# Introduction

The objectives of this ACW is to research, design and implement a distributed, physically-based modelling demonstrator for a series of gravity wells. The world would consist a large plane that is surrounded by a circular wall. A large number of spheres are placed on surface. The physics simulation would be run across a number of peer as a peer to peer network. Each peer would control an individual gravity well and display a consistent view of the world based on the position of the gravity well. Users could be applied forces to spheres via the gravity well from different peers.

# System Architecture

The UML class diagram of the project is shown in Appendix A to establish a brief overview how classes are developed to provide different functions. Based on their functionality, Classes are mainly divided to 5 components: Framework, Visualisation, Networking, Simulation and Entity Management.

## Class Description

### Framework

Framework classes provide a foundation for the software to work on.

#### WinMain

Software entry point for Window

#### System Class

System Class manage the window framework and maintain the lifecycle of all other classes. When the system initialise different classes, new threads would be created and assigned to specific processors. Detail of threading and process affinity would be discussed in detail later.

#### Config Class

Config Class would keep hold of all the configuration data until the life cycle of the application finished. While it is being initialised, a configuration file would be read in order to obtain the variables to set up the simulation scene. In Appendix B, a template of configuration file has been shown that allow users to set starting values for different variables, such as Screen size, radius of the ground surface, Ports for network connections etc..

Config class has also been used for storing run time changeable values including target updating frequency for graphics, physics and network classes, elasticity, frictional forces and timescale.

#### StepTimer Class

The class provides elapsed time information and setting target elapsed seconds between each update. The timer would run and call the update in order to meet the expected fixed update time as close as possible. For physics, graphic and simulation class, they would initialise their own timer with target elapsed time from the Config class. As a result, users would be able to control different updating frequency on rendering, simulating and networking.

### Visualisation

#### Graphic Class

The Graphic Class is developed to handle the visualisation of the physics modelling. To visualise objects on the screen, the Graphic Class manage the Direct3D Class and Camera Class and monitor user input at each frame.

#### D3D Class

This is a class to handle all the Direct3D functions. It maintains the ID3D11Device and Device Context and set up the characteristics of the swap chain in order to create a swap chain instance.

#### Camera Class

The Camera Class handles the view matrix and the location of camera. If the camera is updated to a new location or rotation, a new view matrix will be calculated.

### Networking

#### Network Class

Network class handle all the connections to the peer to peer network, including communicate with other peers with TCP and UDP protocol, Data packing and unpacking.

### Simulation

#### Simulation Class

Simulation Class handle physics simulating including applying gravity, applying forces from gravity wells, collision detection and generate contact Manifold.

#### Contact Manifold Class

This class is designed to resolve the contact Manifold generated by the Simulation Class. The detail of Collision detection and response in this application would be discussed in later session.

### Entity Management

#### Ball Manager

This class is designed to store the data of ball in the simulating world.

#### Gravity Manager

This class is designed to store the data of gravity well including other peer’s gravity well.

## Threading

In order to increase the performance of the software, major components would be needed to operate asynchronously. Therefore multithreading has been introduced to the system.

Firstly visualisation, simulation and networking have been separated into different threads after the system has finished the initialisation. As shown in Appendix C&D, the threads would run until user shutdown the program. Besides separate the main components, more threading have been used for small task in order to reduce the duration for each update.

As illustrated in Appendix C, threads have been created to handle communication with other peers in network Class:

* UDP Broadcasting
* UDP Listening
* TCP sending
* TCP receiving

The usage of threads in Simulation class has also been shown in Appendix D:

* Collision Detection:
  + Ground-Ball
  + Wall-Ball
  + Ball-Ball
* Collision Resolving

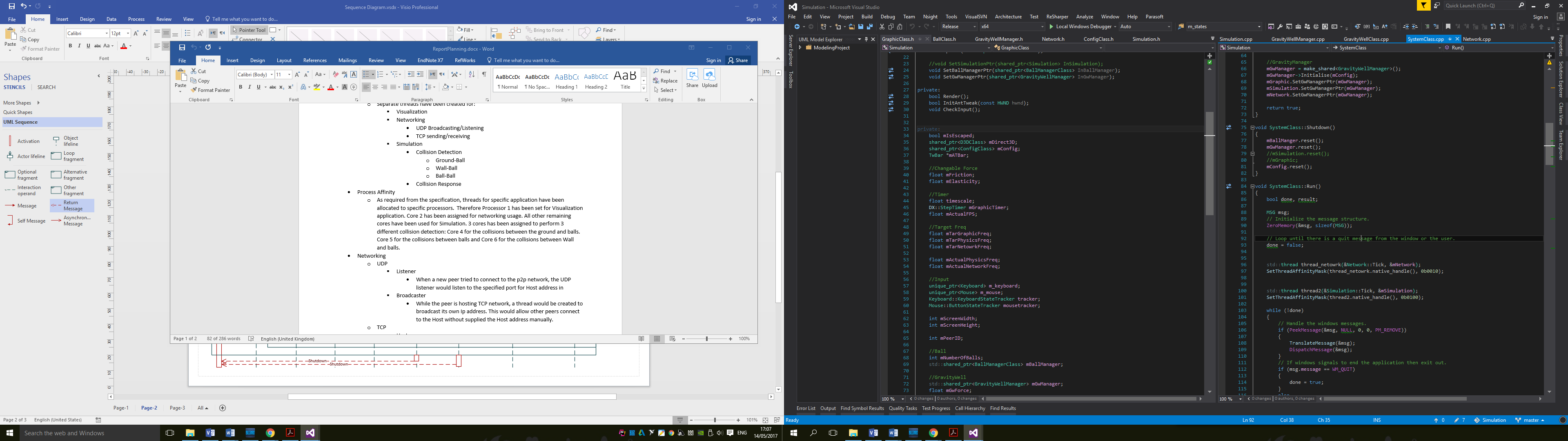
### Race condition

Race conditions would occur when two threads access a shared variable at the same time. To prevent race condition in the simulating system, mutex and lock\_guard have been implemented in the Manager Class and Config Class. With lock\_guard, resources could not be accessed while the mutex is owned by other threads.

In Collision detection, three different collision would be detect at the same time and add new contact point to the manifold and result in race condition. Therefore they would be setting up a temporary local manifold to store collision detected. Then three temporary manifold would be added to the main manifold in one. Similar approach has been used in networking for updating ball information from other peer after each simulation loop.

### Process Affinity

As required from the specification, threads for specific application have been allocated to specific processors by using SetThreadAffinityMask as shown in below diagram.

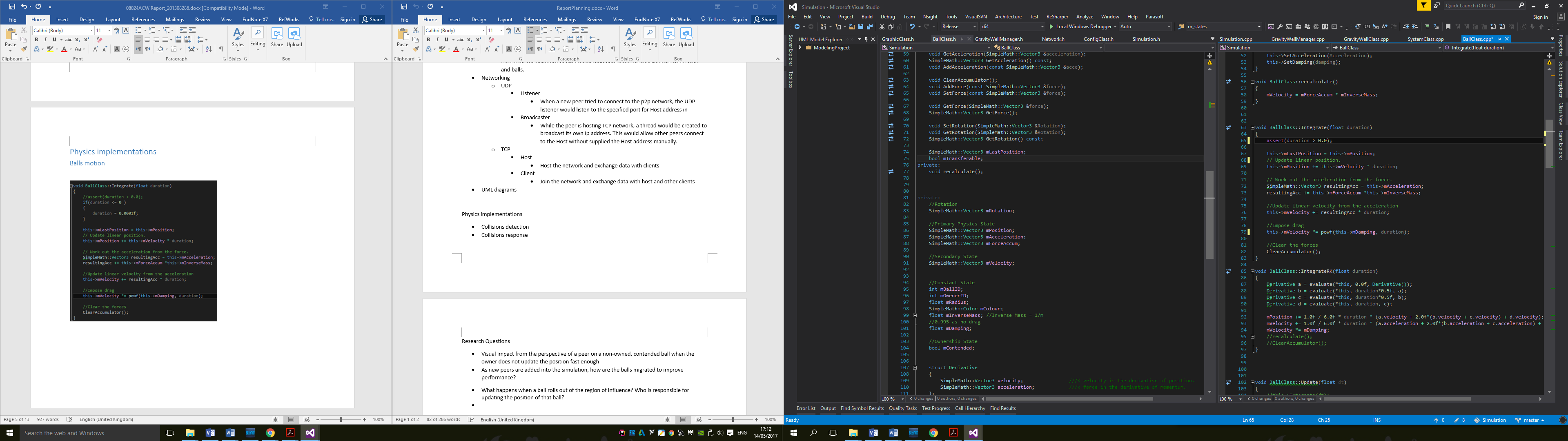


Therefore Processor 1 has been set for Visualization application. Core 2 has been assigned for networking usage. All other remaining cores have been used for Simulation. 3 cores has been assigned to perform 3 different collision detection: Core 4 for the collisions between the ground and balls. Core 5 for the collisions between balls and Core 6 for the collisions between Wall and balls.

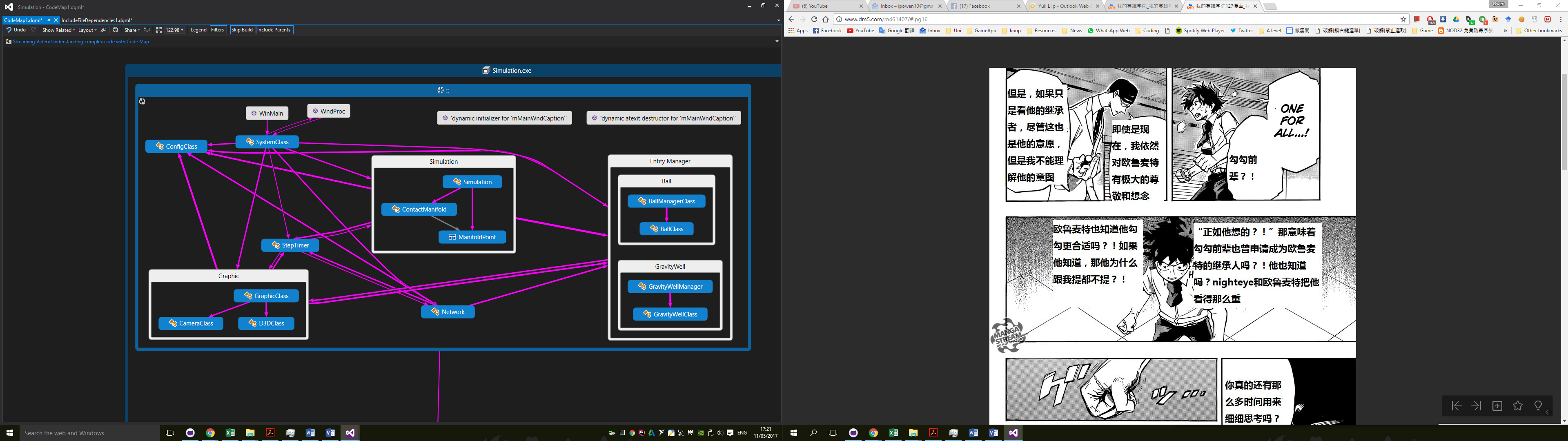
# Physics implementations

## Balls motion implementation

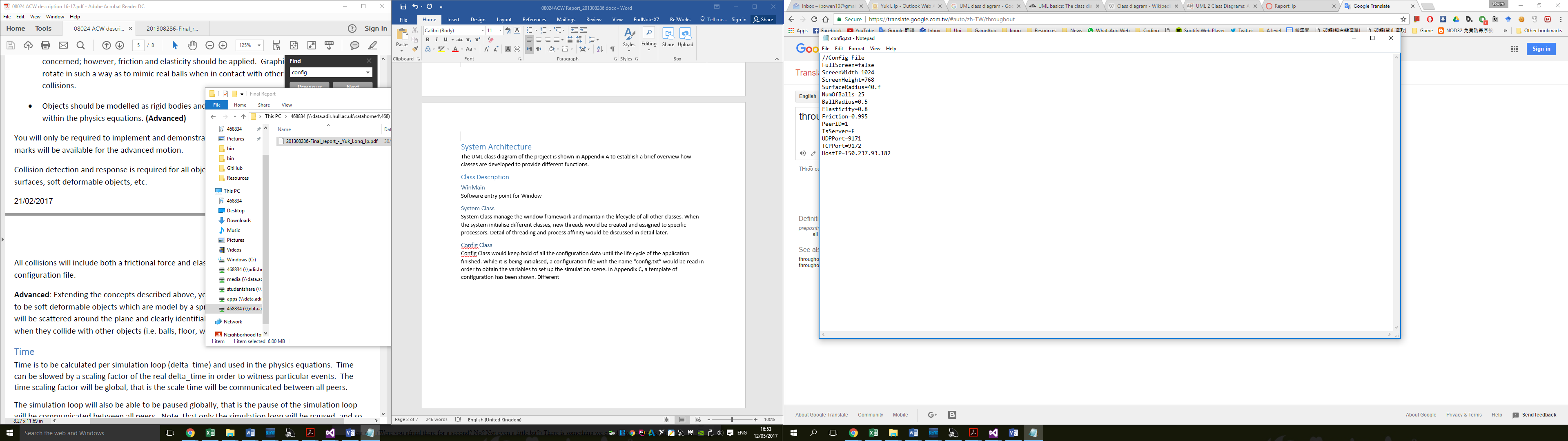
Below diagram has shown the implemenation of the ball montion.



# Appendix A: UML Class Diagram



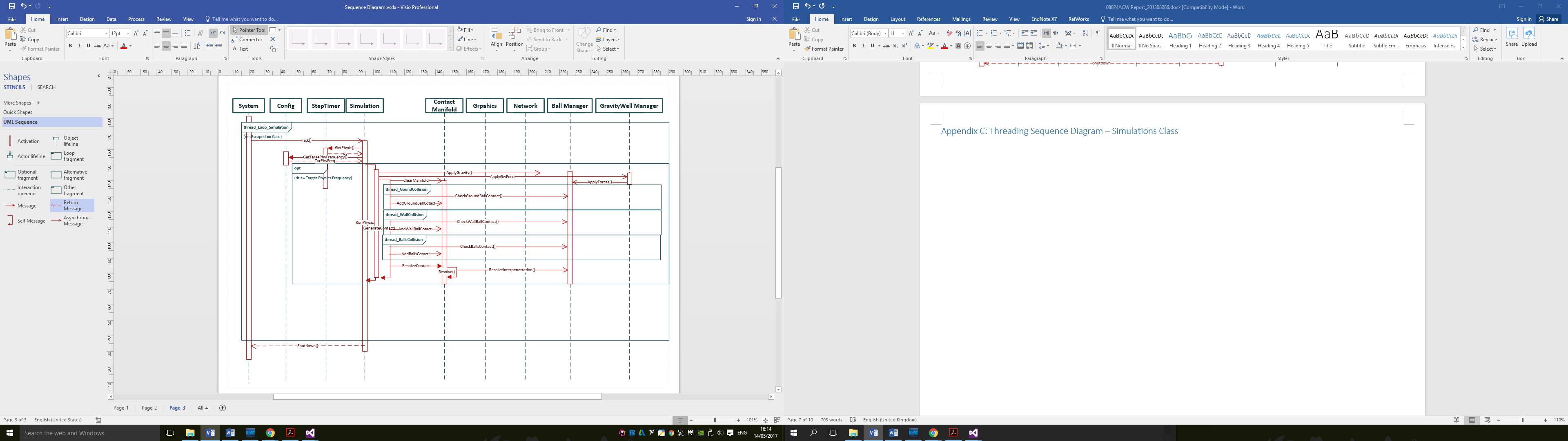
# Appendix B: Configuration file Template



# Appendix C: Threading Sequence Diagram – Network & Graphics Class

# 

# Appendix D: Threading Sequence Diagram – Simulations Class



Appendix B: Sequence Diagram

# Appendix C: Configuration file template

