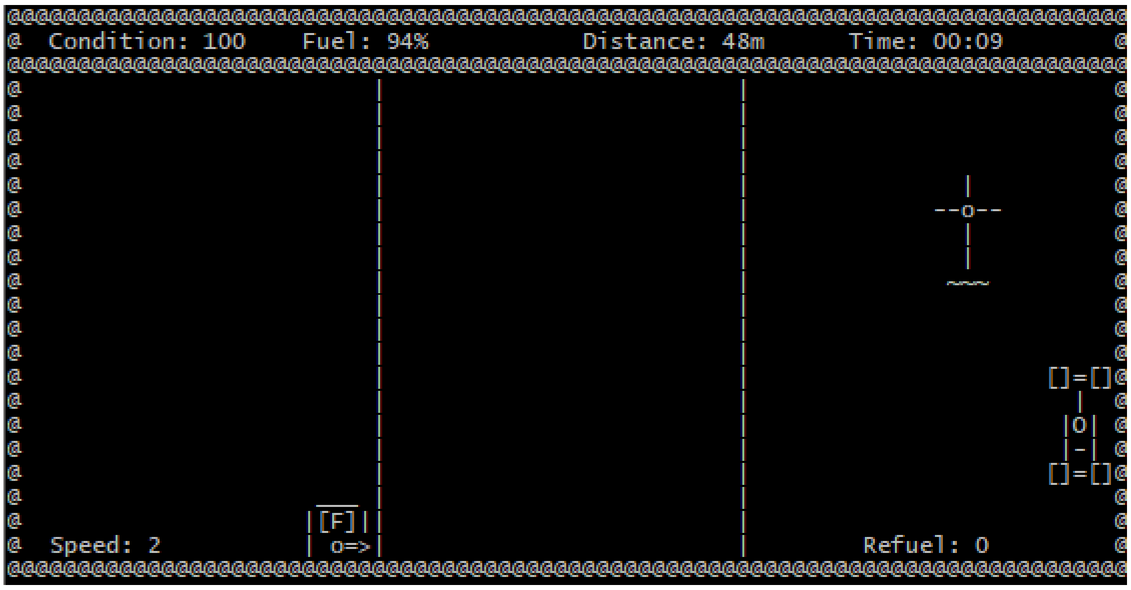
Figure 6 - Racecar initially start on the middle of the road at speed 0.

When the player steers too far left or far right on the screen border, it will stop and not proceed to enter the key pressed by the player. A condition is ruled out by the source code to say the car will only move when the width of the car is greater than the border (left), which is always true and vice-versa.



```
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@  Condition: 30     Fuel: 98%          Distance: 105m     Time: 00:15       @
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@                          |                         |                        @
@                          |                         |                        @
@                          |                         |                        @
@                          |                         |                        @
@                          |                         |                        @
@                          |                         |                        @
@                          |                         |                        @
@                          |                         |                        @
@                          |                         |                        @
@                          |                         |                        @
@                          |         ZZZZ            |                        @
@                          |           Z             |                        @
@[]=[]                     |          Z              |                        @
@  |                       |         ZZZZ            |                        @
@ |O|                      |                         |                        @
@ |-|                      |                         |                        @
@[]=[]                     |                         |                        @
@                          |                         |                        @
@                          |                         |                        @
@  Speed: 1                |                         |       Refuel: 0        @
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
```

Figure 7- The car leaning up against the left border, unable to overlap it.



```
@  Condition: 100     Fuel: 94%          Distance: 48m       Time: 00:09      @
@  Speed: 2                                                  Refuel: 0        @
```

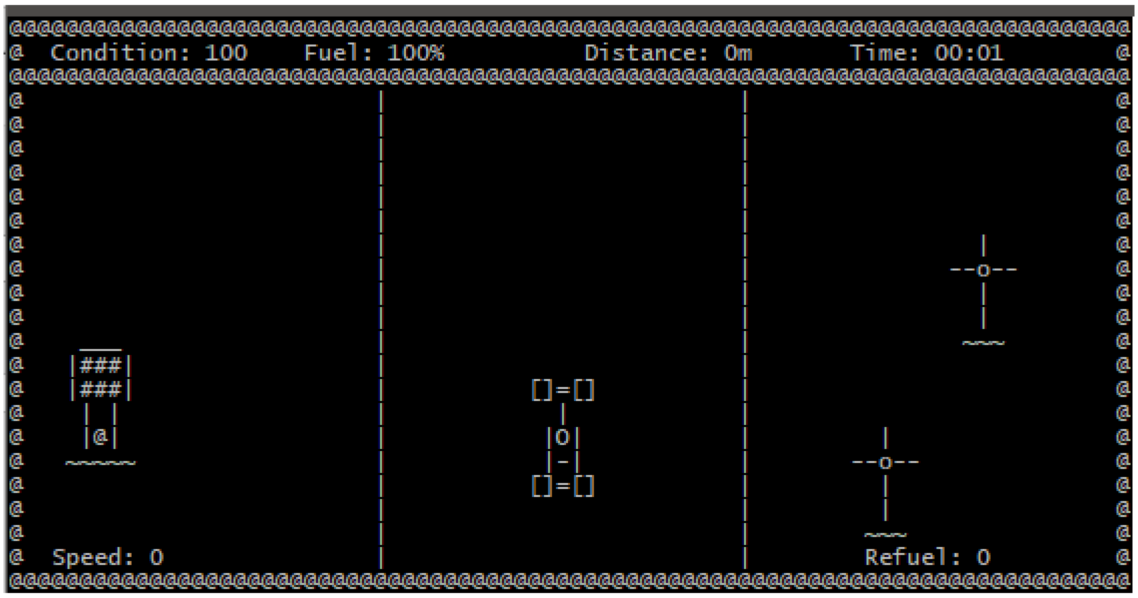
Figure 8- When car width is less than right border, it cannot go any further.

Functions used:

- void setup_game(void) ‘*Sprite player creation*’
- void player_controls(int key) ‘*Conditions for key controls*’

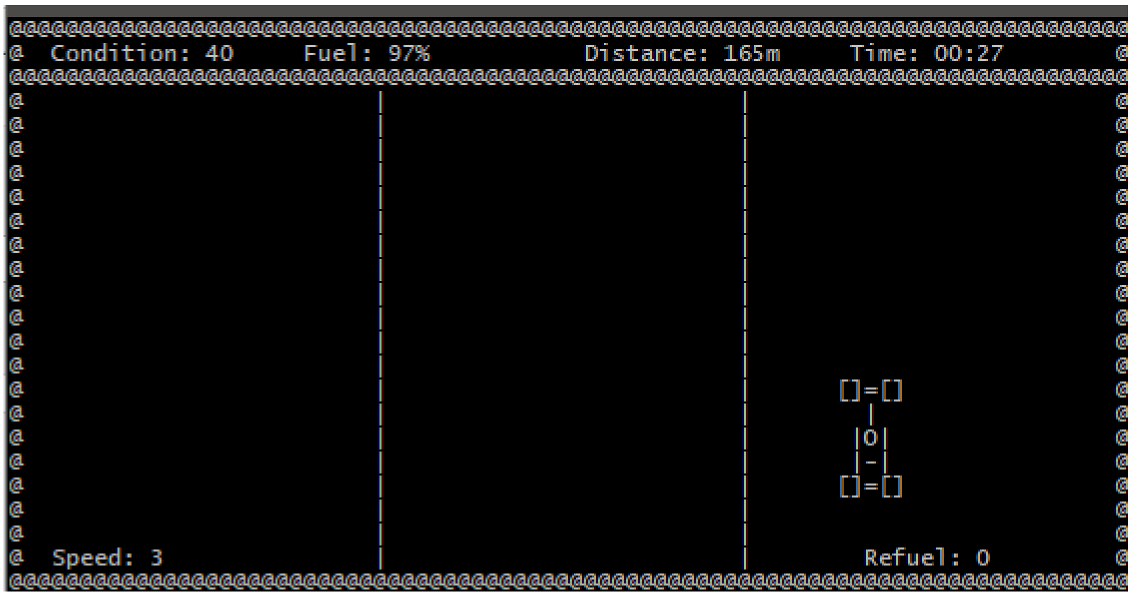
Acceleration and speed

To go faster, the player can accelerate the car pressing ‘W’ on their keyboard, which increases the speed and pressing ‘S’ to slow down the vehicle, decreasing the speed status as well. These pair of keys are also mentioned on the splash screen of the game (Figure 1). Having a speed range that starts from 0 – 10, the speed of the car can always be greater than or equal to zero, where 0 is stationary mode and 10 is extremely fast. Whenever the car is off-road, the speed can never exceed 3 and should it so the player must drive on the road again to increase its speed.



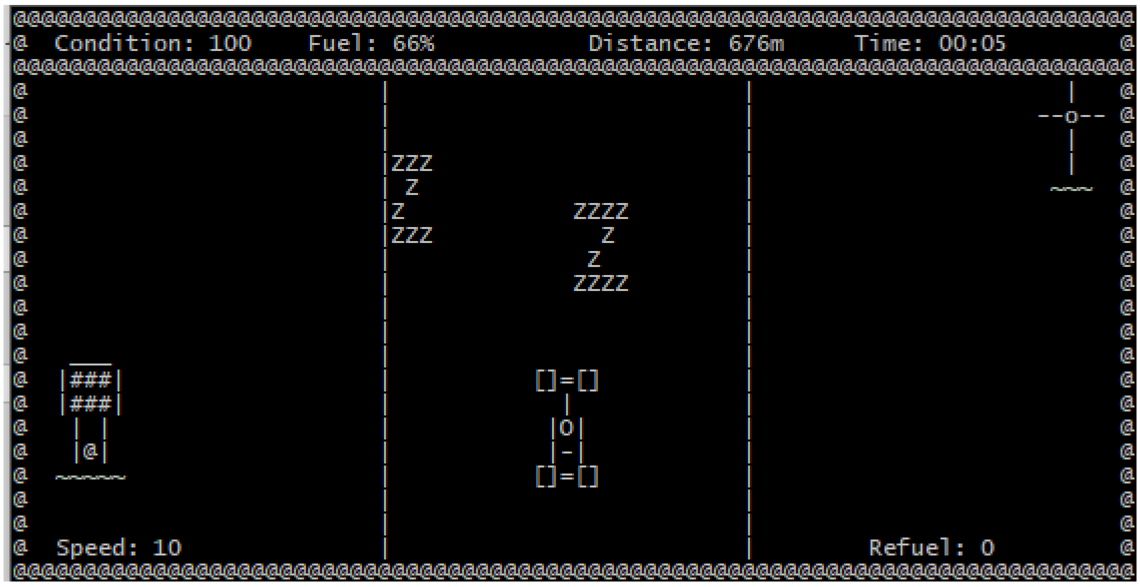
```
@  Condition: 100     Fuel: 100%         Distance: 0m        Time: 00:01      @
@  Speed: 0                                                  Refuel: 0        @
```

Figure 9 - Car in stationary mode indicated by the speed equalling to zero.



```
@  Condition: 40      Fuel: 97%          Distance: 165m      Time: 00:27
@  Speed: 3                                                   Refuel: 0
```

Figure 10 - Car speed exceeding at its limit of 3.



```
@  Condition: 100     Fuel: 66%          Distance: 676m      Time: 00:05
@  Speed: 10                                                  Refuel: 0
```

Figure 11 - Car reaching maximum speed of 10.

Functions used:

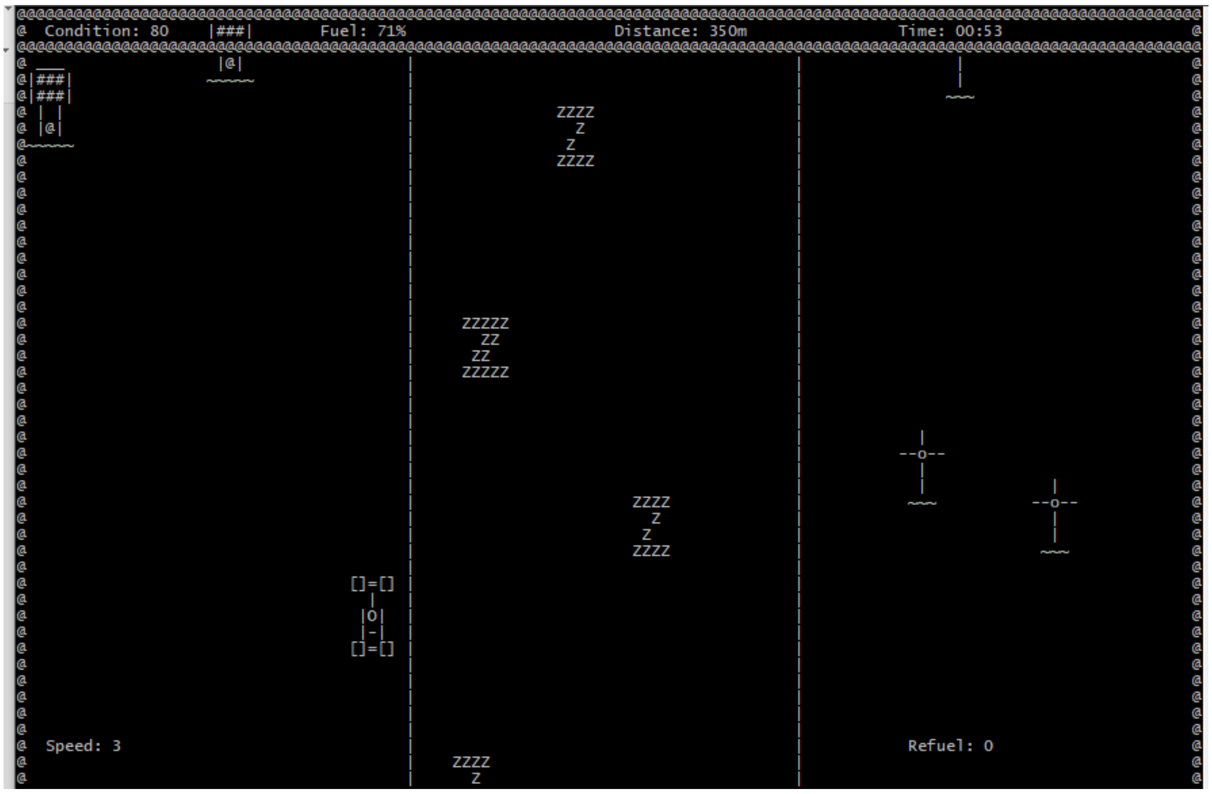
- void player_controls(int key) *'key w increases speed to 10 and key s decreases speed to 0'*
- void player_conditions( ) '*Speed set to 3 if car goes off road*'

Global variables:

- int speed = 0; '*Integer used in HUD will be affected by functions*'

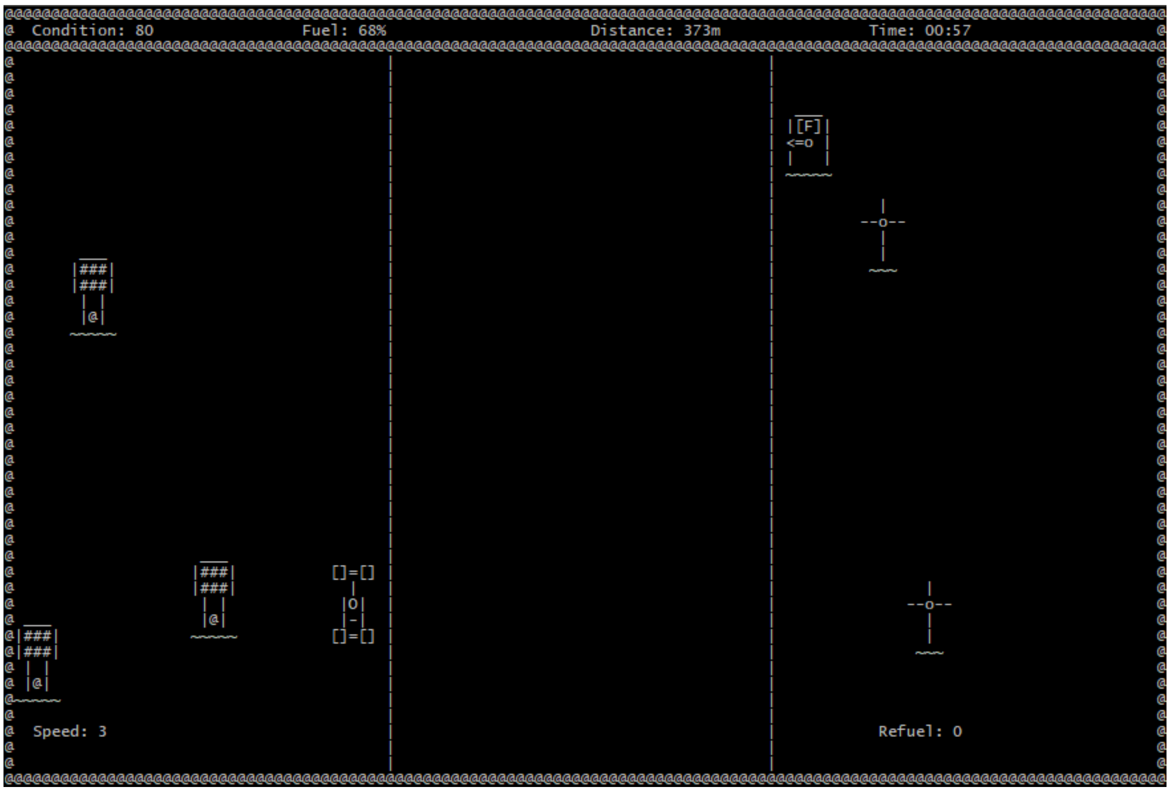
Scenery and obstacles

Throughout the game, the map provides scenery to fill in the void of the world, giving the player a fair challenge to deal with. Zombie Mountain sits on a land where a large grave site is on one side of the road, and a forest on the other side. These objects give players the opportunity to manoeuvre around the game ranging from trees, houses, crosses, churches, and survive against large mass of obstacles: zombies. They only travel down the road and not between it, coming down nonstop through the end of the bottom screen without overlapping. Other than this, the scenery also behaves in such a manner, only that they are stationary and do not move in magnitude speeds such as the zombies. In relation to speed, the faster the player goes, the faster the scenery moves down, providing the illusion of the race car going very fast at a top-down perspective. If the car doesn't move, neither will the scenery. Positioning of the objects and obstacles are randomised throughout the game, both horizontally and vertically.



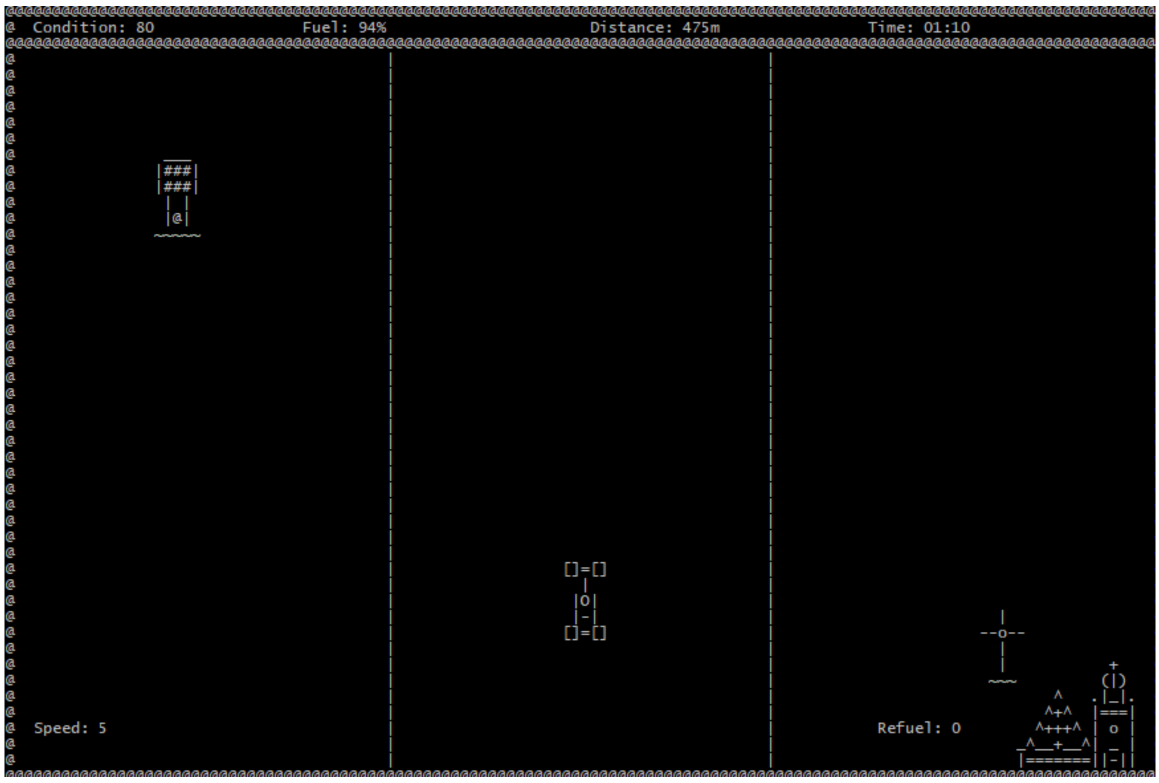
```
@  Condition: 80    |###|       Fuel: 71%                    Distance: 350m              Time: 00:53
@  Speed: 3                                                                     Refuel: 0
```

Figure 12- Objects scrolling in at top of window, moving down in motion relative to car's speed.



```
Condition: 80          Fuel: 68%              Distance: 373m          Time: 00:57

Speed: 3                                                              Refuel: 0
```

Figure 13 – Objects move down closely towards the bottom of the screen after a couple of seconds pass.



```
Condition: 80          Fuel: 94%              Distance: 475m          Time: 01:10

Speed: 5                                                              Refuel: 0
```

Figure 14 - A church scrolling out at bottom of window.

Functions used:

- sprite_id create_zombie_mountain(void)
- sprite_id create_zombieZM(void)
- sprite_id create_miniZombie(void)
- sprite_id create_tree(void)
- sprite_id create_house(void)
- sprite_id create_church(void)
- sprite_id create_cross(void)
- void scenery_stationary(sprite_id sprite) '*Sets sceneries to move based on speed*'
- void zombie_movement(sprite_id zombie) *'Settings for zombies to move at magnitude directions'*
- void object_movement() *'Setup movement for process '*
- void object_spawn() *'Setup spawn for process'*

Sprite ID declarations:

- sprite_id player;
- #define MAX_TREES (100)
- sprite_id tree[MAX_TREES];
- #define MAX_CROSS (100)
- sprite_id cross[MAX_CROSS];
- #define MAX_BUILDINGS (10)
- sprite_id house[MAX_BUILDINGS];
- sprite_id church[MAX_BUILDINGS];
- #define MAX_ZOMBIES (100)
- sprite_id zombieWave[MAX_ZOMBIES];
- #define MAX_ZM (4)
- sprite_id zombieWaveZM[MAX_ZM];
- sprite_id zombie_mountain;

Fuel depot

Fuel stations are positioned on the left and right side of the road, appearing randomly whenever the player is travelling just like sceneries. They provide the player a chance to refuel if the car is to run out of fuel, displaying at a size as large as the car.

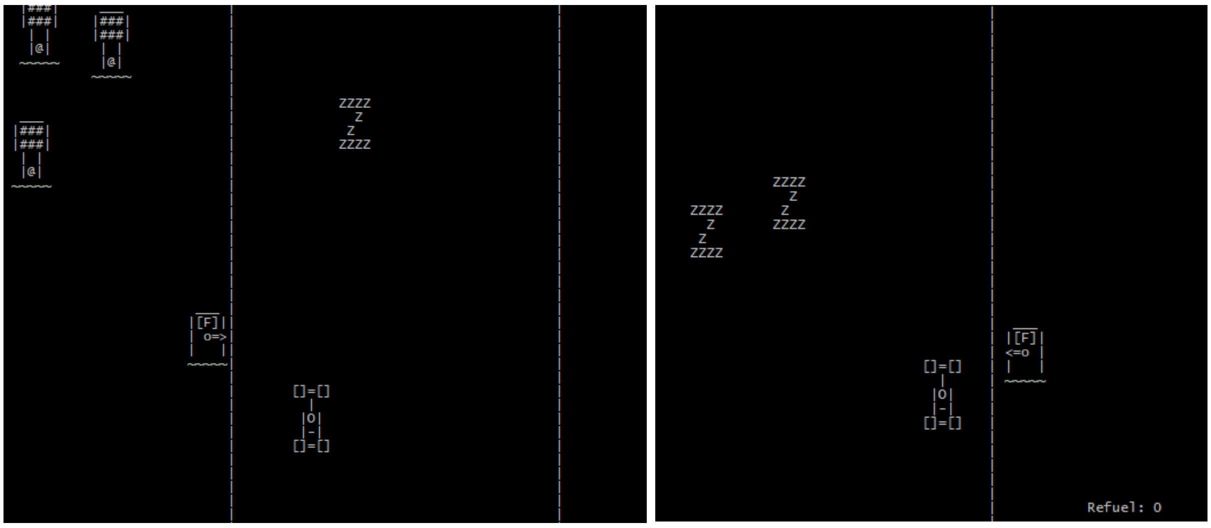


Figure 15 - Side-by-side screen capture of two fuel stations at both sides of the road.

Functions used:

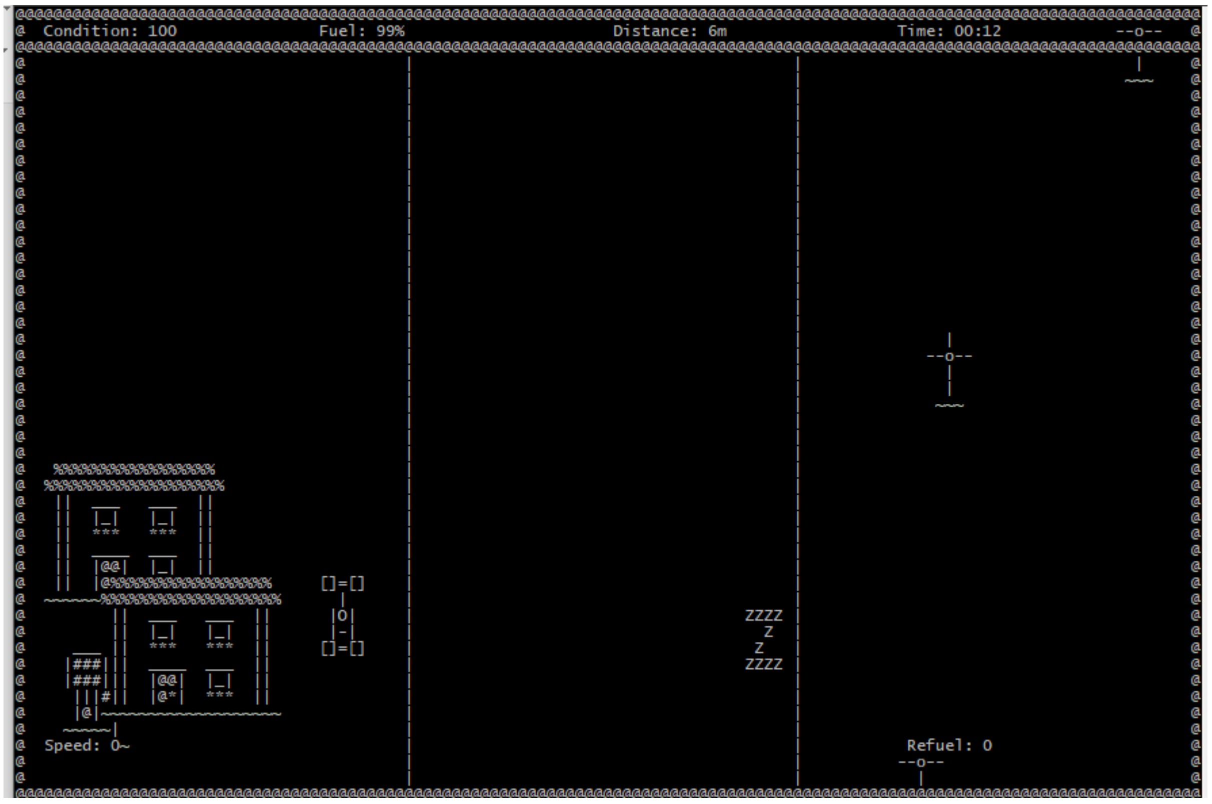
- sprite_id create_station_left(void)
- sprite_id create_station_right(void)
- void scenery_stationary(sprite_id sprite)
- void object_movement()
- void object_spawn()

Sprite ID declarations:

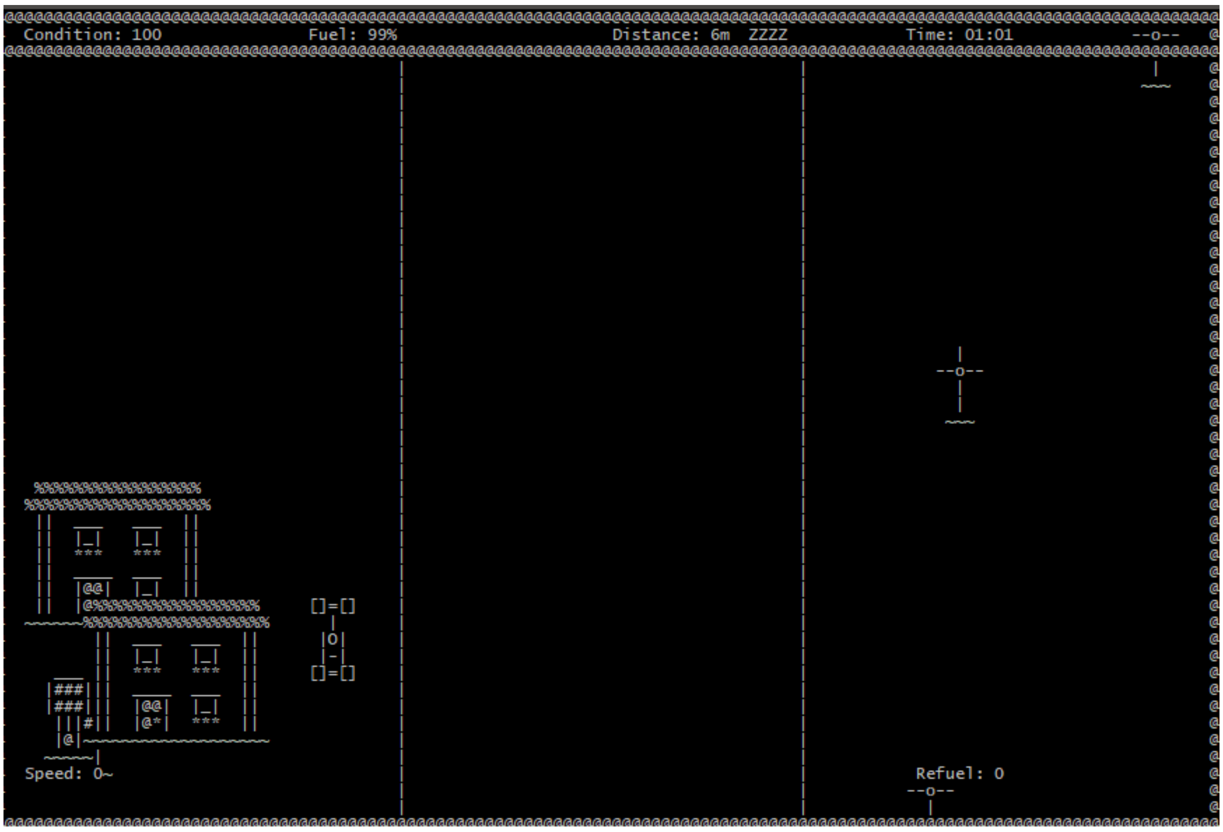
- #define MAX_FUEL (10)
- sprite_id fuel_depot_left[MAX_FUEL];
- sprite_id fuel_depot_right[MAX_FUEL];

Fuel

Indicated by the fuel percentage in the player's hud (Figure 5), the car starts off with a full fuel, which is consumed at a rate proportional to the speed of the car if the car begins to move. Importantly, if the car runs out of fuel, then the car will stop and the game will be over, given that this is a warning for the player to immediately find a fuel station to refuel whenever the race car is running low. To refuel, the car must align itself next to the fuel station, close enough that it doesn't collide it. The refuel timer will start, and the player won't be able to move for 3 seconds, which after that they are moved one step horizontally on the road with a full tank.

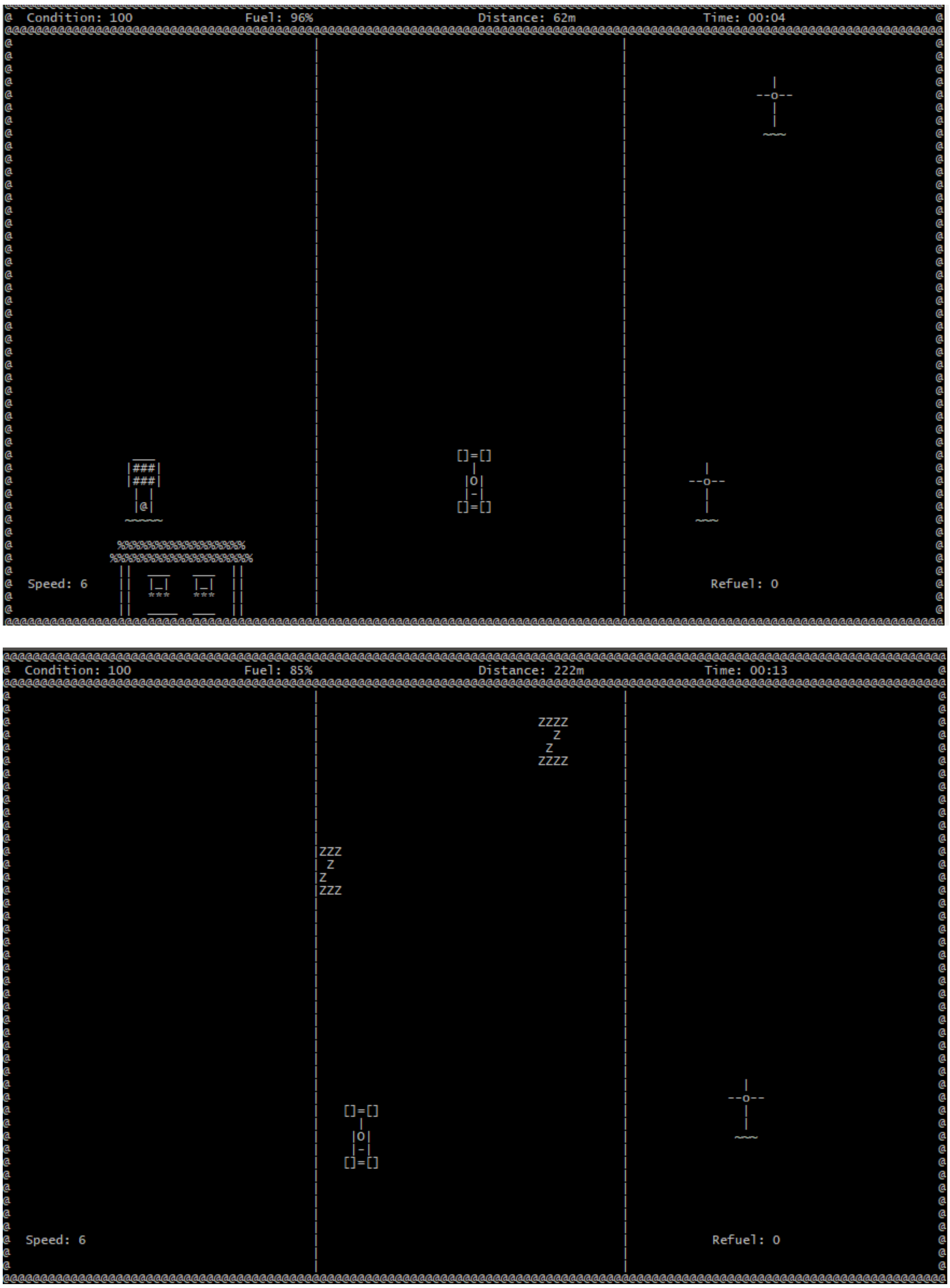


```
Condition: 100          Fuel: 99%          Distance: 6m          Time: 00:12
Speed: 0~
Refuel: 0
```



```
Condition: 100          Fuel: 99%          Distance: 6m  ZZZZ          Time: 01:01
Speed: 0~
Refuel: 0
```

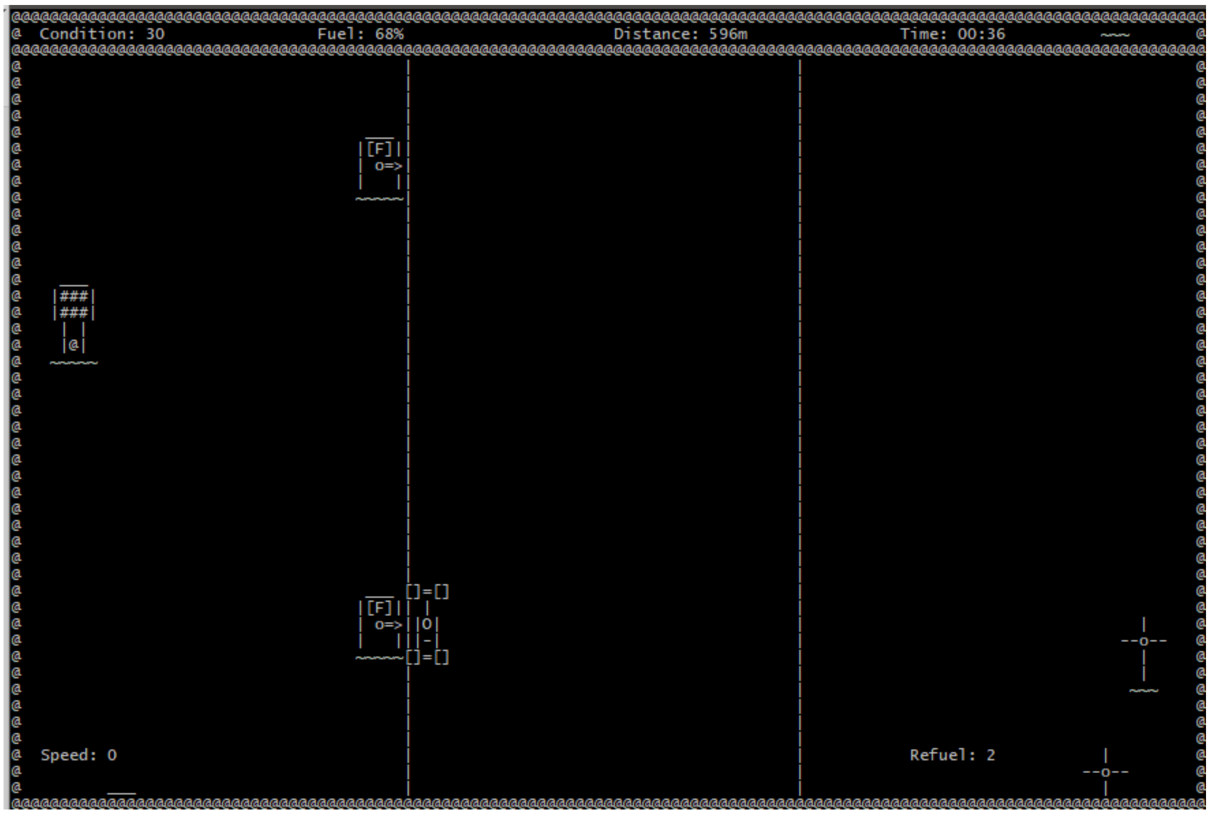
Figure 16 - Car fuel remains 99% for more than a minute while remaining stationary, never decreasing.



```
Condition: 100          Fuel: 96%          Distance: 62m          Time: 00:04

Speed: 6                                                          Refuel: 0
```

```
Condition: 100          Fuel: 85%          Distance: 222m         Time: 00:13

Speed: 6                                                          Refuel: 0
```

Figure 17 - Car fuel amount decreasing based off consumption that is proportional to speed of 6.



```
Condition: 30          Fuel: 68%          Distance: 596m          Time: 00:36

Speed: 0                                                  Refuel: 2
```

Figure 18 - Car adjacent to fuel station, refuelling at a duration of 2 seconds.

Functions used:

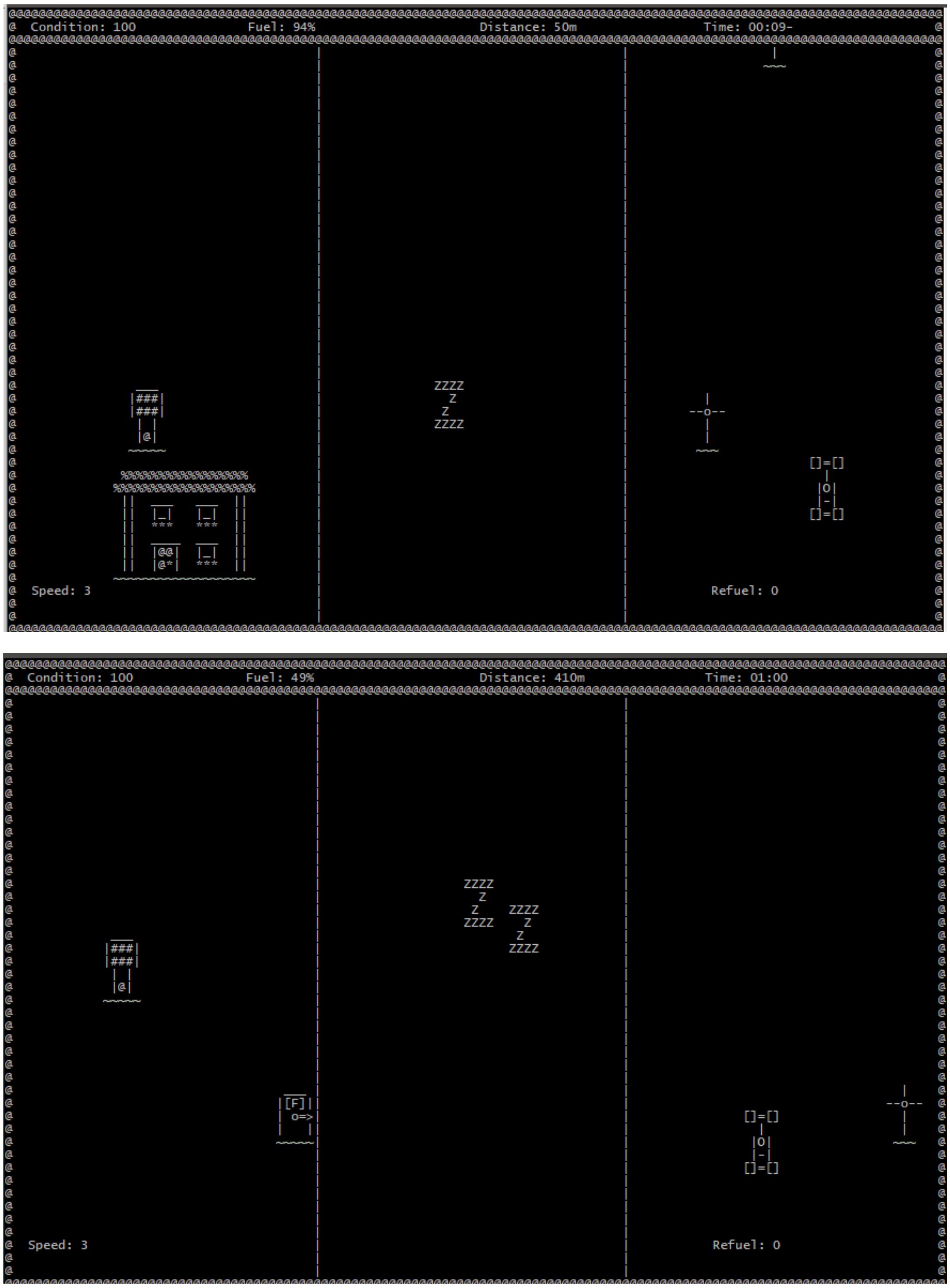
- void refuel_function () *'Function to handle global variables. The player's speed will be 0 and refuel timer will count up to 3 seconds, then fuel will fill back up to 100 and the player is positioned 1 step away from the depot horizontally.'*
- bool refuel_left (sprite_id player, sprite_id fuel_depot_left)
- bool refuel_right (sprite_id player, sprite_id fuel_depot_right) *'Collision boolean for two sprites – the player has to be adjacent to the fuel depot on the road to activate the refuel function'*
- void player_conditions () *'Setup condition for if the player collides adjacent to the fuel depot'*
- void fuel_consumption()

Global Variables:

- int speed = 0; '*car parked for refuel*'
- time_t refuel_timer = 0; *'refuel timer set for 3 seconds to refuel car gas'*
- time_t refuel_count = 0; *'Accumulates to a certain number to bring increment refuel timer'*

Distance Travelled

To determine how far the player must go to reach the end of the game, distance is accumulated on the player's hud, and much of the accumulation of distance acquired depends on the rate of speed. Initially starting at zero, if the player reaches 6000m in the game, then the player will able to see Zombie Mountain coming down from the top window. The game finishes once the car collides with the mountain.



```
Condition: 100          Fuel: 94%          Distance: 50m          Time: 00:09-
Speed: 3                                                          Refuel: 0
```

```
Condition: 100          Fuel: 49%          Distance: 410m         Time: 01:00
Speed: 3                                                          Refuel: 0
```

Figure 19 - Car accumulated 410m for 1 minute at speed 3. See Figure 16 for car not moving and thus distance isn't changed.

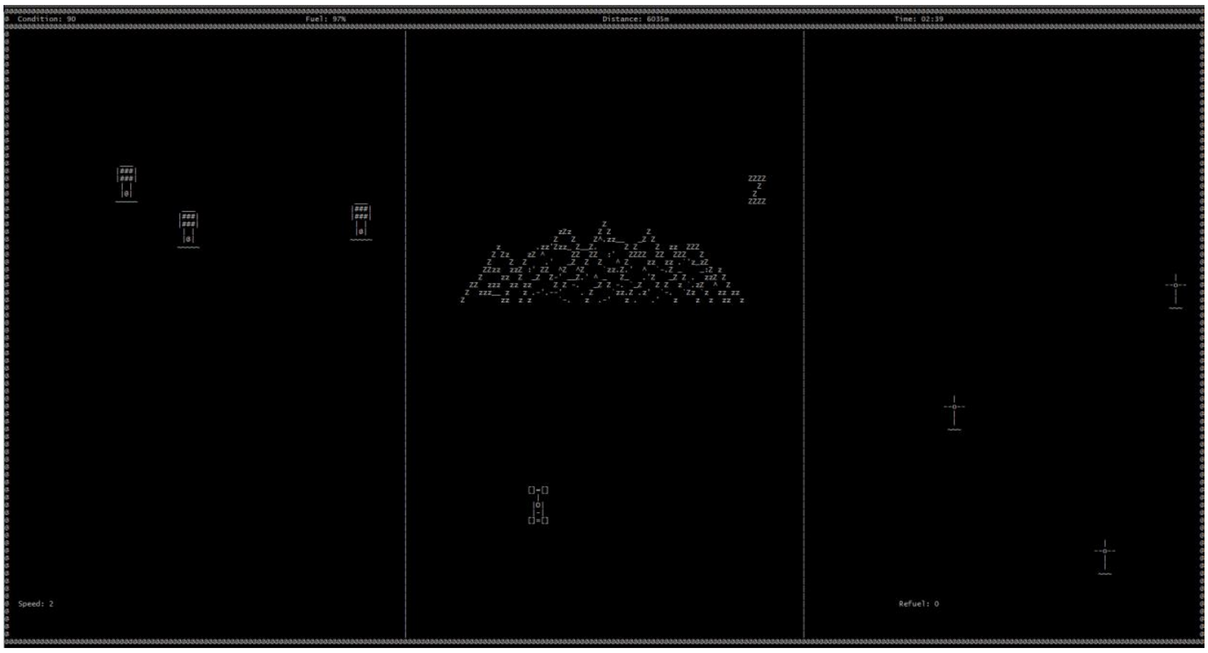


Figure 20 – After reaching 6km, the player will reach their destination. The Zombie Mountain will show up.

Functions used:

- void distance_traveled() *'Handles the increase of distance based on speed'*
- void player_conditions () *'If the player didn't collide zombie mountain and reaches a distance of 6500, then the player still wins.'*
- void process(void) *'If distance is 6000, zombie mountain will spawn'*

Global Variables:

- int distance = 0; *'The integer used to record how much distance is travelled'*
- int speed = 0; *''Speed determines how much distance will be accumulated"*
- time_t distance_count = 0; *'Accumulates the distance in cm but hidden in game'*

Collision

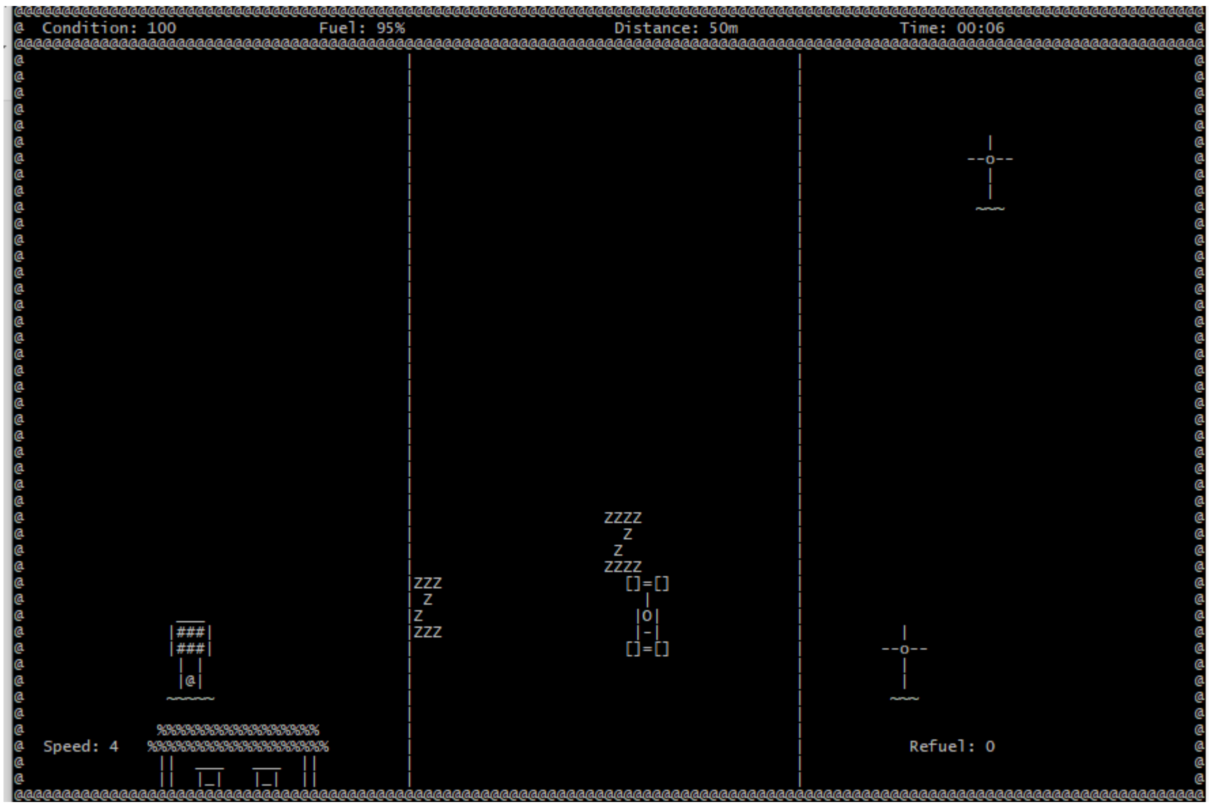
The game provides collision detection, which can detect when the player hits an object or scenery in the game. If the player collides any scenery/zombie other than fuel depots whether it's a head-on collision or side collision, then damage is inflicted upon the car and the player is spawned else where away from the incident, regaining full fuel and losing 10 health. However, if the car crashes into a fuel depot, then the player is instantly killed and the game is over. If the player collides with the Zombie Mountain, then the player will win the game.

Functions used:

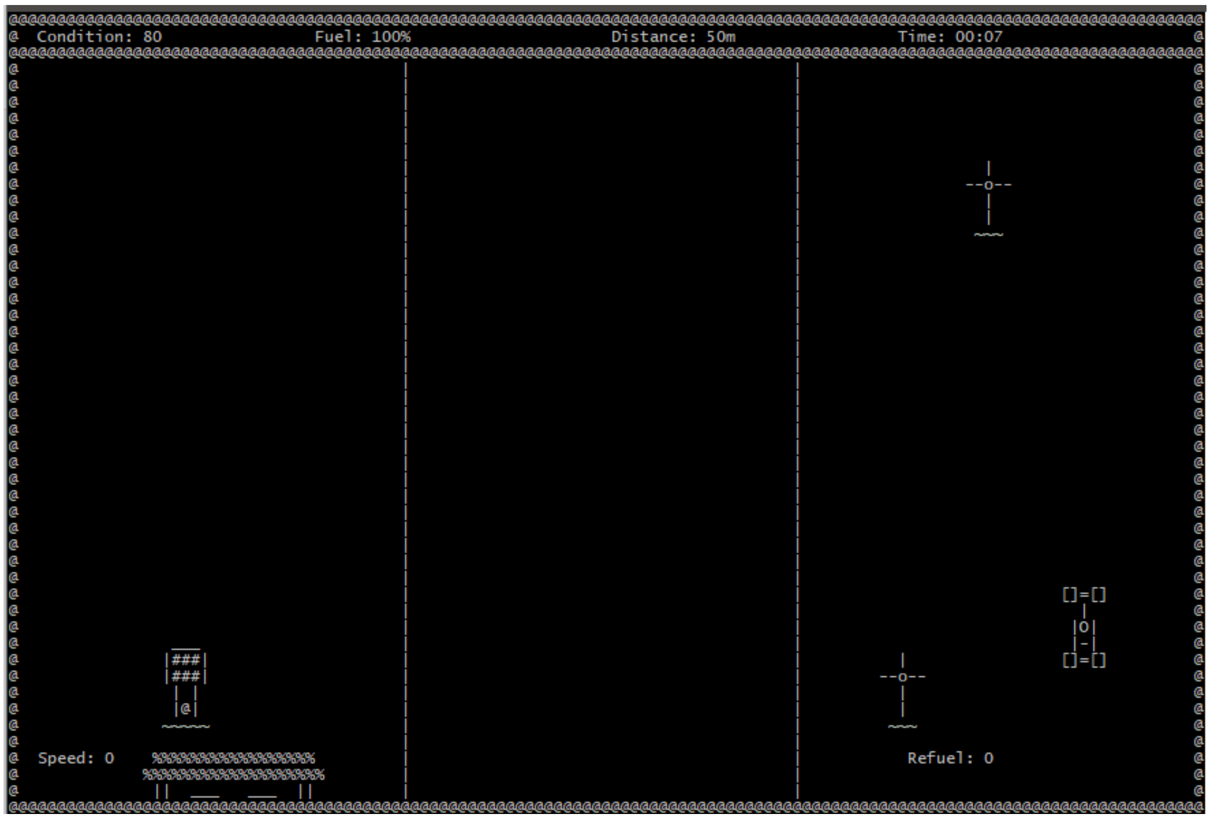
- bool collided (sprite_id player, sprite_id sprite)
- bool refuel_left (sprite_id player, sprite_id fuel_depot_left)
- bool refuel_right (sprite_id player, sprite_id fuel_depot_right)
- void damage_function () *'Manipulates the variables based on collision'*
- void player_conditions ()

Global Variables:

- int fuel = 100;
- time_t fuel_count = 0; *‘These two global variables reset when car is damaged’*



```
Condition: 100        Fuel: 95%        Distance: 50m        Time: 00:06
Speed: 4                                                    Refuel: 0
```

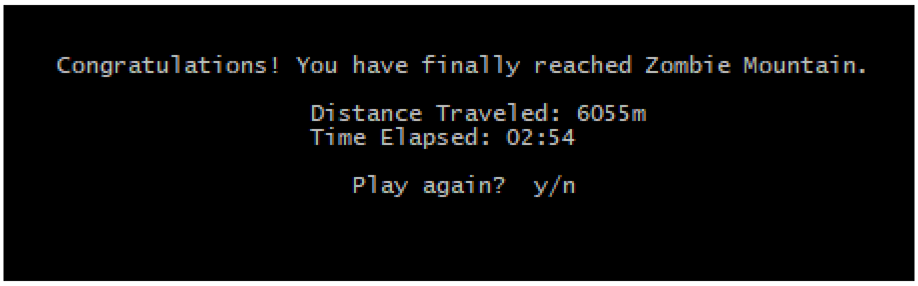


```
Condition: 80         Fuel: 100%       Distance: 50m        Time: 00:07
Speed: 0                                                    Refuel: 0
```

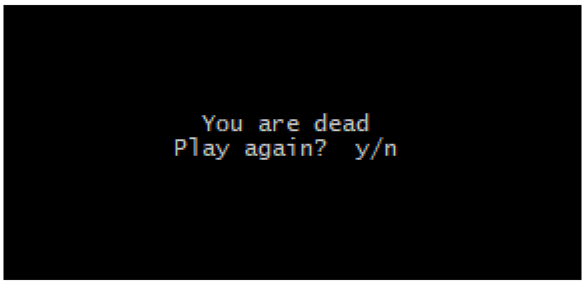
Figure 21 - The player head-on approaches the zombie at an adjacent level. After it collides, the car respawns at a random point.

Game Over Dialogue

As the player progresses through the level, they will encounter the Zombie Mountain indicating that they have reached the end of the road, finishing the game. A new screen will show up, telling the player that they have won the game, along with their elapsed time and the distance that they have travelled to get to Zombie Mountain. If the player wants to play again, they must press the ‘y’ key to restart the game shown in the figures below, which also applies when the player also loses. The game will reset, and the game can be played once again.



```
Congratulations! You have finally reached Zombie Mountain.

                 Distance Traveled: 6055m
                 Time Elapsed: 02:54

                    Play again?  y/n
```



```
  You are dead
Play again?  y/n
```

Figure 22 - When the game is finished / game over.

Functions used:

- void clock() ‘*Tells the player how long the game went*’
- void game_reset(int key) *‘If player retries, the game restarts, otherwise the game ends’*
- void display_gameover() *‘When the player loses’*
- void display_finish() ‘*a blank screen indicating the player won*’
- void player_conditions () ‘*Triggers the winning function if the player completes the objective”*

Booleans used:

- bool game_over = false;
- bool retry = true;

Pause and single step (feature)

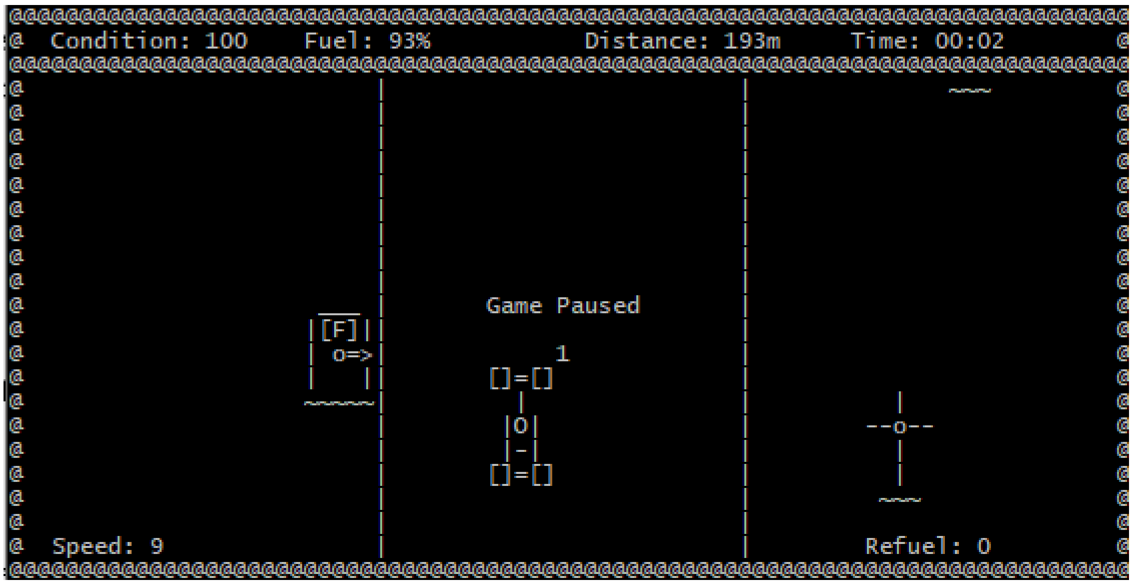
The player can pause the game by pressing the ‘p’ key. In doing so, the text will display a string called ‘Game Paused’, indicating that the game is waiting for the player to press a key to resume the game. After a key is pressed, the game prepares the player with a countdown of 3 seconds before continuing.

Functions used:

- void pause_mode () ‘*pauses the game*’
- void testCapture () *‘For clear screenshot captures’*

Boolean used:

- bool pause = false; ‘*Is true when pause key is pressed and vice versa*’
- bool pauseTest = false;



```
@  Condition: 100    Fuel: 93%           Distance: 193m     Time: 00:02
Game Paused
1
@  Speed: 9                                                Refuel: 0
```

Figure 23 - Game paused, ready to play after 1 second

Part B – Advanced Game

Extensions of the Basic Game

As the player progresses, the player will encounter a fleet of 100 zombies coming from Zombie Mountain, which move freely on the road unlike the sceneries. They often appear in groups, making it difficult for the player to move freely on the road at great speeds. Upon approaching Zombie Mountain, three super zombies will arrive to take out the player – they are bigger, faster, and can instantly kill the player.

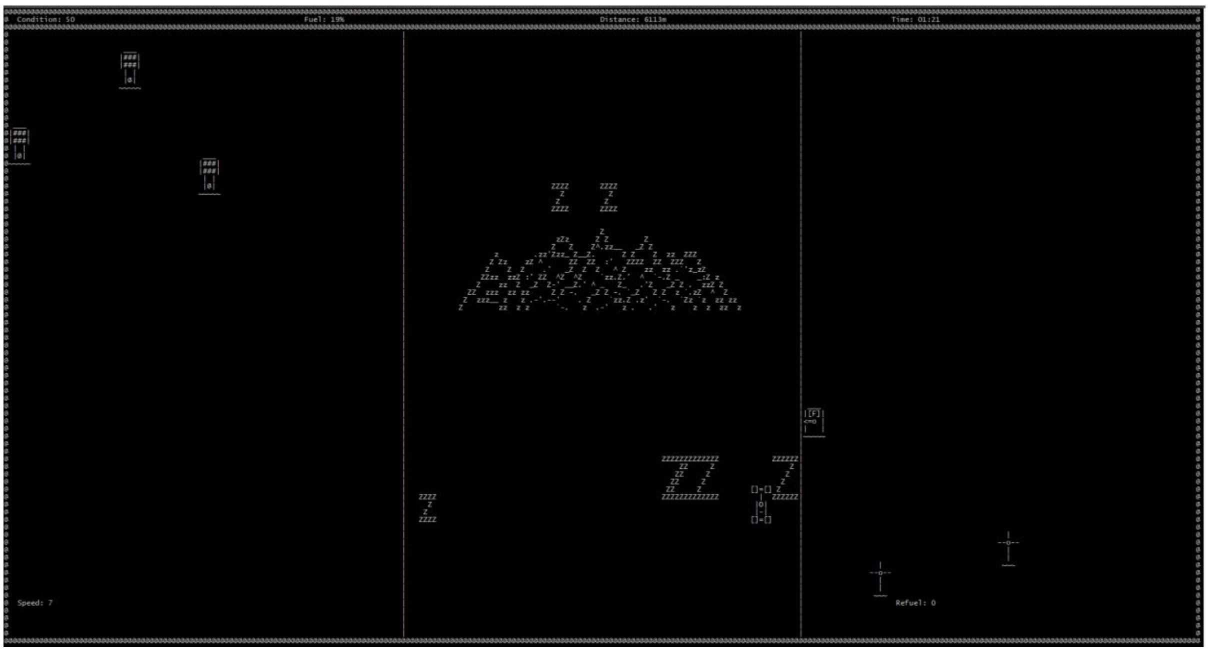


Figure 24 - Three super zombie mountain guardians

Functions used:

- sprite_id create_zombieZM(void)
- void object_movement()
- void object_spawn()
- void player_conditions () *'If collided, the player is instantly killed'*

Global variable efficiency

Use of arrays

Function task decomposition

Conclusion

Race To Zombie Mountain comprises features and abilities which makes the program at a retrospective view, an example of an arcade game featuring simple 2D graphics like that of ‘Pong’ and ‘Space Invaders’ by using a top-down scrolling view as the game’s main theme. The documentation of the game and capturing screenshots of the game concludes features such as a welcoming screen/game over, different sprite models for scenery, zombie AI, and the player. A collision detection which can cause damage to the player, a dashboard which gives the player information regarding about health, speed, distance, and time – all which function throughout the game and creates a big difference and much more.   Aside from this, each function written in the file’s source code provide the necessary features which make the game meant to be played and at the same time enjoyable.