Technical Manual

Project Name: Drone Traffic Simulation

Course: CASE4

Module Title: CA Final Year Project

Module Code: CA400

Student Name: Jack Murphy (13436498)

Supervisor Name: Prof. Alan Smeaton

Submission Date: 22/05/2017

Abstract:

The following document details my project which is on Drone Traffic Simulation. It outlines the various details of the project as well as motivation behind choosing it. The document also examines the results of the simulation outlining the optimal parameters for it. The purpose of this project was to prove that a drone is safe to use and have the potential to be highly profitable. This is proven by statistics recorded by the system. Using this statistics analysis can be done to determine the best possible outcome for the simulation and thus a real-life scenario of drones being automated.

Table of Contents

1. Introduction

1.1 Overview

My project is a system simulation of drone traffic in a cityscape. The simulation is a 3D Model of a basic cityscape with drones flying above the streets. The drones fly at a certain using the cover of the buildings to protect them from the elements, this is the attraction of cityscape environments. This simulation is looking towards the future of drones for commercial use. Their payloads will be deliveries, ultimately replacing standard means of delivery such as car or bicycle. It could also be useful for assisting with first responders for accidents and emergencies. With such a future, what should drone traffic look like, how many and how fast should they be able to fly? Should there be a traffic system with traffic lights, rights of way at crossings, roundabouts, etc.? These are the questions I hoped to answer with this simulation.

The simulation runs a smooth flow of traffic for a given number of drones. Each drone navigates along a series of points from one point in the city to another and back again. While the simulation is running a series of statistics are taken, which capture various performance outputs of the system, such as average distance travelled by the drone, number of drones at the current time, if there are collisions, etc. The optimal output of the system is to maximise the system output.

The map is divided into a series of horizontal and vertical streets. Each street has two possible lanes of traffic which both travel in a single direction. The streets all contain a list of point which have both horizontal and vertical neighbours. All the drones generated come in varying sizes. This provides a better representation of how it would be in the real world. The simulation can be run with all drones going the same speed or all drones travelling at different speeds. It can be determined from this whether a speed limit should be applied to the drones or not. Drones are generated in a normally distributed fashion and once they have completed their route they are removed from the map.

It is my view that this system will shape the future of legislation around drones. As it stands the laws controlling drone usage are very limiting. These stringent controls have a negative effect on the many beneficial uses of drones. These uses range from commercial uses like package delivery to emergency response. The aim of this system is to create a model by which you can have hundreds or thousands of auto-piloted drones flying above the streets in a safe and continuous manner. The statistics I have generated should back this up.

1.2 Glossary

1.2.1 .Net Framework

- Software framework that allows interoperability between various programming languages.

1.2.2 CSharp (C#)

- Programming language that incorporates multiple programming disciplines

1.2.3 R

- Language developed for use with probabilities and statistics, I will use to analyse the data output and graphically represent the data.

1.2.4 SQL Server Management Studio (SSMS)

- SQL Server Management Studio (SSMS) is an integrated environment for managing any SQL infrastructure, from SQL Server to SQL Database.

1.2.5 Entity Framework (EF)

- Entity Framework (EF) is an open source object-relational mapping (ORM) framework for ADO.NET.

2. Motivation

While researching for project ideas last summer I came across drones and their use in a commercial environment. Reading articles from high profile journalistic establishments such as Forbes and The Economist, I realised that this is a viable and possibly highly profitable emerging industry. According to one article from 2015 in The Economist, writing about the founder of a French company Delair Tech who fly provide commercial drone flying services they say, "Mr. Benharrosh says annual turnover is €3m ($3.2m), but growing "by 200%", after four years in business." This is a large turnover for such a new company, if this was implemented by a larger company with greater resources the return on investment could be significant.

Seeing the potential in this newly emerging industry, I set about researching the rules and regulations associated with drones. As of now the regulation of drones in Ireland are managed by the Irish Aviation Authority. The legislation on the operation of drones is quite strict, the reason for this is that drones are a relatively new and there is a significant number of unknowns associated with their use. Where there are unknowns there is the potential for risk. Currently drones are not allowed to be automated and must be operated by an authorised person and it cannot leave the line of sight of the operator. So, the motivation for this project is to eliminate some of these unknowns thus proving that automating drones is safe and profitable.

3. Research

3.1 Research into the Idea

Beginning this project my knowledge of drones was limited to stories and experiences of seeing drone flying around the park, and of videos taken by photography drones. It had never occurred to be that they could provide a commercial benefit. That is until I heard that Amazon were trialling drones for delivery purposes. I then began researching drones for commercial use. I came across several beneficial articles such as the one I highlighted earlier on in the document. These articles highlighted the commercial benefits of using drones.

After discovering these benefits my next job was to find whether automated drones are a feasible option. For this I researched many use cases of automated drones. Primarily, I looked at the Amazon's trial of drones for delivery. So far although, facing a few obstacles the project has been, more or less, a success. This was the evidence I needed to prove that automating drones was feasible. As well as this, it had been done on a relatively small scale also.

3.2 Research into Technologies

Once I had decided on my idea of a simulation, I began researching the technologies I would need to complete the task. I made the decision that the simulation should use a game engine.

The reason for this is that I felt the graphical element would engage users, making the simulation more appealing. Having worked with .Net Framework before I knew the many benefits of it and how it could benefit my project. When researching gaming engines many seemed to use .Net. Initially I chose Unity as all my research showed that this was the most well supported and documented gaming engine available. However, when implementing this I found that the .Net version which it was running was long out of date. This was a major problem which I will outline in a later section. I eventually found a gaming engine that was relatively new but supported the newest versions of .Net, Xenko was the game engine I chose.

Another technology I had to put my research in was the storing of my data. As I would be collecting a large amount of statistics I needed a data store capable of integrating with .Net and allowing easy access to the data. My research pointed to SQL Server, SQL server is easily integrated with Visual Studio and .Net projects as they are both part of the Microsoft technology stack they are built to work in conjunction with each other.

Test-driven development is an essential practice used in industry, it has many benefits. To implement this I had to find a suitable testing framework. Researching different possibilities, I found that two stuck out, these were MsTest and NUnit. On weighing the pros and cons of both options it was clear that NUnit had many advantages over MsTest. An example of one of these is readable Assertion methods.

A key component of my system is the gathering and representation of statistics. Currently there are many analytical languages available, the main contenders in this area are Python and R. Both are freely available through Visual Studio, therefore the advantages and disadvantages of both had to be weighed. However, most articles and sources I read agreed that R has a slight advantage given that it was built with statisticians in mind and therefore is more useful for data analysis.

3.3 Researching Algorithms

The main algorithm which forms the backbone of the simulation is the drone navigation. This is a complex algorithm which takes in various constraints. When I first began my algorithm, I made the decision that when a drone is to be generated a route should be created to from a random point on the map. This route would travel through various points along the way. This first thing I researched was an optimum path algorithm and the travelling salesman problem. During my research, I came across Lee's algorithm for the Maze Router problem, this problem is used for solving the circuit paths on motherboards. These problems were like what I needed but did not quite satisfy the needs of my system.

After much research, I went to Dr. Liam Tuohey for advice on this problem as his expertise are in mathematical programming and simulation. He helped me to define the problem as a minimum Manhattan network problem. This allowed me to continue my research into the navigation problem and gave me a starting point. Researching this, I came across dynamic programming which breaks a complex problem down into smaller sub problems. This research was pivotal to the algorithm I have implemented today

4. Design

4.1 System Architecture

4.1.1 Architectural Overview Diagram

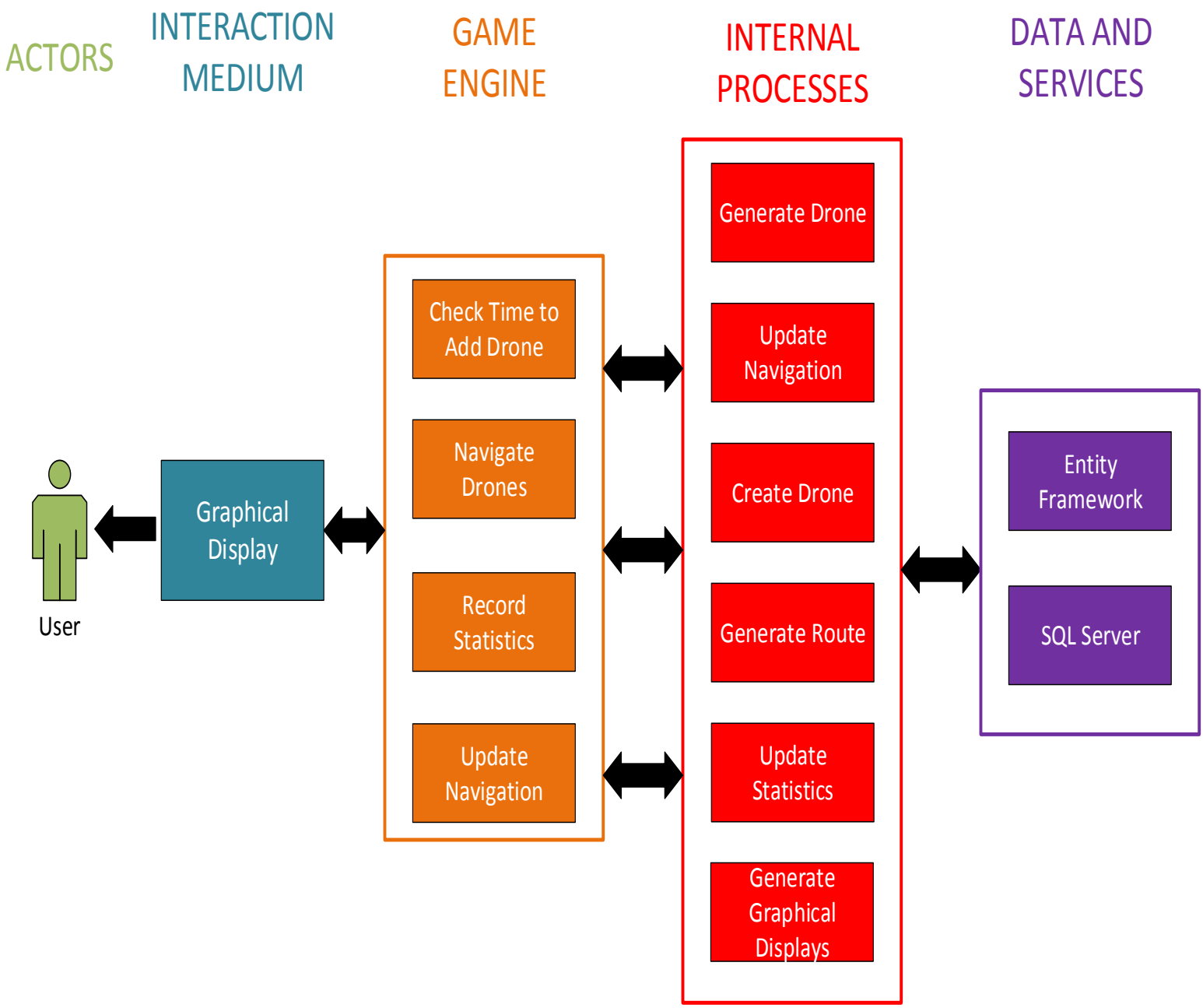


Figure 1. Architectural Overview Diagram

4.1.2 System Architecture Diagram

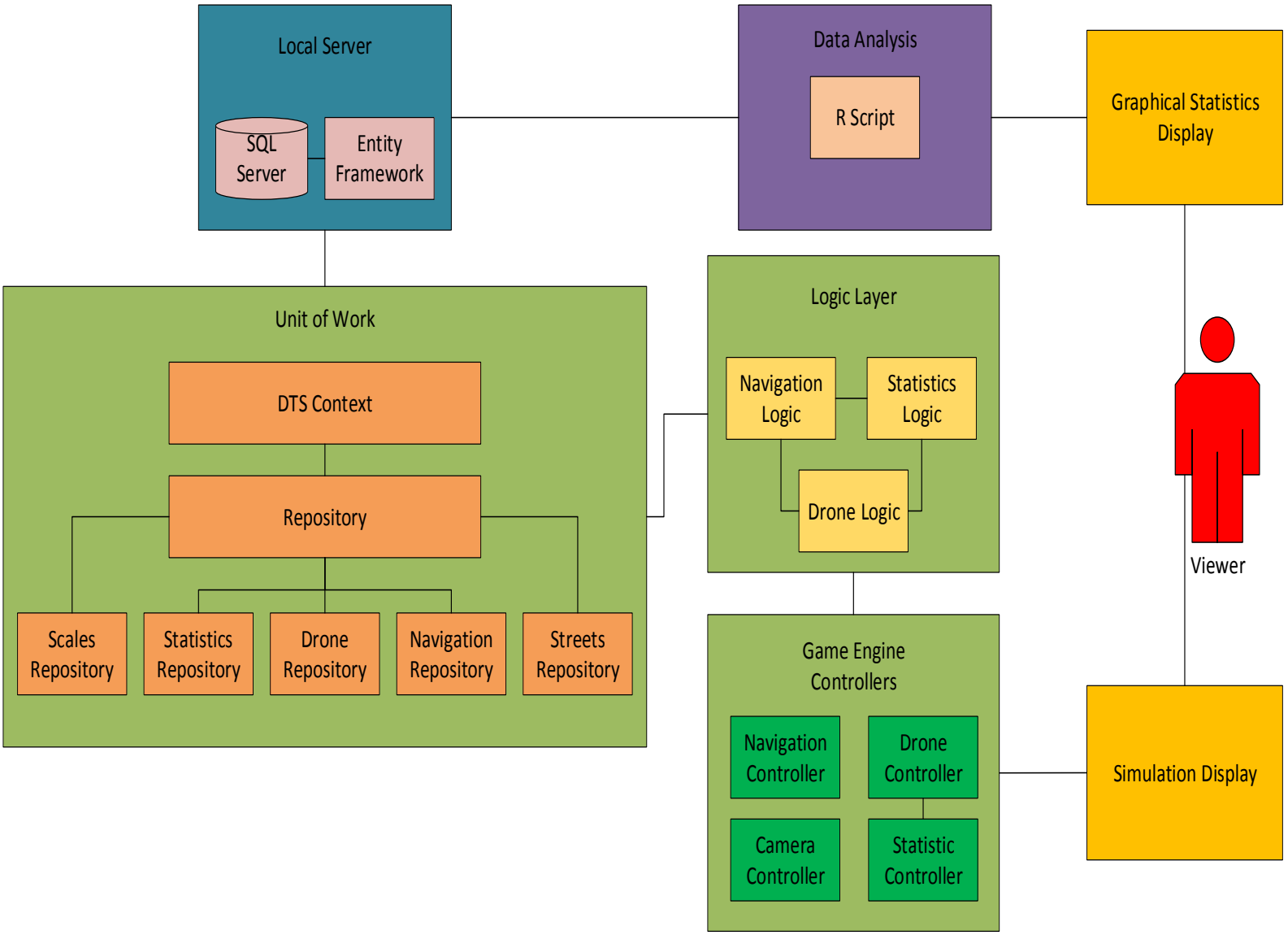


Figure 2. System Architecture Diagram

4.2 High-level Design

4.2.1 Component Model

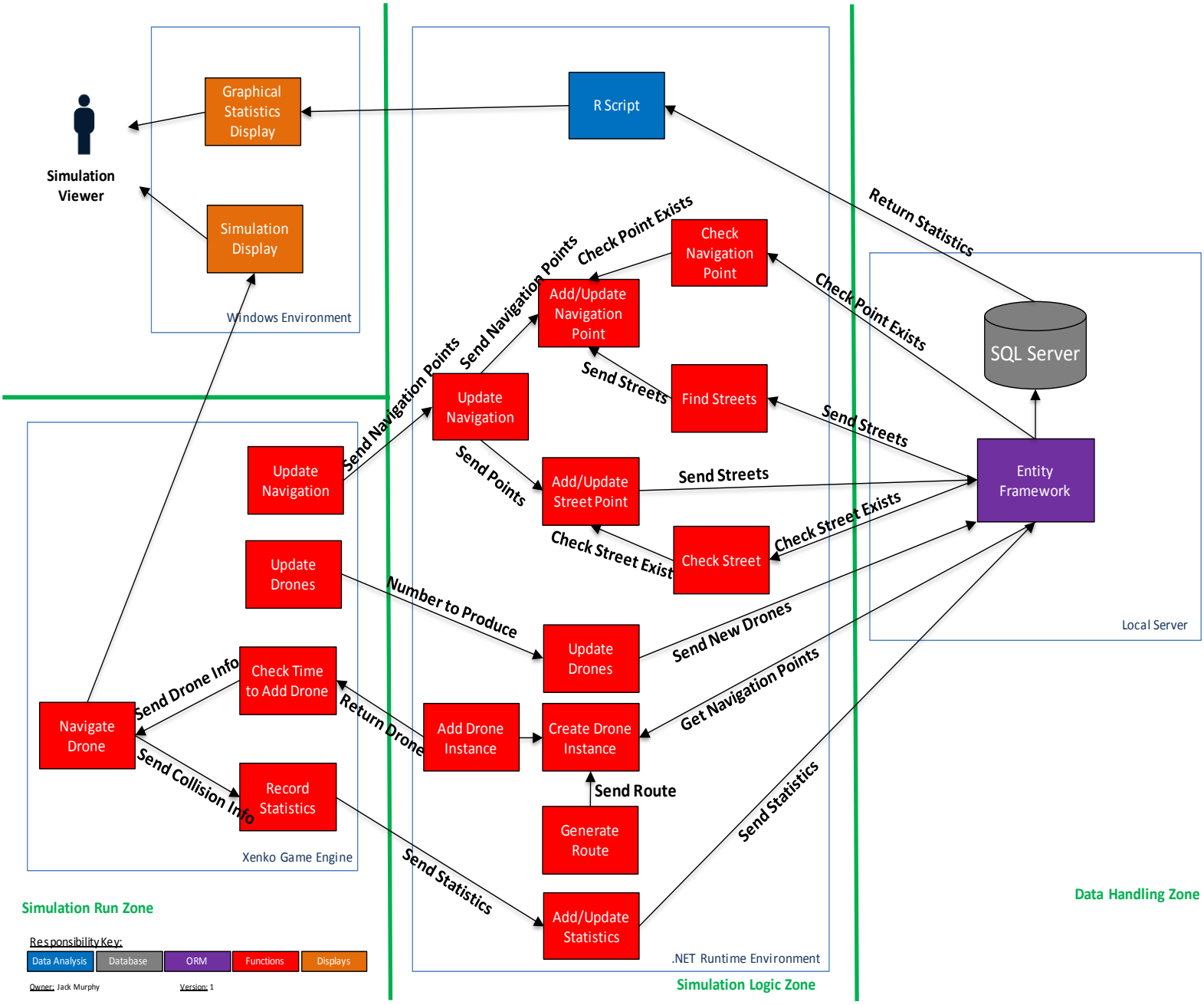


Figure 3. Component Model

4.3 Database Design

4.3.1 Entity-Relationship Diagram

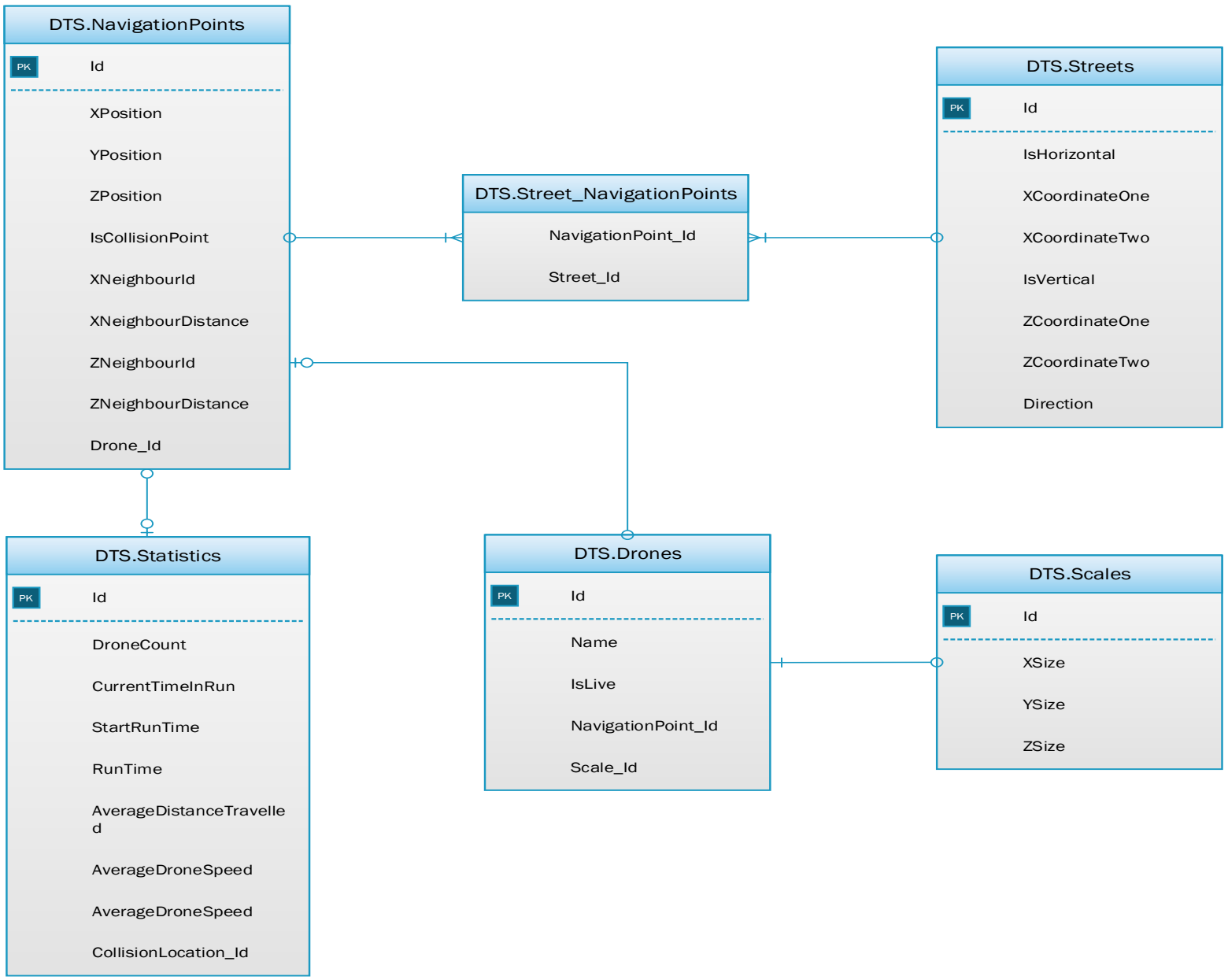


Figure 4. Entity-Relationship Diagram

5. Implementation

5.1 Approach

From the beginning of this project I would use the best standards and practices which I have learned throughout the course and on my INTRA placement. First and for most of these practices was the use of a layered architecture. This layered architecture separates the different areas of a project into their own project. So, the presentation layer or in my case the game engine, is separated from the main logic and the main logic is separated from the data access. This separation of concerns has many benefits, for example, one part of the project can be worked on without affecting another part of the project.

One of the practices that I learned on INTRA was the use of design patterns. A design pattern is a general repeatable solution that can be used to solve a common software development problem, essentially it is a description of a template in software development. In my system, I implemented a common design pattern that is in use with .Net systems. It is the Repository pattern inside a Unit of Work pattern. The Unit of Work repository pattern is way of handling database transactions in a .Net environment. It maintains in-memory updates and then sends these in memory updates to the database as one big transaction. So, one of these transactions is one unit of work. The Repository pattern is a way of separating the business logic in an application from the entity models and data mappings. This has many advantages over other methods in that it centralises the data logic or data access logic, it provides a substitution point for unit and integration testing as well as providing a flexible architecture which can adapt over time. The Unit of Work with Repository pattern incorporates both design pattern essentially placing the repositories inside the unit of work.

Another beneficial practice that I used was test-driven development. This practice allowed me to constantly improve and refactor my code as the tests were in place to detect any breaking changes that refactoring may have caused.

5.2 Navigation

5.2.1 Updating Navigation

Navigation is a key element for my system. The drones' ability to manoeuvre around the map is essential. The first function that is implemented is the UpdateNavigation function. Information is passed from the Navigation Controller in the game engine to the Navigation Logic class my Simulation Logic Layer. This information is a list of navigation points. Navigation points are the unique coordinates used by the drones to navigate through the map, each of these navigation points has two streets, a street going up or down and a street going left or right. For this reason, the streets are added first.

The streets are added by taking the unique X and Z coordinates. These lists are checked against the existing table. If they already exist then no new ones are created, otherwise the new streets are added. Once the streets have been added then the navigation points can be added, for each navigation point, it's streets are found and then it is checked against the database to ensure it exists. If it doesn't exist, then a new point is added.

5.2.2 Generating the Route

Route generation is a key part of this system. When a drone is created, a route must be created for that drone. The algorithm begins with the start point and the end-point. Its goal is to navigate to the end-point and back. However, there are multiple constraints on the system like the direction in which the drone can travel. The drone may only travel straight along the X or Z

axis, and it can only go a single direction on each of these axes. This direction is dependent on the street on which the drone is on.

The navigation algorithm, starts at the start point, obtaining both the X and Z neighbour point. It then triggers a function to determine which point to choose next. This function chooses the point based on several varying factors. The reason for this is that the problem is NP-Complete, meaning there is no optimal solution to the problem. Thus, all variances must be taken into consideration. The method first checks which direction the target point is in, i.e. South-East, South-West, North-East or North-West. If it is directly north, south, east or west then it moves to the point nearest the target. For each of these conditions it checks, does the current point match certain combinations of street directions e.g. "right" and "up" if any of these conditions match then it chooses the point accordingly. Otherwise it checks the direction of the target point and chooses accordingly. This is a highly complex method with a Cyclomatic Complexity score of 98.

When the next point has been chosen then the algorithm checks it the current point either the X or the Z neighbour of the target point, if not then it recurses and the process repeats all over again. If it is the target point, then it swaps the start point and the target point and continues. The method exits when it has reached the target point for the second time i.e. navigating there and back.

5.3 Drones

5.3.1 Updating drones

When the simulation begins, a method is triggered from the drone controller to update the number of drones in the database. The number of drones is passed from the controller, if the number is less than the number of drones in the database then the difference will be added. If the number is greater than the number in the database, then the difference is removed and the table is reseeded. This keeps the data consistent throughout all the simulation runs.

5.3.2 Adding a drone

Adding a drone is another crucial method to this system, it is triggered by the drone controller in the game engine. This controller updates on each iteration of the game loop. When the function is called, it checks if it is time to create a drone, as per the normally distributed trigger times. If it is time, then it initialises the drone with a random target point and updates the database for that drone to be set live. The method then creates a new random start point and generates a route. Once this it done the initialised drone object is sent back to the drone controller where it navigates through the list of points.

5.4 Statistics

5.4.1 Updating the Statistics

Statistics for each run of the simulation are stored in the database. The statistics are stored every five minutes in the database run. The statistics are taken in a controller class in the game engine, these include Average Distance Travelled, Average Speed, Number of Drones, etc. The statistics may also be triggered from the drone controller. If a collision is detected, then the statistics for this are recorded including the location of the collision. This information can be used in my analysis of the system, to determine what the best parameters are.

6. Testing

As I mentioned earlier, throughout this project I have maintained a continuous test-driven approach. This has been extremely beneficial for me as it has allowed me to continue making

changes and testing each individual method as I go. My testing is made up of a combination of unit tests and integration tests. The integration tests cover methods that call other methods for example this test for my Generate Route Method.

```
[Test]
0 references | Jack, 15 hours ago | 1 author, 4 changes
public void TestGenerateRoute()
{
    //Arrange
    var unitOfWork = new UnitOfWork(new DtsContext());
    var drone = new Drone
    {
        TargetPoint = new NavigationPoint(unitOfWork.NavigationPoints.GetSingle(Guid.Parse("B955D974-2D5C-45A3-AEBA-009AD23BAC3A"))),
        CurrentPoint = new NavigationPoint(unitOfWork.NavigationPoints.GetSingle(Guid.Parse("161E22AE-B152-4D7F-AC68-58201EF025DA"))),
    };
    var route = new List<NavigationPoint>
    {
        drone.CurrentPoint,
        drone.TargetPoint
    };
    var droneRandom = new Drone
    {
        TargetPoint = new NavigationPoint(unitOfWork.NavigationPoints.GetRandom(Guid.Parse("B955D974-2D5C-45A3-AEBA-009AD23BAC3A"))),
        CurrentPoint = new NavigationPoint(unitOfWork.NavigationPoints.GetRandom(Guid.Parse("B955D974-2D5C-45A3-AEBA-009AD23BAC3A"))),
    };
    var routeRandom = new List<NavigationPoint>
    {
        droneRandom.CurrentPoint,
        droneRandom.TargetPoint
    };

    File.AppendAllText(
        @"C:\Users\Jack\Documents\Fourth Year\Fourth Year Project\2017-ca400-murpj238\docs\blog\RandomNavigationPoints.csv",
        "TestGenerateNavigationPoint," + droneRandom.TargetPoint.Id + "," + droneRandom.TargetPoint.XPosition + "," +
        droneRandom.TargetPoint.ZPosition + "," + droneRandom.TargetPoint.Id + "," +
        droneRandom.CurrentPoint.XPosition + "," + droneRandom.CurrentPoint.ZPosition+"\n");
    //Act
    var result = NavigationLogic.GenerateRoute(drone, route,
        unitOfWork.NavigationPoints.GetNaviationPoints().Where(x => x.Id != drone.TargetPoint.Id).ToList(), 0);
    var resultRandom = NavigationLogic.GenerateRoute(droneRandom, routeRandom,
        unitOfWork.NavigationPoints.GetNaviationPoints().Where(x => x.Id != droneRandom.TargetPoint.Id)
            .ToList(), 0);

    //Assert
    Assert.IsNotEmpty(result,"The list does not contain any values");
    Assert.AreEqual(result.Last(),drone.TargetPoint,"The last value should be the target point");
    Assert.Greater(route.Count,2,"The route only contains the start and finish points");
    Assert.AreEqual(35, result.Count, "The list has too many or too few points");
    Assert.True(result.Any(x => x.Id == Guid.Parse("e378a340-b368-4cc1-b412-28ed418a3d24")),"This point does not exist in the list");
    Assert.IsNotEmpty(resultRandom,"The route returned empty");
    Assert.True(result.Any(x => x.Id == Guid.Parse("B955D974-2D5C-45A3-AEBA-009AD23BAC3A")));

    unitOfWork.Dispose();
}
```

This example shows how the generate route method is called in the test method. As can be seen the unit of work pattern is also being tested. If the get methods fail, then the test will also fail. The results of these database transactions are also logged in an excel file. This allows me to quickly discover any errors that may occur before the GenerateRoute method is run. I decided to use this in my test cases as .Net and Entity Framework use lazy loading for database entries meaning that values are only called when needed.

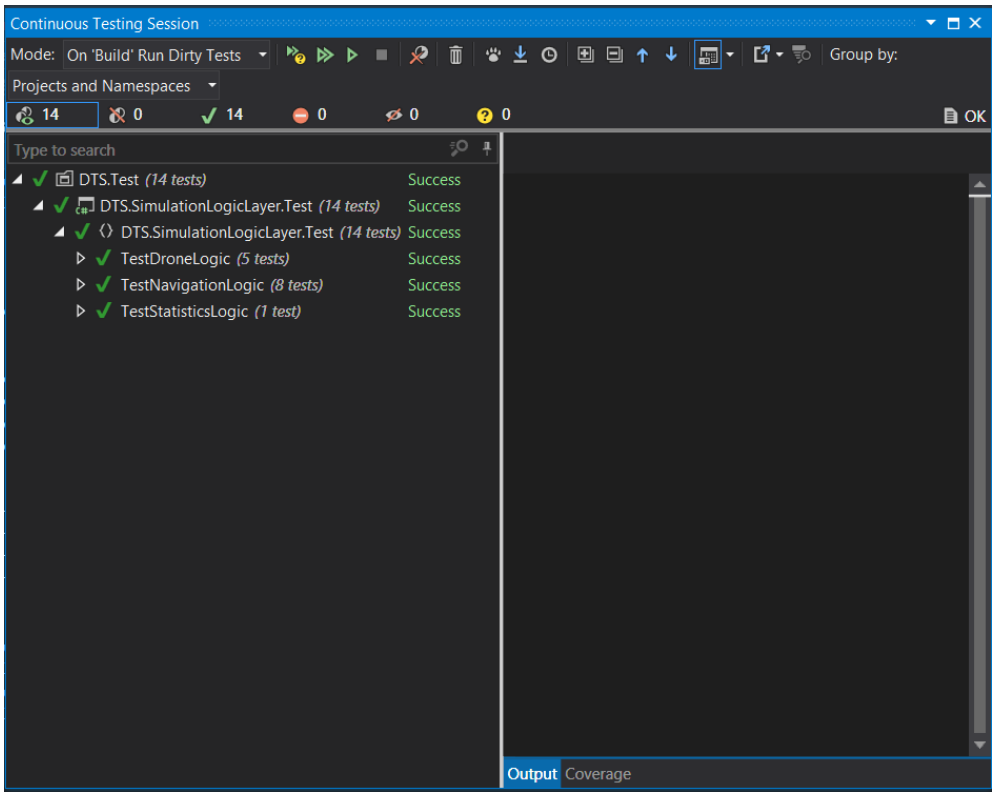
```
[Test]
0 references | Jack, 7 days ago | 1 author, 1 change
public void TestCheckStreets()
{
    //Arrange
    var xCoordExist = new KeyValuePair<float,float>(-131.02f ,-116.02f);
    var xCoordNotExist = new KeyValuePair<float,float>(1,1);
    var zCoordExist = new KeyValuePair<float, float>(-471.16f, -456.16f);
    var zCoordNotExist = new KeyValuePair<float, float>(10, 10);

    //Act
    var resultOne = NavigationLogic.CheckStreet(xCoordExist);
    var resultTwo = NavigationLogic.CheckStreet(xCoordNotExist);
    var resultThree = NavigationLogic.CheckStreet(zCoordExist);
    var resultFour = NavigationLogic.CheckStreet(zCoordNotExist);

    //Assert
    Assert.True(resultOne,"The point does not exist in the database");
    Assert.False(resultTwo,"The point exists in the database");
    Assert.True(resultThree, "The point does not exist in the database");
    Assert.False(resultFour, "The point exists in the database");
}
```

This is an example of a basic unit test I carried out. It is a simple black box testing; asserting does the given inputs provide the correct output.

My testing framework is set up to provide continuous testing each time the code is built. This useful feature alerts me if a change to one area of code affect another area of code. So, if one area causes an error in another area it can be detected.



This is an example of the continuous testing session, if a test does fail it displays the assertion message in right hand window. Assertion messages are an essential part of unit testing which allows the tester to dig in the root of the problem. The right-hand window also displays information about the testing coverage. This is another useful feature which allows me to determine areas where testing coverage needs to be increased. However, this feature does not consider the context of the methods and classes as updating and deleting should not be included in the testing coverage.

7. Problems Solved

7.1 Creating Database Mappings

This was the first area of the project that I sent up. From the outset, I knew a backend was needed to store the data created by and for the simulation. As I mentioned earlier I settled on an SQL Server. This was project was my first experience setting up a back like this. Entity framework has capability for dealing with projects that follow a domain driven design. By this I mean the practice of implementing a software project based on evolving models. This EF feature is known as the Code-First approach, where classes are created first and EF creates the database entities to match on the fly. EF uses the mappings for your domain object in the SQL Server. Thus, the SQL Server side requires very little setup aside from creating the tables and setting primary keys. EF will handle the rest.

As straightforward as this seems I still had some difficulty setting this up. For EF to work mappings between the domain objects and the tables must be set up. To do this I used the Fluent API which provides domain configuration options. Figuring out the correct configurations for the table mappings was a big challenge. The complex relationship between the tables required a lot of research as well as trial and error.

I solved this problem, using a combination of things. The first was the EF tutorials for Fluent API configurations (http://www.entityframeworktutorial.net/code-first/fluent-api-in-code-first.aspx). These provide several examples of how to the relationships between tables. I also used an Entity-Relationship diagram to allow me to visualise the table relationships. Trial and error using try-catch blocks allowed me to view and debug error messages thrown by EF to fix issues with the configuration. I persevered and overcame the difficulties which I was having and this has been of great benefit to the project.

7.2 Changing the Game Engine

A major problem that I faced undertaking this project was the game engine. As per my functional specification I was due to use Unity 3D, a popular game engine that is compatible with .Net, or so I thought. The problem I encountered came when I tried to connect the presentation layer of my architecture to the logic layer and the data access layer. This caused a major error due to Unity using a much lower version of .Net then the class libraries for my project. This was a major set-back as I had made good progress with my application up until then.

Without the latest version of .Net I would not be able to implement various areas of core functionality as well as being unable to create a layer architecture. I decided that a change must be made to my game engine. After exhaustive searching of forums, blog posts and articles I was left with no other option. So, I set about researching alternative game engines, I came across one that worked with the latest versions of .Net. After performing a SWOT analysis, outlined in my blogs, where I weighed the advantages and disadvantages of this new game engine. After a brief period of familiarising myself with the new game engine as well as dealing with the lack of documentation I was back on track. This game engine was also much more

beneficial as it allowed greater control over the game and the game mechanics. Which is a major benefit over Unity 3D.

7.3 Increasing the Navigation Complexity

The final and possibly most important problem which I had to tackle was the increasing complexity of my navigation algorithm. Originally, my map was a simple crossroads made up of 12 navigation points. My algorithm used a simple shortest path algorithm for this which took the quickest route to the target point. I then drastically increased the size of the map to incorporate 144 navigation points. With this increase in size I also increased the complexity by adding constraint on the direction the drones can travel.

This was a major difficulty for me as there were many possible conditions which affected the direction a drone should take. To fix this problem I had to go through trial and error to fully understand how the navigation should work. I started by creating an Excel file to analysis all the conditions that should be accounted for. This can be found in my latest blog entry. I also drew out a grid on paper that contained each of the points, then I wrote out the pseudo code for the algorithm of determining the next point. Testing it with the grid. This technique of writing out the problem before attempting to code it, was a practice I learned from the outset of this course which benefited me greatly when creating this algorithm.

8. Results

Examining the output of this project and looking at the statistics created, I have been able to make various conclusions about the problem I set out to solve. First of all, according to my simulation collisions between the drones do occur. So, in light of this, the goal has been about minimising the number of collisions that occur in the system and maximising distance travelled by the drones. The reason for this is due the fact that more time could produce an optimal navigation algorithm. However, with the time and resources available to me my job is to configure the simulation to optimise the various statistical parameters of the drones.

To this I changed the speed of the drones, i.e. I ran the simulation with the drones at random speeds between 0 and 2 m/s and I ran the simulation with drones at a standard speed of 1 m/s. As expected the standard speed create far better results, with the random speeds causing an exponentially larger amount of collision. So, taking this knowledge I could adjust the parameters of the system, increasing the standard speed to 1.5 m/s and checking how this performed. As expected this performed better again.

Using different parameters like this I can create an optimal simulation. This leads me to a hypothesis which suggests that the faster the drones travel the less collisions occur. However, I imagine this only goes so far until it reaches a threshold.

I also examined the maximum number of drones that the simulations handle without collisions. Could take the most drones with the lowest amount of collisions. This again would indicate that 1.5m/s is the ideal speed for the drones to travel as it has the highest traffic threshold.

9. Future Work

Taking this project into the future, there are many areas that can be improved upon as well as features that can be added. The problem I set out to address was the problem of strict regulations on drones. Through my simulation I made the discovery that collisions do happen and attempted to try and minimise their occurrence.

To completely remove the occurrence of collisions would be a difficult task, however not impossible. My system was designed with extensibility in mind meaning that it was designed

with an eye to making improvements and adding new features. To reduce the number of collisions I would first reduce the work required by the system. By storing previously calculated routes a significant number of operations would be reduced and eventually there may be no need to re-compute the routes. This would allow for heavier algorithms on the drone end. To reduce collisions, I would look at the direction the drones are travelling i.e. the next point they are travelling to. If two drones are travelling towards the same point, then I would slow one of them down. This would add a kind of right-of-way system which would hopefully significantly reduce the number of collisions.

As well as reducing the number of collision, I would also add several events which the drones would have to deal with. This would increase the complexity of the navigation meaning that drones would have to recalculate the route mid-flight. These events may be a crash, a broken drone or even a road closure. However, the generate route algorithm has been designed to allow this to take place as it is a recursive method which can be called from any stage in the route.

There is also a great deal that could be refactored, many of the methods I can see possibly more optimal alternatives but due to the time constraint I cannot implement them. This is not a poor reflection on my code quality but a necessary element of any software development project where the code can be improvement infinitely.