

Table of Contents

Requirements Specification

Abstract	2
System Description	2
External view of the system	3 to 7
Functional Behavior	7
Environmental Considerations and Requirements	8

Design and Design Documentation

Abstract	9
System Description	9
Functional Decomposition	10
CRC Cards	11 to 15
Classes and Functions	
Classes	15 to 17
Functions	17 to 18
UML Class Diagram	19 to 20

Test Plan	21
-----------	----

Test Cases	22 to 38
------------	----------

Requirement Specification

Abstract

This simulation is designed to measure an average length of time and amount of fixed fuel units given on vehicle. Vehicle is travelled from point A to point B without refueling, where destination point B locates on a diagonal grid on the other end. Vehicle moves across each block at a time, it can encounter any object such as dead-ends, blockades by traffic accident, traffic parade vehicle and people, traffic bicycle race and etc.. Vehicle is equipped with navigation system that allows user to see any obstacle within the defined roads. User can utilize on board information systems to navigate through the maze.

System Description

A vehicle is a class of automobile consists of car, truck, van and SUV; each has similar and different functions such as a limit of fuel unit capacity and number of weapons that allow user to fire within limit to merge the vehicle through traffic. Similarities can be the electronic devices like the GPS, the system monitor and other safety features. There are also EPA gas mileage guidelines on each vehicle.

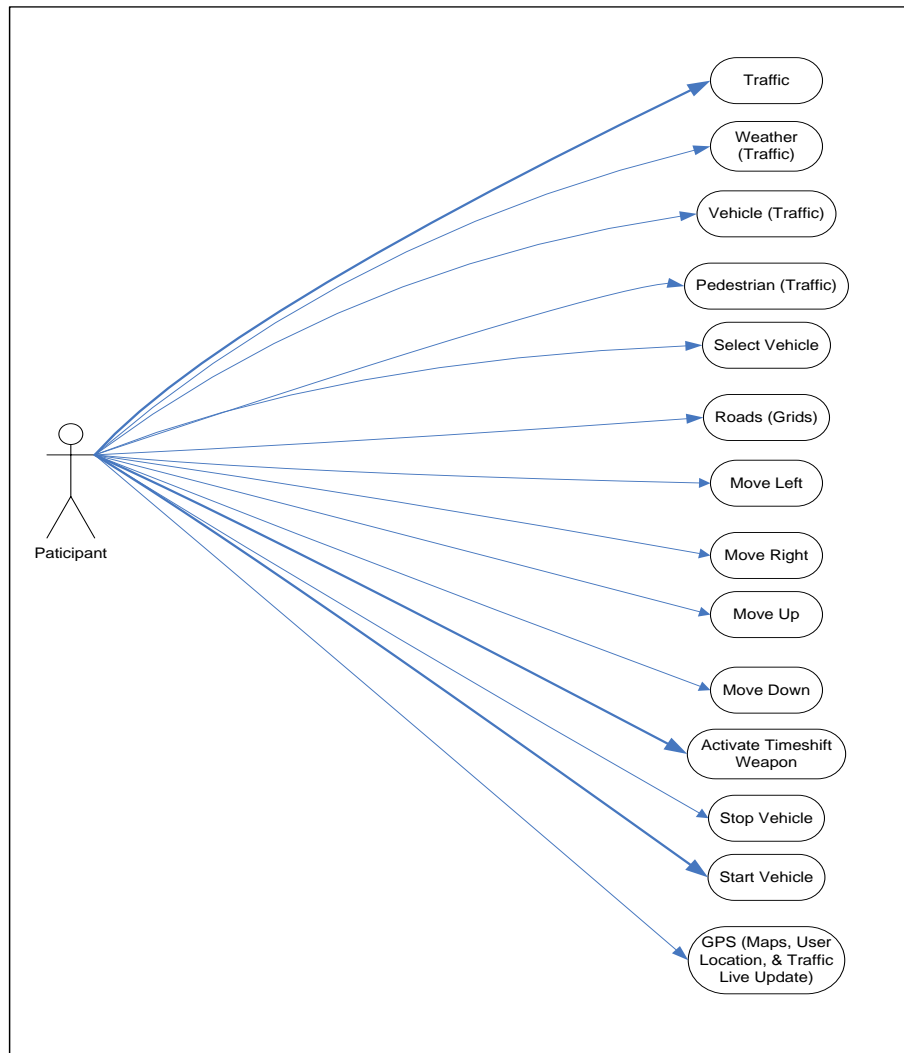
A vehicle can except basic commands such as move up, move down, turn left, turn right, display map, check traffic update surrounding the areas of the block and the vehicle current coordinate. A vehicle also can except a command to manipulate time and space so that the vehicle can merge through any traffic without incident.

To simulate traffic scenes, system can create multiple scenarios of an execution of a command: Several scenarios based on the effects of various weather conditions such as icy road, flood, and landslide. Vehicle traffic are common on any road side. Pedestrian can come across at any time of the day and the crowd tends to get higher at peak hour before and after workday. Riots can lurk on any corner of street, causing all kinds of traffic jam, and a group of bicycle race can come from nowhere across on one direction.

Despite all these obstacles, the system must able to guide the vehicle to its destination.

External view of the system

Use Case Diagram



Use Case Textual Description

Traffic

Initializes traffic functions, crossing, bicycle race, icy road, flood, brush fire, parade vehicle, emergency vehicle, riot and accident to null.

Weather (Traffic)

Given a fixed coordinate x-axis and y-axis by GPS function, the weather traffic can create random generated numbers of treacherous road conditions with black ice on the road, flooded road, and landslide.

Exceptions:

Moving Vehicle (Traffic)

Given a fixed coordinate x-axis and y-axis by GPS function, the Moving Vehicle traffic can create random generated numbers of treacherous road conditions with scenes of bicycle race, parade vehicle and emergency vehicle.

Exceptions:

Pedestrian (Traffic)

Given a fixed coordinate x-axis and y-axis by GPS function, Pedestrian traffic can create random generated numbers of treacherous road conditions with scenes of pedestrians crossing road, riots, and accident.

Exceptions:

Select Vehicle

Initializes Start Engine, Stop Engine, Base Fuel, Base Timeshife (weapon), One Unit Count, Two Unit Count, Three Unit Count, Timeshift Count and Base EPA gas mileage. Initializes functions, Start, Stop, Turn Right, Turn Left, Move Down, Move Up, Launch Timeshift weapon.

Exceptions:

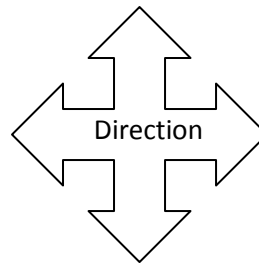
- Vehicle has limited units of fuel

- Vehicle has limited units of weapons that can be fired.

Roads (grids)

Creates data storage can be access anyway. Attributes are integer types included, left block, right block, up block, down block, one way up, one way down, one way left, one right, and school zone Function Initializing these integers to either 1 or 0.

(0,0)	(0,1)	(0,2)	(0,3)	(0,4)
(1,0)	(1,1)	(1,2)	(1,3)	(1,4)
(2,0)	(2,1)	(2,2)	(2,3)	(2,4)
(3,0)	(3,1)	(3,2)	(3,3)	(3,4)
(4,0)	(4,1)	(4,2)	(4,3)	(4,4)



Exceptions:

Move Left

This command allows user to move an object one grid to left direction or westbound road.

Exceptions:

User cannot move an object bypassing another object that is being occupied on the grid, onto a dead-end road. User cannot move an object beyond the boundaries that defined by 5x5 grids. User cannot initiate the command if insufficient fuel is available.

Move Right

This command allows user to move an object one grid to right direction or eastbound road.

Exceptions:

User cannot move an object bypassing another object that is being occupied on the grid, onto a dead-end road. User cannot move an object beyond the boundaries that defined by 5x5 grids. User cannot initiate the command if insufficient fuel is available.

Move Up

This command allows user to move an object one grid to up direction or southbound road.

Exceptions:

User cannot move an object bypassing another object that is being occupied on the grid, onto a dead-end road. User cannot move an object beyond the boundaries that defined by 5x5 grids. User can not initiate the command if insufficient fuel is available.

Move Down

This command allows user to move an object one grid to up direction or northbound road.

Exceptions:

User cannot move an object bypassing another object that is being occupied on the grid, onto a dead-end road. User cannot move an object beyond the boundaries that defined by 5x5 grids. User can not initiate the command if insufficient fuel is available.

Timeshift

This command allows user to activate Timeshift weapon to freeze time and move through time.

Exceptions:

This only works with a moving object and each unit cost 3 units of fuel. User can not initiate the Timeshift if insufficient fuel is available.

Stop Vehicle

While the vehicle approaches an emergency vehicle or arrives in the direction of school zone grid, user must initiate this command. Failure to stop the vehicle stop the vehicle, the engine consumes one unit of fuel.

Start Vehicle

This feature complements the Stop command. Vehicle cannot be moved if Stop command was initiated until the Start command crank up the engine.

Display GPS Map

GPS can serve multi-purpose like displaying the map, allows driver to view traffic update on the grids and allows driver to see the vehicle location on the map.

Exceptions:

None

Functional Behavior

To initiate the system, user is asked to enter a command “start” at the DOS prompt. Help menu can be accessed at any time by simply type “HELP” . To exit the program is simply type “QUIT” or “Q”.

Pre conditions:

Application is to be loaded on DOS prompt. To activate a DOS prompt is simply run a command on Windows (see figure 1.1). Type “Vehicle” without the double quote on the DOS command.

Figure 1.1

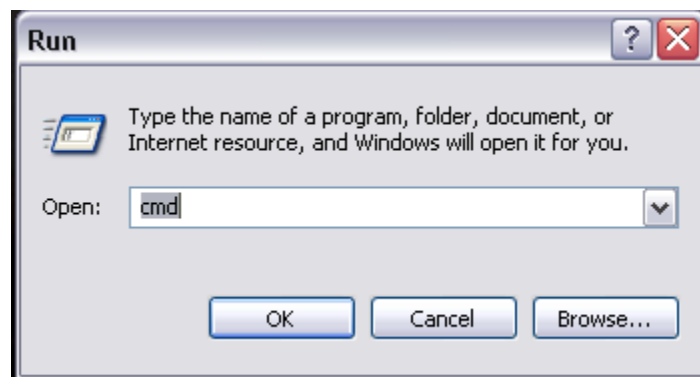


Figure 1.2



Post-conditions:

User can exit the program at any time during run-time by simply execute a command "Q".

[Environmental Considerations and Requirements](#)

Speed: a minimum of Intel/AMD single CPU

Memory: a minimum of 256mg RAM

Host Requirement: DOS, Windows XP and Vesta

Display Card: should be able to run DOS prompt.

Design and Design Documentation

Abstract

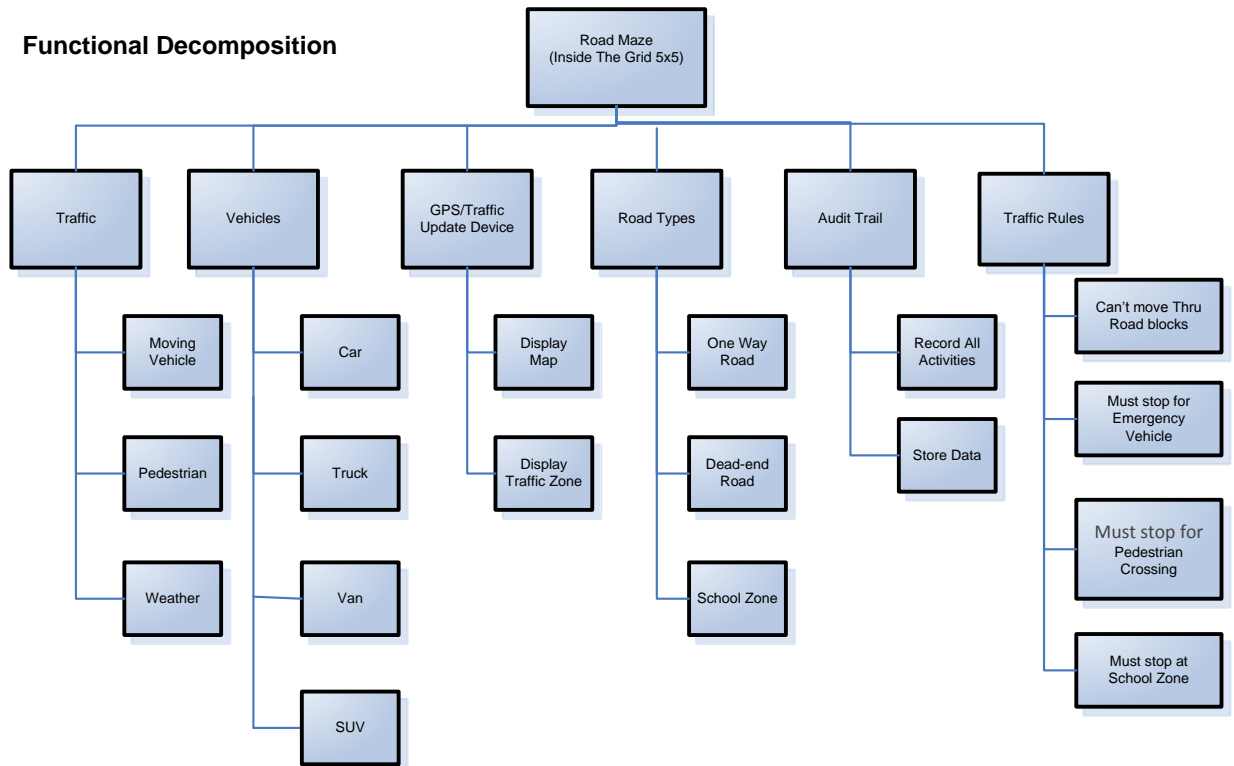
The goal of this application is to design a moving object similar to a board game with table of workspace that enable an object to move within the defined limited area. Several other none moving objects are incorporated to work side by side with the moving object. The grid is an overall top down function where every other element is in its subset.

System Description

The grid size is 5x5; each block contains several objects that can function dependent and independent from one another. The moving object is a vehicle. Its function is to except a series of commands to navigate within the defined limit of maze. There are several class functions such as the GPS navigation system, the vehicle system monitor and on board weaponry are incorporated to aid the vehicle.

External systems included included oneway roads, two land road, dead-end, school zone are, and all known traffic conditions. These external systems work against the vehicle object.

Functional Decomposition



CRC Cards

Class: GridObject	
Super Class:	
Subclasses:	
Responsibilities	Collaborators
<ol style="list-style-type: none">1. Create grid size 5x52. Create dead-end road3. Create school zone4. Create one-way road	<ol style="list-style-type: none">1. Display Interface

Class: Traffic (based class)	
Super Class:	
Subclasses: MovingVehicle, Pedestrian, Weather	
Responsibilities	Collaborators
<ol style="list-style-type: none">1. Pedestrian Crossing2. Bicycle Racing3. Icy Road4. Flood5. Brush Fire6. Parade Vehicle7. Emergency Vehicle8. Riot9. Accident	<ol style="list-style-type: none">1. Display Interface

Class: Traffic (based class)	
Super Class:	
Subclasses: MovingVehicle, Pedestrian, Weather	
Responsibilities	Collaborators
<ol style="list-style-type: none"> 1. Pedestrian Crossing 2. Bicycle Racing 3. Icy Road 4. Flood 5. Brush Fire 6. Parade Vehicle 7. Emergency Vehicle 8. Riot 9. Accident 	<ol style="list-style-type: none"> 1. Display Interface

Class: Weather	
Super Class: Traffic	
Subclasses:	
Responsibilities	Collaborators
<ol style="list-style-type: none"> 1. Icy Road 2. Flood 3. Brush Fire 	<ol style="list-style-type: none"> 1. Display Interface

Class: MovingVehicle	
Super Class: Traffic	
Subclasses:	
Responsibilities	Collaborators
<ol style="list-style-type: none"> 1. Bicycle Race 2. Flood 3. Brush Fire 	<ol style="list-style-type: none"> 1. Display Interface

Class: Pedestrian	
Super Class: Traffic	
Subclasses:	
Responsibilities	Collaborators
<ol style="list-style-type: none"> 1. Crossing 2. Rioting 3. Brush Fire 	<ol style="list-style-type: none"> 1. Display Interface

Class: GPS	
Super Class:	
Subclasses:	
Responsibilities	Collaborators
<ol style="list-style-type: none"> 1. Tracking Vehicle Coordinates 2. Create x & y Axis 3. Create Map on the grid 4. Tracking Vehicle moving thru traffic 5. Update traffic activities 	<ol style="list-style-type: none"> 1. Display Interface 2. Display map 3. Accessing Data from Road database 4. Accessing Data from Traffic Data

Class: Vehicle	
Super Class:	
Subclasses: Car, Truck, Van, SUV	
Responsibilities	Collaborators
<ol style="list-style-type: none"> 1. Start Engine 2. Stop Engine 3. Turn Left 4. Turn Right 5. Move Up 6. Move Down 7. Initiate Timeshift (Replaced Photon Torpedoe) 8. Fuel Units 9. Timeshift Units 	<ol style="list-style-type: none"> 1. Display Interface 2. GPS

Class: Car	
Super Class: Vehicle	
Subclasses:	
Responsibilities	Collaborators
<ol style="list-style-type: none"> 1. Fuel 2. Weapon (Timesift) 3. EPA Gas mileage 	<ol style="list-style-type: none"> 1. Display Interface

Class: Truck	
Super Class: Vehicle	
Subclasses:	
Responsibilities	Collaborators
<ol style="list-style-type: none"> 1. Fuel 2. Weapon (Timesift) 3. EPA Gas Mileage 	<ol style="list-style-type: none"> 1. Display Interface

Class: Van	
Super Class: Vehicle	
Subclasses:	
Responsibilities	Collaborators
<ol style="list-style-type: none"> 1. Fuel 2. Weapon (Timesift) 3. EPA Gas mileage 	<ol style="list-style-type: none"> 1. Display Interface

Class: SUV	
Super Class: Vehicle	
Subclasses:	
Responsibilities	Collaborators
<ol style="list-style-type: none"> 1. Fuel 2. Weapon (Timesift) 3. EPA gas mileage 	<ol style="list-style-type: none"> 1. Display Interface

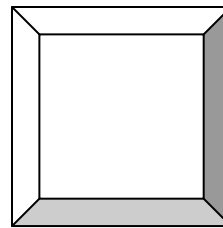
Class: Audit Trail	
Super Class:	
Subclasses:	
Responsibilities	Collaborators
<ol style="list-style-type: none"> 1. Add Record 2. Append Record 3. Remove Record 	<ol style="list-style-type: none"> 1. Display Interface

Classes and Functions

Classes

GridObject

Initializes invariants of a block.



Each object can have multiple invariants with an integer value 1 and 0, where 1 is presented as dead-end road, one way road, or school zone area. A function void called initializeBlock is called to initialize the invariants. These variants are served as constant data.

Traffic

This is a based class allows all other traffic classes such as the Weather class, the MovingVehicle class, and the Pedestrian class access via its data via polymorphism methodology. All invariants are hit in private data.

Weather

Derived class of traffic; all public data are integer values:

- icyRoad
- flood

- brushFire

MovingVehicle

Derived class of traffic; all public data are integer values:

- bicycleRace
- paradeVehicle
- emergencyVehicle

Pedestrian

Derived class of traffic; all public data are integer values:

- crossing
- riot
- accident

GPS

Initializes GridObject, Displays map, tracks vehicle object coordinates, calls all traffic class functions to generate random traffic effects :

Invariants: maxX or X-axis, maxY or Y-axis, minX, minY

Vehicle

Based class from the derived classes Car, Truck, Van and SUV; initial requirements are based fuel, weapon and counters.

Car

Derived class from Vehicle; invariants are fuel units and weapon units

Truck

Derived class from Vehicle; invariants are fuel units and weapon units

Van

Derived class from Vehicle; invariants are fuel units and weapon units

SUV

Derived class from Vehicle; invariants are fuel units and weapon units

AuditTrail

Create a link-list with private attributes string data type, streetType, streetName, streetEvent, vehicleCoordinate, fuelStatus, weaponStatus, and vehicleStatus

Functions

inputCommand

Initializes input buffer pointer to Null and references to the **prompt** function. Takes a string from the **prompt** function and parse only the first character. **validateCommand** is then called to validate user input.

prompt

Takes up to 80 character string from input buffer assigns to the address is referenced by the inputCommand function.

validateCommand

Data are stored in integer array and character array; using a binary search to find a character passed by a parameter.

helpCommand

Reads input string buffer from a file and display on screen.

carFunction

All things considered; invokes every class and function that were described above.

TruckFunction

Most other functionalities are same as carFunction. The pre-conditions of fuel units capacity and fuel units consumption are different.

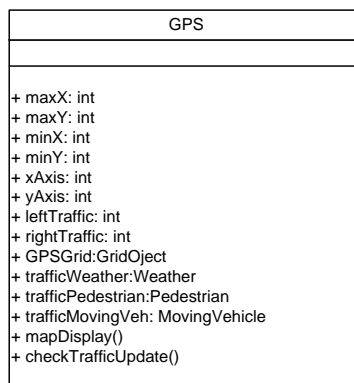
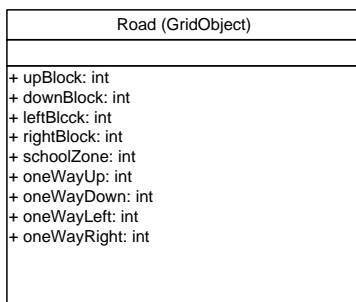
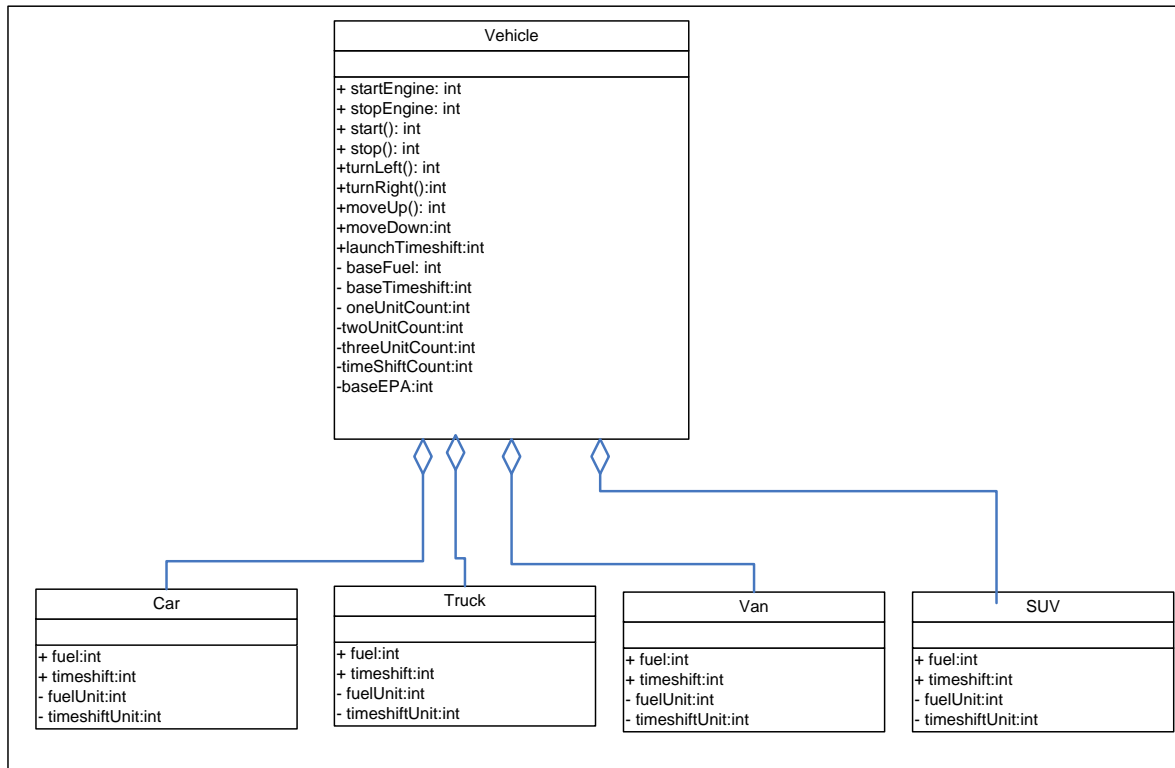
VanFunction

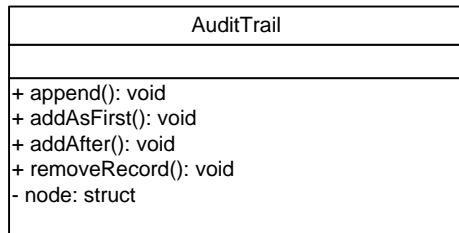
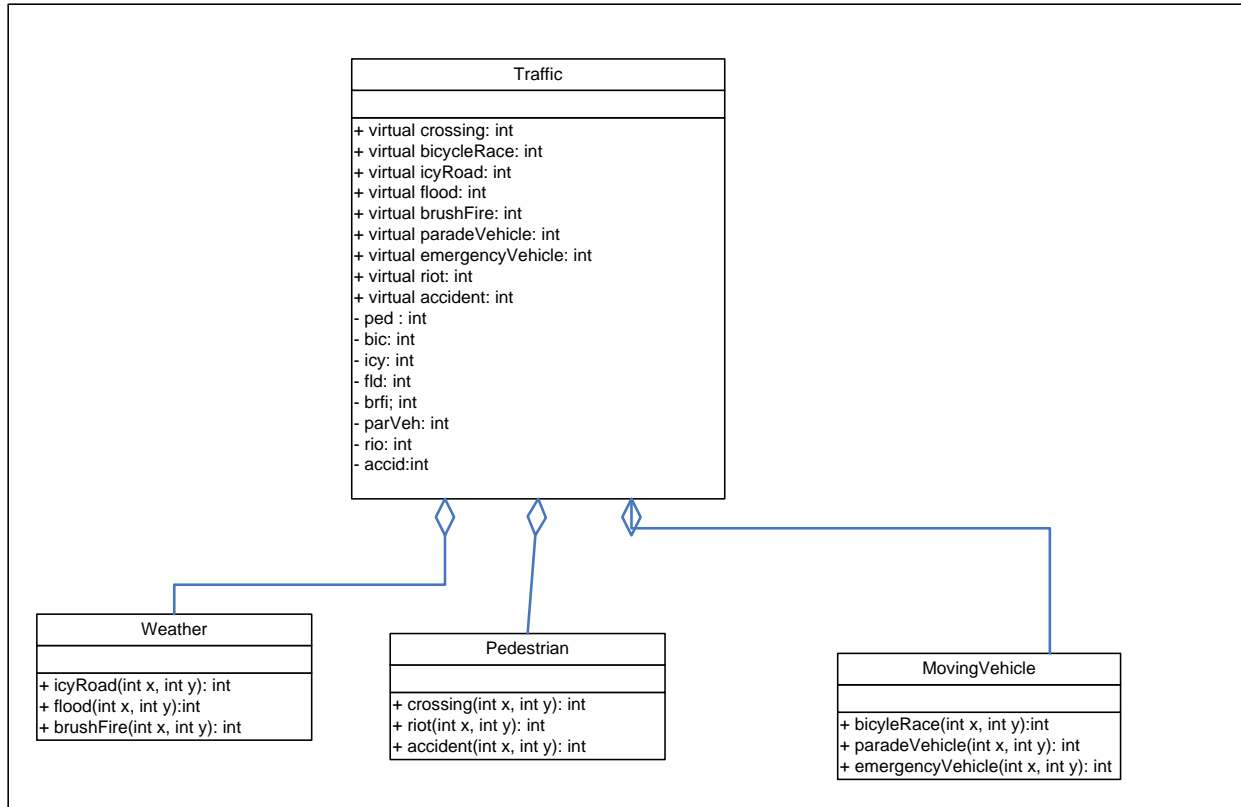
Most other functionalities are same as carFunction. The pre-conditions of fuel units capacity and fuel units consumption are different.

suvFunction

Most other functionalities are same as carFunction. The pre-conditions of fuel units capacity and fuel units consumption are different.

UML Class Diagram





Test Plan

Title of test plan

We are going to cover three scenarios, each under different circumstances. Three maps, with a 5x5 grid are provided to guide our driver step-by-step as the vehicle moves along the maze. Each scenario, we started the vehicle beginning from the initial coordinate, (0,0). On the grid, the roads have multiple fixed point locations defined as one way roads, two lane roads, dead-ends and the school zone areas.

Multiple traffic scenarios can be picked randomly anywhere on the grid. We are using the weather, other vehicles on the roads, and pedestrians.

Our first test is to demonstrate good driving skills, moving a vehicle without violating the traffic rules. With that in mind, of course we have GPS on board system.

Second, we're going to demonstrate how the vehicle can burn more fuel while taking alternate routes.

Finally, to wrap up with the final test, we will demonstrate how we can use the GPS to display map function and an on-board ultimate weapon to aid our Washingtonian reckless driver to reach her destination. However, firing the weapon, it can drain up the vehicle fuel units. Each weapon unit costs 3 units of fuel. The weapon is very limited, three unit capacity.

Cases

As outlined on the test plan, we demonstrated simple executions. We moved our vehicle with GPS guidance (see figure below map 1.0). When tasks are completed, we printed screenshots of activities and the audit reports (a list of vehicle executing activities). We ran through the program source codes and traced functions.

Test setup:

Environment:

Hardware - Laptop equipped with Intel Core CPU with 2g RAM

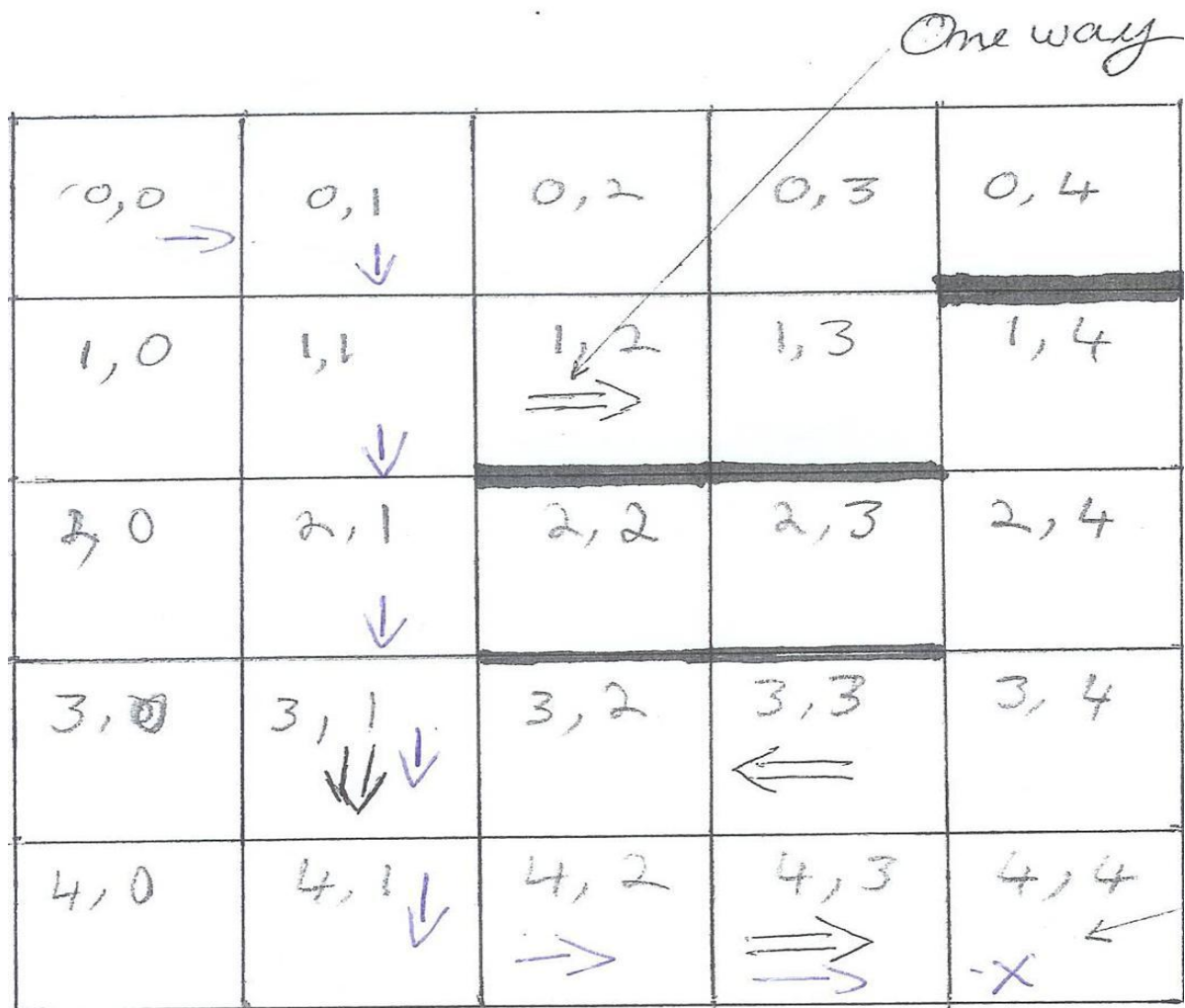
Operating System - Vesta Home Premium

Compiler - Visual C++ 2005 Edition

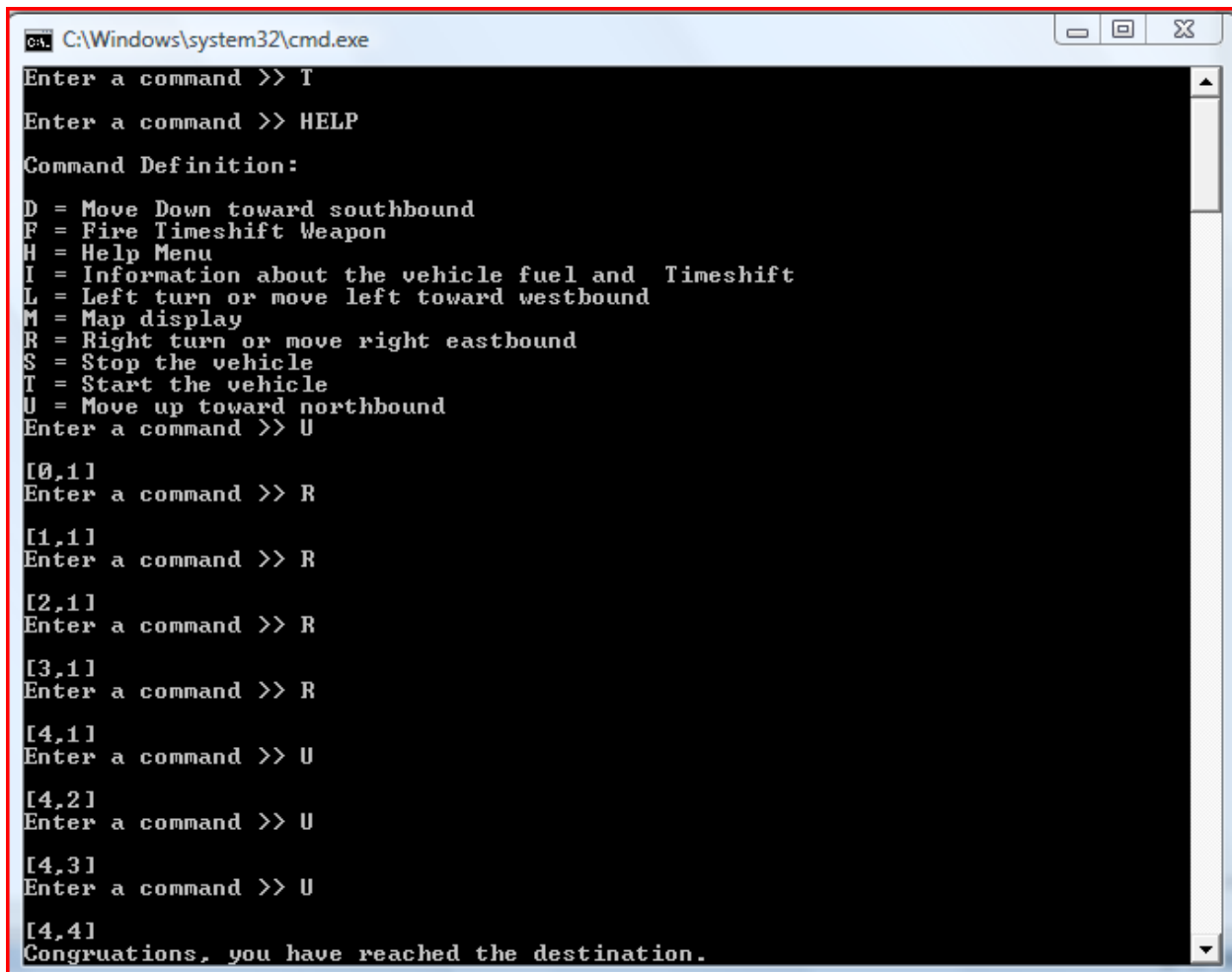
Executions: Case 1

The map said, our vehicle can travel on the following paths: Notice, the vehicle paths are the small arrows. The big arrows are one way roads, and solid lines are dead-ends.

0,0	0,1	1,1	2,1	3,1	4,1	4,2	4,3	4,4
-----	-----	-----	-----	-----	-----	-----	-----	-----

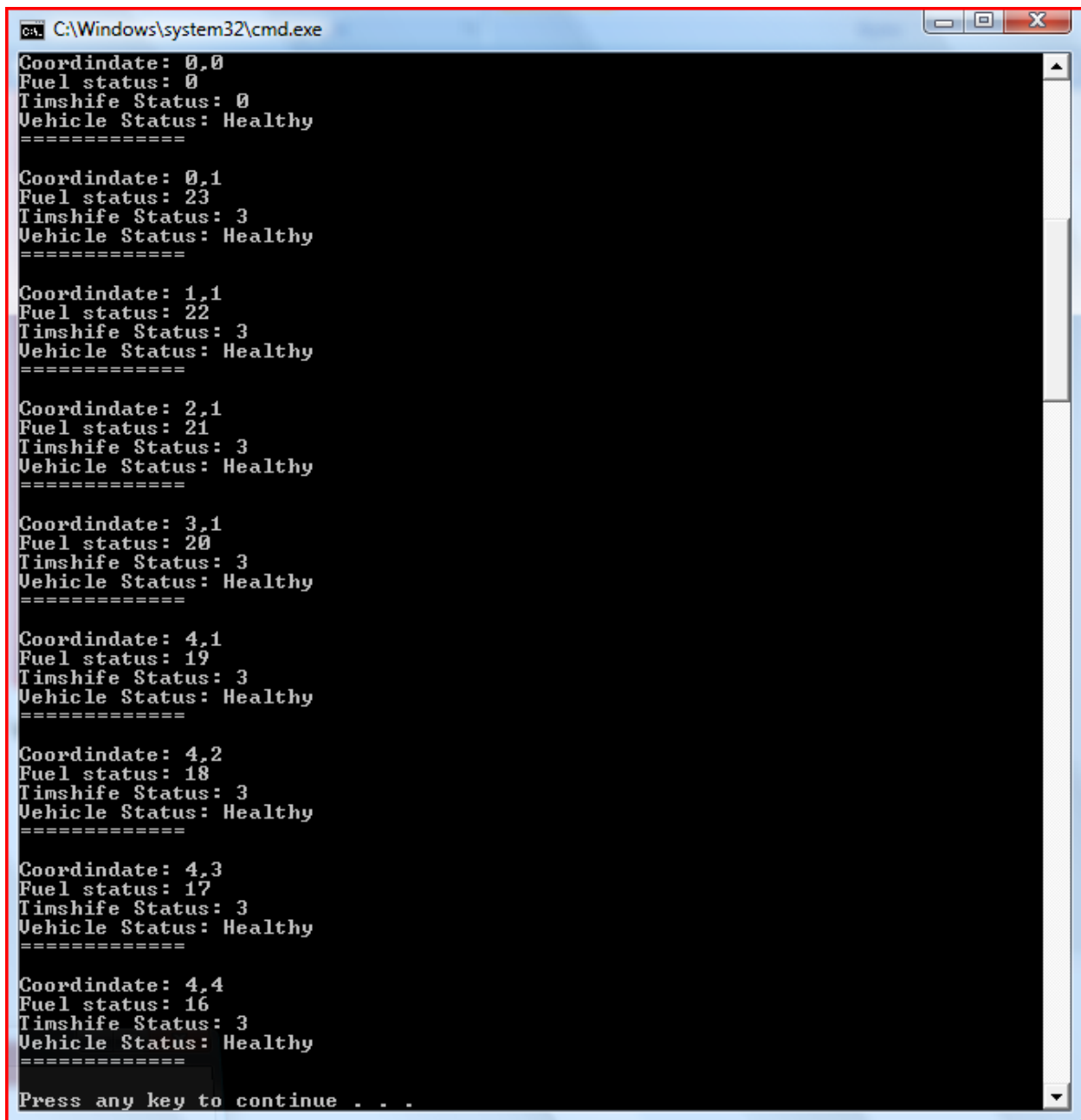


Here we have executed nine commands. The audit report showed that we've reach our destination with some units of fuel to spare.



```
C:\Windows\system32\cmd.exe
Enter a command >> T
Enter a command >> HELP
Command Definition:
D = Move Down toward southbound
F = Fire Timeshift Weapon
H = Help Menu
I = Information about the vehicle fuel and Timeshift
L = Left turn or move left toward westbound
M = Map display
R = Right turn or move right eastbound
S = Stop the vehicle
T = Start the vehicle
U = Move up toward northbound
Enter a command >> U
[0,1]
Enter a command >> R
[1,1]
Enter a command >> R
[2,1]
Enter a command >> R
[3,1]
Enter a command >> R
[4,1]
Enter a command >> U
[4,2]
Enter a command >> U
[4,3]
Enter a command >> U
[4,4]
Congruations, you have reached the destination.
```

We ran through a list of audit report, as you can see our our vehicle is healthy. The vehicle consumed only 7 units of gas. We still have all 3 units of our weapon.

A screenshot of a Windows command prompt window titled "C:\Windows\system32\cmd.exe". The window has a black background with white text. The output shows a series of data points for a vehicle's status over time. Each data point is separated by a line of seven asterisks. The data points are: (0,0) Fuel: 0, Timshife Status: 0; (0,1) Fuel: 23, Timshife Status: 3; (1,1) Fuel: 22, Timshife Status: 3; (2,1) Fuel: 21, Timshife Status: 3; (3,1) Fuel: 20, Timshife Status: 3; (4,1) Fuel: 19, Timshife Status: 3; (4,2) Fuel: 18, Timshife Status: 3; (4,3) Fuel: 17, Timshife Status: 3; (4,4) Fuel: 16, Timshife Status: 3. All vehicle statuses are "Healthy". At the bottom, it says "Press any key to continue . . .".

```
C:\Windows\system32\cmd.exe
Coordindate: 0,0
Fuel status: 0
Timshife Status: 0
Vehicle Status: Healthy
=====
Coordindate: 0,1
Fuel status: 23
Timshife Status: 3
Vehicle Status: Healthy
=====
Coordindate: 1,1
Fuel status: 22
Timshife Status: 3
Vehicle Status: Healthy
=====
Coordindate: 2,1
Fuel status: 21
Timshife Status: 3
Vehicle Status: Healthy
=====
Coordindate: 3,1
Fuel status: 20
Timshife Status: 3
Vehicle Status: Healthy
=====
Coordindate: 4,1
Fuel status: 19
Timshife Status: 3
Vehicle Status: Healthy
=====
Coordindate: 4,2
Fuel status: 18
Timshife Status: 3
Vehicle Status: Healthy
=====
Coordindate: 4,3
Fuel status: 17
Timshife Status: 3
Vehicle Status: Healthy
=====
Coordindate: 4,4
Fuel status: 16
Timshife Status: 3
Vehicle Status: Healthy
=====
Press any key to continue . . .
```

To show how these functions worked, we brought in some of the check list on our program source codes. The first execution, takes input string of characters. The function `inputCommand()` initiates the function `prompt()` get the characters from the input buffer. `ValidateCommand()` function takes a single character and validate in the database dictionary. The `helpComand()` function displays

```
int inputCommand(); //Translate input text to a numeric value
void prompt(char* &buffer); //prompt input command
int validateCommand(char cmd); //validate user input command
```

```
void helpCommand(); //help user to view command definitionhel
```

help text on screen.

Next eight iterations, we executed only two main activity functions. Two overloaded functions from the Super class vehicle, moveUp(GPS&, myGPS) and turnRight(GPS, myGPS).

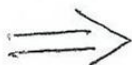
```
class Vehicle
{
public:
    Vehicle();
    GPS myGPS;
    int startEngine;
    int stopEngine;
    void start();
    void stop();
    int turnLeft(GPS& myGPS);
    int turnRight(GPS& myGPS);
    int moveUp(GPS& myGPS);
    int moveDown(GPS& myGPS);
    void launchTimeshift(GPS& myGPS);
private:
    int baseFuel;
    int baseTimeshift;
protected:
    int oneUnitCount; //passing through one unit of grid
    int twoUnitCount; //violating traffic rules
    int threeUnitCount; //firing torpedo
    int timeshiftCount; //
    int baseEPA; //base EPA mileage
};
```

Executions: Case 2

Follow our quest, direction 1.1 is shown on the map below:

One way

0,0	0,1	0,2	0,3	0,4
1,0	1,1	1,2	1,3	1,4
2,0	2,1	2,2	2,3	2,4
3,0	3,1	3,2	3,3	3,4
4,0	4,1	4,2	4,3	4,4

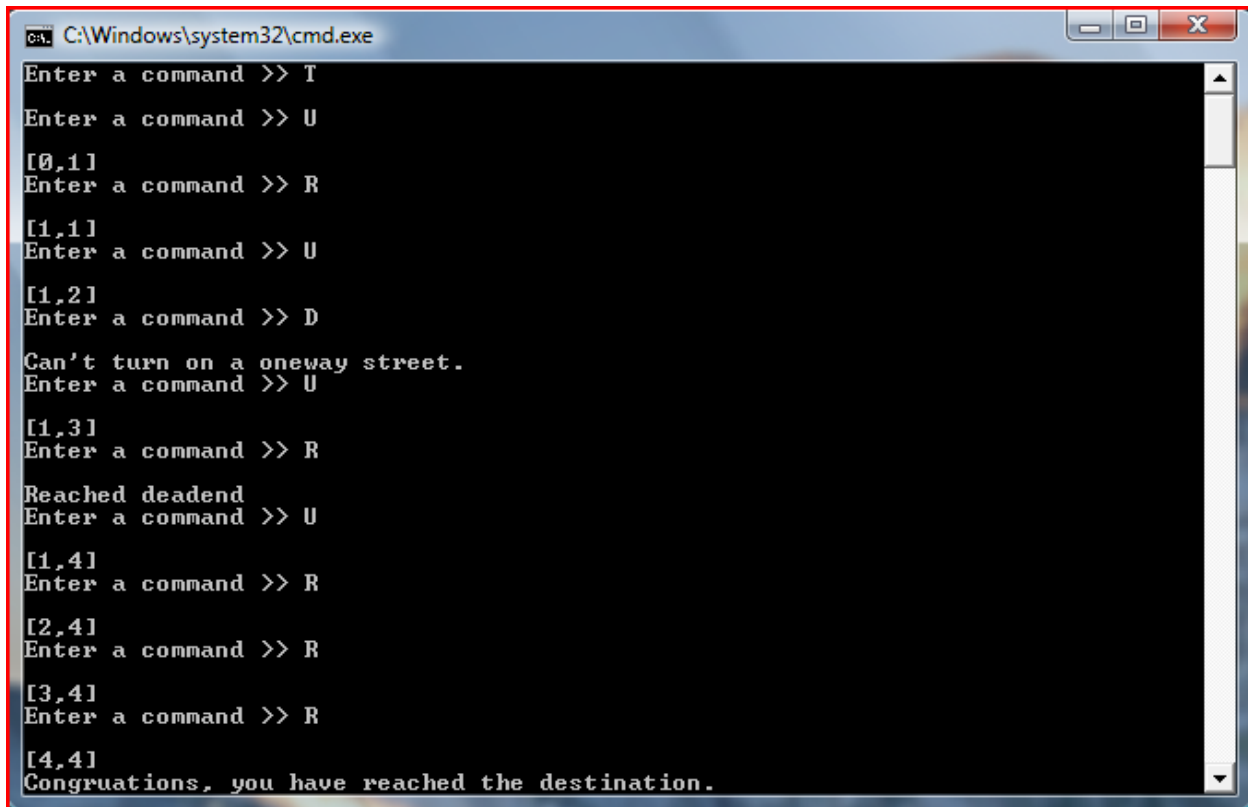


L = WEST



U = NORTH

Ok, it took us a little longer to reach the destination. Somehow our GPS system seemed to have a mind on its own. But at the end we got to hotel safe and sound without an incident. Let's digest a little bit more how the system works. With similar screenshots as Case 1, our audit report calculated the remaining fuel units the same as last time, with 16 units.



```
C:\Windows\system32\cmd.exe
Enter a command >> T
Enter a command >> U
[0,1]
Enter a command >> R
[1,1]
Enter a command >> U
[1,2]
Enter a command >> D
Can't turn on a oneway street.
Enter a command >> U
[1,3]
Enter a command >> R
Reached deadend
Enter a command >> U
[1,4]
Enter a command >> R
[2,4]
Enter a command >> R
[3,4]
Enter a command >> R
[4,4]
Congruations, you have reached the destination.
```

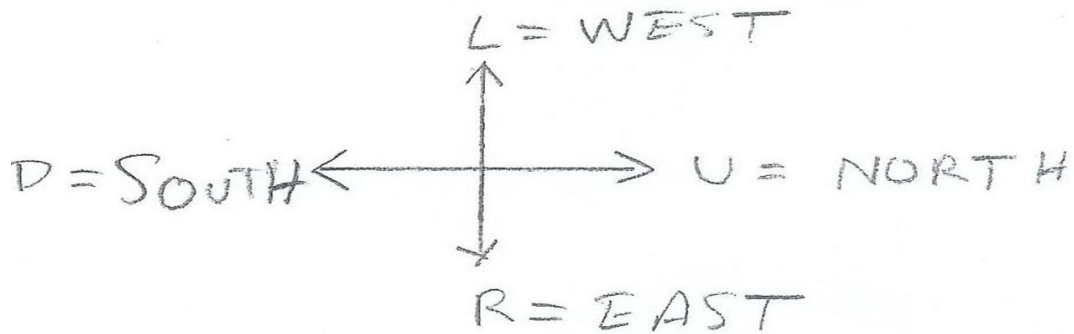
When we ran into obstacles like into the dead-end roads or one way roads, the on board system prevents the vehicle from moving. Therefore, there is no unit of fuel taken.

```
C:\Windows\system32\cmd.exe
Coordindate: 0,1
Fuel status: 23
Timshife Status: 3
Vehicle Status: Healthy
=====
Coordindate: 1,1
Fuel status: 22
Timshife Status: 3
Vehicle Status: Healthy
=====
Coordindate: 1,2
Fuel status: 21
Timshife Status: 3
Vehicle Status: Healthy
=====
Coordindate: 1,2
Fuel status: 20
Timshife Status: 3
Vehicle Status: Healthy
=====
Coordindate: 1,3
Fuel status: 20
Timshife Status: 3
Vehicle Status: Healthy
=====
Coordindate: 1,3
Fuel status: 19
Timshife Status: 3
Vehicle Status: Warning: dead-end
=====
Coordindate: 1,4
Fuel status: 19
Timshife Status: 3
Vehicle Status: Healthy
=====
Coordindate: 2,4
Fuel status: 18
Timshife Status: 3
Vehicle Status: Healthy
=====
Coordindate: 3,4
Fuel status: 17
Timshife Status: 3
Vehicle Status: Healthy
=====
Coordindate: 4,4
Fuel status: 16
Timshife Status: 3
Vehicle Status: Healthy
=====
Press any key to continue . . .
```

Executions: Case 3

Please help me to investigate the scenes of a reckless driver.

0,0 →	0,1 ← →	0,2 ← →	0,3 ← →	0,4 →
1,0 ← →	1,1 ↓ →	1,2 ← → →	1,3 ↓	1,4 → ↓
2,0	2,1	2,2	2,3 ↓ ←	2,4 ←
3,0	3,1 ↓ ↓	3,2	3,3 ← ←	3,4
4,0	4,1	4,2	4,3 → →	4,4 ←



Our reckless driver has a very long list of audit report. We examined more codes, starting with the audit report. We designed our link list table a class called, AuditTrail. When a Move

command is executed, a recorded activities are copied and appended onto the link like. Prior to the program exit, an overloaded function is called to list all the data.

Final Notes:

Due to time constrain , I was unable to incorporate all the functionalities needed to complete the whole application package. The traffic functions didn't seem to response well on the test case. The vehicle start and stop functions are still needed to be done All other types of vehicle functions are needed to be completed as well.

Attached Audit Trail Report (case 3):

Enter a command >> T

Enter a command >> U

[0,1]

Enter a command >> U

[0,2]

Enter a command >> U

[0,3]

Enter a command >> U

[0,4]

Enter a command >> D

[0,3]

Enter a command >> D

[0,2]

Enter a command >> D

[0,1]

Enter a command >> R

[1,1]

Enter a command >> D

[1,0]

Enter a command >> U

[1,1]

Enter a command >> U

[1,2]

Enter a command >> U

[1,3]

Enter a command >> R

Reached deadend

Enter a command >> U

[1,4]

Enter a command >> D

[1,3]

Enter a command >> M

[0,0]: [0,1]: [0,2]: [0,3]: [0,4]:

[1,0]: [1,1]: [1,2]: rB oWU [1,3]: rB [1,4]:

[2,0]: [2,1]: [2,2]: lB rB [2,3]: lB rB [2,4]:

[3,0]: [3,1]: oWR [3,2]: lB [3,3]: lB oWD [3,4]:

[4,0]: [4,1]: [4,2]: [4,3]: rB [4,4]:

LEGION:

dB: Southbound deadend road

uB: Northbound deadend road

lB: Westbound deadend road

rB: Eastbound deadend road

oWD: Oneway Southbound road

oWU: Oneway Northbound road

oWL: oneway Westbound road

oWR: oneway Eastbound road

sZ: School zone area

Enter a command >> U

[1,4]

Enter a command >> R

[2,4]

Enter a command >> D

[2,3]

Enter a command >> H

Command Definition:

D = Move Down toward southbound

F = Fire Timeshift Weapon

H = Help Menu

I = Information about the vehicle fuel and Timeshift

L = Left turn or move left toward westbound

M = Map display

R = Right turn or move right eastbound

S = Stop the vehicle

T = Start the vehicle

U = Move up toward northbound

Enter a command >> F

Enter a command >> R

Reached deadend

Enter a command >> D

[2,2]

Enter a command >> D

[2,1]

Enter a command >> R

Warning: low fuel

[3,1]

Enter a command >> U

Warning: low fuel

Fuel reaches zero, vehicle stopped.

Coordindate: 0,1

Fuel status: 23

Timshife Status: 3

Vehicle Status: Healthy

=====

Coordindate: 0,2

Fuel status: 22

Timshife Status: 3

Vehicle Status: Healthy

=====

Coordindate: 0,3

Fuel status: 21

Timshife Status: 3

Vehicle Status: Healthy

=====

Coordindate: 0,4

Fuel status: 20

Timshife Status: 3

Vehicle Status: Healthy

=====

Coordindate: 0,3

Fuel status: 18

Timshife Status: 3

Vehicle Status: Healthy

=====

Coordindate: 0,2

Fuel status: 17

Timshife Status: 3

Vehicle Status: Healthy

=====

Coordindate: 0,1

Fuel status: 16

Timshife Status: 3

Vehicle Status: Healthy

=====

Coordindate: 1,1

Fuel status: 16

Timshife Status: 3

Vehicle Status: Healthy

=====

Coordindate: 1,0

Fuel status: 14

Timshife Status: 3

Vehicle Status: Healthy

=====

Coordindate: 1,1

Fuel status: 14

Timshife Status: 3

Vehicle Status: Healthy

=====

Coordindate: 1,2

Fuel status: 13

Timshife Status: 3

Vehicle Status: Healthy

=====

Coordindate: 1,3

Fuel status: 12

Timshife Status: 3

Vehicle Status: Healthy

=====

Coordindate: 1,3

Fuel status: 11

Timshife Status: 3

Vehicle Status: Warning: dead-end

=====

Coordindate: 1,4

Fuel status: 11

Timshife Status: 3

Vehicle Status: Healthy

=====

Coordindate: 1,3

Fuel status: 9

Timshife Status: 3

Vehicle Status: Healthy

=====

Coordindate: 1,3

Fuel status: 9

Timshife Status: 3

Vehicle Status: Healthy

=====

Coordindate: 1,4

Fuel status: 9

Timshife Status: 3

Vehicle Status: Healthy

=====

Coordindate: 2,4

Fuel status: 8

Timshife Status: 3

Vehicle Status: Healthy

=====

Coordindate: 2,3

Fuel status: 6

Timshife Status: 3

Vehicle Status: Healthy

=====

Coordindate: 2,3

Fuel status: 6

Timshife Status: 3

Vehicle Status: Healthy

=====

Coordindate: 2,3

Fuel status: 3

Timshife Status: 2

Vehicle Status: Healthy

=====

Coordindate: 2,3

Fuel status: 3

Timshife Status: 2

Vehicle Status: Warning: dead-end

=====

Coordindate: 2,2

Fuel status: 2

Timshife Status: 2

Vehicle Status: Healthy

=====

Coordindate: 2,1

Fuel status: 1

Timshife Status: 2

Vehicle Status: Healthy

=====

Coordindate: 3,1

Fuel status: 1

Timshife Status: 2

Vehicle Status: Warning: one way

=====