

University of Bolton

|  |  |
| --- | --- |
| **Student Name** | Thomas Plumpton |
| **Student Number** | 1500936 |
| **Student Email** | TP2AMT@Bolton.ac.uk |

|  |  |
| --- | --- |
| **Programme** | Computing |
| **Module Code** | CPU6001 |
| **Module Title** | Major Project |
| **Assignment Title** | Final Report - Dissertation |
| **Date of Submission** | 1st April 2018 |
| **Word Count** | 17367 |

|  |  |
| --- | --- |
| **Tutor Name** | Amanda Dewhurst |
| **Project Supervisor** | Abdul Razak |
| **Project Title** | Analysing the Random Nature of Social Media |

# Abstract

This report encompasses the mathematical study of randomness and its application in real world industries. The study employs the use of several technical elements, the majority of which are programming orientated. Randomness is used in a myriad of industries. Including; Video Games; Politics; Science; Cryptography and more. It is important to continuously alter the method in which randomness is generated in order to make it harder to predict. This report describes the theory of a new method and determining its nature of randomness.

In this report I have documented background research that was conducted before development had started and research that was discovered during the project itself. Each stage, including; the approach, methodology, analysis, design, development, testing, evaluation and conclusion is described and illustrated throughout the report.

# Acknowledgements

My final year at University has been a difficult one. I may have written more reports and conducted more research during this seven months than I have in the entirety of my life. I worked very hard on this project and invested a lot of time. However, I would like to reflect on the people whole supported me throughout this time as they helped me greatly.

First a foremost, I’d like to thank my project supervisor, Abdul Razak. He was tasked with the responsibility of overseeing my Major Project development throughout the final year. We agreed on weekly one-to-one meetings to regularly assess the development of the project. He provided useful information and advice and was always lightning fast on the email responses.

I would also like to thank Andrew Parker, my Software Engineering & Advanced Databases lecturer. During the final semester of the second year, we started thinking of ideas for our Major Project. I had a few in mind, but wanted something unique, instead of an application or a website. Given his past experience working at IBM and with Java, he mentioned a programming game called Robocode. Vinesh Mistry, a fellow student and friend, also mentioned controlling the game with an external influence like Twitter. This idea formed the start of my project which I then took into my own hands.

Finally, I would like to thank all my tutors and University peers as a collective. They are always happy to meet up and discuss any information or issues I may have. Furthermore, not only were all able to support one another with our work, but we could also talk about things other than just our assignments.

Thank you very much, everyone!

Thomas Plumpton

December 30, 2017.

Table of Contents

[Abstract 2](#_Toc509568615)

[Acknowledgements 2](#_Toc509568616)

[Table of Figures 5](#_Toc509568617)

[Formatting Key 6](#_Toc509568618)

[[1] Introduction 6](#_Toc509568619)

[[2] Background Research 6](#_Toc509568620)

[[2.1] Twitter 6](#_Toc509568621)

[[2.1.1] Twitter API & Rate Limiting 7](#_Toc509568622)

[[2.1.2] Importance of Sample Size 7](#_Toc509568623)

[[2.1.3] Frequency of Letters in the English Language 7](#_Toc509568624)

[[2.2.2] Emoticons, Emoji & Fitzpatrick Modifiers 8](#_Toc509568625)

[[2.3] Robocode 10](#_Toc509568626)

[[2.3.1] Game Physics, Coordinates & Directional Systems 10](#_Toc509568627)

[[2.3.2] Anatomy of a Robot 11](#_Toc509568628)

[[2.3.3] Robocode Processing Loop 11](#_Toc509568629)

[[2.3.4] Creating My Own Robot 12](#_Toc509568630)

[[2.3.5] Robots vs Advanced Robots 13](#_Toc509568631)

[[2.4] Data Visualisation & NodeJS 14](#_Toc509568632)

[[2.4.1] Past Experience 14](#_Toc509568633)

[[2.4.2] Model, View, Controller Architecture 14](#_Toc509568634)

[[2.5] Mathematics of Randomness 14](#_Toc509568635)

[[3] Approach & Methodology 14](#_Toc509568636)

[[3.1] Agile Software Development 14](#_Toc509568637)

[[4] Analysis & Design 15](#_Toc509568638)

[[4.1] Twitter 15](#_Toc509568639)

[[4.1.1] Java Standard Edition 15](#_Toc509568640)

[[4.1.2] Association, Aggregation & Composition 16](#_Toc509568641)

[[4.1.3] Single Responsibility Principle (SRP) 17](#_Toc509568642)

[[4.1.4] Formatting & Sanitisation of Tweets 17](#_Toc509568643)

[[4.1.5] Java Documentation (Javadoc) 19](#_Toc509568644)

[[4.2] Robocode Programming Game 20](#_Toc509568645)

[[4.2.1] Parsing Tweets from the TweetReader 20](#_Toc509568646)

[[4.2.2] Battle Results Data Structure 21](#_Toc509568647)

[[4.3] Data Visualisation & NodeJS 22](#_Toc509568648)

[[4.3.1] MVC Folder Structure 22](#_Toc509568649)

[[5] Program Development 22](#_Toc509568650)

[[5.1] Twitter 23](#_Toc509568651)

[[5.1.1] Twitter Authentication & API Connection 23](#_Toc509568652)

[[5.1.2] Mongo Atlas Cluster Connection & Inserting Documents 25](#_Toc509568653)

[[5.1.3] Tweet String Sanitisation 29](#_Toc509568654)

[[5.1.4] Tweet String Parsing 30](#_Toc509568655)

[[5.2] Robocode 31](#_Toc509568656)

[[5.2.1] Integrating Robocode with IntelliJ 31](#_Toc509568657)

[[5.2.2] Creating an Advanced Robot Class 32](#_Toc509568658)

[[5.2.3] Overcoming Security Exceptions 34](#_Toc509568659)

[[5.2.4] Controlling the Robots 35](#_Toc509568660)

[[5.3] Data Visualisation & NodeJS 35](#_Toc509568661)

[[5.3.1] Configuring NodeJS & ExpressJS Framework 35](#_Toc509568662)

[[5.3.2] Automating NodeJS Runtime in JetBrains WebStorm 37](#_Toc509568663)

[[5.3.3] Creating the Page & Integrating D3.JS 38](#_Toc509568664)

[[5.4] Mathematical Analysis 38](#_Toc509568665)

[[6] Testing Phase 38](#_Toc509568666)

[[6.1] Twitter 38](#_Toc509568667)

[[6.1.1] Accessing the MongoDB Atlas Cluster 38](#_Toc509568668)

[[6.1.2] Committing Twitter Status Objects to the MongoDB Atlas Cluster 39](#_Toc509568669)

[[6.1.3] Sanitising Tweets 39](#_Toc509568670)

[[6.2] Robocode 40](#_Toc509568671)

[[6.2.1] Adding Custom Robots to Robocode 40](#_Toc509568672)

[[6.2.2] Robot Controller Random Values 40](#_Toc509568673)

[[7] Results 40](#_Toc509568674)

[[8] Evaluation 41](#_Toc509568675)

[[9] Conclusions 41](#_Toc509568676)

[[10] Glossary of Terms 41](#_Toc509568677)

[[11] Bibliography 42](#_Toc509568678)

[[12] Appendices 48](#_Toc509568679)

[[12.1] Terms of Reference 48](#_Toc509568680)

[[12.2] Literature Review 50](#_Toc509568681)

[[12.3] What is Twitch? 65](#_Toc509568682)

[[12.4] What is Robocode? 65](#_Toc509568683)

[[12.5] What is Twitch Plays Pokémon? 65](#_Toc509568684)

# Table of Figures

[Figure 1 - Frequency of Letters (Oxford Dictionary vs Tweet Sample) 8](#_Toc509568569)

[Figure 2 - Emoji Operating System Interpretation 9](#_Toc509568570)

[Figure 3 - Emoji Usage Statistics (U.S 2016) 9](#_Toc509568571)

[Figure 4 - Fitzpatrick Skin Phototype Scale 10](#_Toc509568572)

[Figure 5 - Robocode Directional & Rotational Convention 10](#_Toc509568573)

[Figure 6 - Robocode Robot Anatomy 11](#_Toc509568574)

[Figure 7 - Avoiding Firing Pitfall in Robocode Processing Loop 12](#_Toc509568575)

[Figure 8 - MajorProjectv2 Compile Output Path 13](#_Toc509568576)

[Figure 9 - Java SE 8 Lambda Expression 16](#_Toc509568577)

[Figure 10 - Original Twitter Java Program UML 17](#_Toc509568578)

[Figure 11 - Twitter Tweet JSON Object 18](#_Toc509568579)

[Figure 12 - Twitter Hexadecimal Byte Sequence Differentiation 18](#_Toc509568580)

[Figure 13 - Initial Design of Tweet Object Document 19](#_Toc509568581)

[Figure 14 - Javadoc Example Method 19](#_Toc509568582)

[Figure 15 - IntelliJ Contextual API Documentation 20](#_Toc509568583)

[Figure 16 - Robocode Results Data Structure Example 22](#_Toc509568584)

[Figure 17 - Adding Twitter4J .JAR to IntelliJ Project 23](#_Toc509568585)

[Figure 18 - Java External Library Import Statements 23](#_Toc509568586)

[Figure 19 - Twitter Developer Application Access Tokens 24](#_Toc509568587)

[Figure 20 - Twitter ConfigurationBuilder Method 24](#_Toc509568588)

[Figure 21 - MongoDB Atlas Cluster 25](#_Toc509568589)

[Figure 22 - Asynchronous CompletableFuture Method 27](#_Toc509568590)

[Figure 23 - Private Visibility MongoConnection Member 28](#_Toc509568591)

[Figure 24 - User Defined MongoConnection Class 28](#_Toc509568592)

[Figure 25 - MongoDB Compass Collection with 6400 Tweets Inserted 29](#_Toc509568593)

[Figure 26 - TweetSanitiser Class Snippet 30](#_Toc509568594)

[Figure 27 - TweetParser Class 30](#_Toc509568595)

[Figure 28 - Start Robocode Application Run Configuration 31](#_Toc509568596)

[Figure 29 - TwitterRobot Class with Default Template 32](#_Toc509568597)

[Figure 30 - IntelliJ Project Structure Pane 33](#_Toc509568598)

[Figure 31 - Adding TwitterRobot's to Battle 33](#_Toc509568599)

[Figure 32 - Robocode Security Manager Destroying Robots 34](#_Toc509568600)

[Figure 33 - Robocode Security Manager Logging Violation 34](#_Toc509568601)

[Figure 34 - Configuring HTTP Server in NodeJS 35](#_Toc509568602)

[Figure 35 - Installing ExpressJS Framework via NPM Command Line 35](#_Toc509568603)

[Figure 36 - Configuring Middleware Stack & Page Routing in NodeJS 36](#_Toc509568604)

[Figure 37 - NodeJS package.json file contents 36](#_Toc509568605)

[Figure 38 - Running NodeJS Application from NPM Command Line 37](#_Toc509568606)

[Figure 39 - JetBrains WebStorm NodeJS Application Run Configuration 37](#_Toc509568607)

[Figure 40 - Running NodeJS Application from WebStorm 38](#_Toc509568608)

[Figure 41 - Testing with MongoConnection Main Method 38](#_Toc509568609)

[Figure 42 - MongoDB Compass Test Document 39](#_Toc509568610)

[Figure 43 – MongoDB Compass Displaying 19200 Documents 39](#_Toc509568611)

[Figure 44 - TweetSanitiser JUnit Test Method - downloadTweets() 40](#_Toc509568612)

[Figure 45 - TweetSanitiser JUnit Test Pass 40](#_Toc509568613)

[Figure 46 - TweetSanitiser JUnit Test Fail 40](#_Toc509568614)

# Formatting Key

|  |  |
| --- | --- |
| **Bold Purple** | Referencing method signatures from the project source code |
| Underlined Blue | Hyperlinks to chapters or sections in this report. Ctrl + Click to use |
| [X] 16pt Calibri | The largest blue headings preceded by numbers are main chapters |
| [X.Y] 13pt Calibri | The smaller blue headings are sub-chapters of the main ones |
| [X.Y.Z] 12pt Calibri | The smallest blue headings are detailed sub-chapters of the above |

# [1] Introduction

Randomness is a mathematical concept that is commonly misunderstood. Computer-generated random numbers are not *truly* random, they are actually *pseudo*-random. This means that although they appear to be random, they are really pre-determined. Pseudo-Random Number Generation (PRNG) is efficient and deterministic, however, these are not desirable attributes in scenarios that require True-Random Number Generation (TRNG). Such scenarios include the generation of data encryption keys and cryptography or simulation of natural phenomena like weather.

This project aims to analyse and determine the random nature of the social media platform ‘Twitter’ by parsing Tweets and using them to control virtual robots in a game called ‘Robocode’. The data produced will be analysed to determine whether it is pseudo or truly random in an attempt to create a new method of generating truly random numbers. The inspiration for this project originates from a social experiment called ‘Twitch Play Pokémon’.

# [2] Background Research

This section documents research that was conducted both at the outset, or during development as new information was discovered. A large quantity of this was done prior to the project start date and is documented in the [Terms of Reference](#_[13.1]_Terms_of). However, only a certain degree of research findings was known at that point and so this section is a continuation of that investigation during the project.

## [2.1] Twitter

Twitter Inc. provide developer access to their API which allows registered developers to write server-side scripts that interact with the Twitter platform and access data. Research has shown some limitations and constraints set by Twitter Inc and raises other considerations as a result.

### [2.1.1] Twitter API & Rate Limiting

The Twitter API has a restriction called ‘Rate Limiting’. This means that each user and access token have 15 requests per rate limit window. The HTTP GET request window is 15 minutes. (Twitter, 2017) Furthermore, this particular method only allows a maximum of 3200 objects to be returned per API call. Therefore, assuming every account used has a posted at least 3200 tweets, a maximum of 48,000 (15 x 3200) Tweets can downloaded per 15 minutes.

To overcome the obstacle of rate limiting, the official Twitter documentation suggests methods such as caching, or streaming can be used. ‘Caching’ is the process of temporarily storing data locally to a system, so that a website or application doesn’t need to download resources or assets each time it accessed. Caching increases the speed and efficiency of the application as it can access the locally stored data (Bell, 2017). Whereas, the Streaming API, also provided by Twitter Inc., has rate limiting and access levels that are appropriate for long-lived connections. (TwitterInc., 2018)

Given the mathematical nature of the project, a large dataset is desired in order to ensure the accuracy of the outcome.

### [2.1.2] Importance of Sample Size

Sample size is important as it can determine the reliability and margin of error produced by the study. If the size is small, a larger sample size is required otherwise the effect will be masked by the randomness of the samples. Conversely, a larger sample size provides more reliable results with greater precision. Therefore, the more tweets that can be downloaded and parsed, the more reliable the data produced will be (Anon., 2017).

Furthermore, due to an increase in the accuracy of the data, database analysis will be a lot more reliable. This includes producing statistics such as the average tweets length, usage of emoji and frequency of letters used.

### [2.1.3] Frequency of Letters in the English Language

One factor that immediately comes to mind with regards to the random nature of this project is

the alphabet of the language used. The problem is that the frequency distribution of letters introduced a bias into the system. The graph below shows the relative frequency of letters used from two sources. The first is from the Concise Oxford Dictionary and you can see that the vowels are some of the highest, where other letters such as J, X and Z are not used even half as much. (Anon., n.d.) The second source is sampled from 13,200 downloaded Tweets from different Twitter accounts. We can see that there is some deviation from all the words in the dictionary, but nonetheless, there is still a very similar distribution.

Figure 1 - Frequency of Letters (Oxford Dictionary vs Tweet Sample)

This was considered at the outset of the project and is the reason why Robocode was introduced into the system. By parsing the characters in the tweets, the behaviour of the robot and results of the match is what will be analysed to determine the random nature. Technically, a higher frequency of one character will mean a higher frequency of parsing on the character. But the algorithm used to randomise the values of the robot parameters will be designed in such a way to reduce the amount of bias as much as possible.

### [2.2.2] Emoticons, Emoji & Fitzpatrick Modifiers

Emoticons are typographic displays of facial representation that are used to convey emotion in a text only medium. For example, a basic smiley face “:)”. Unlike emoticons, emoji are actual pictures displayed in-line with a message, typically used on social media. Emoji are extensions to the Unicode character set that is used by most operating systems. Given that each emoji has its own Unicode value, each operating system interprets and displays the emoji differently. For example, the figure below shows the dancer emoji and how various operating systems and platforms interpret the code. (Hern, 2015)

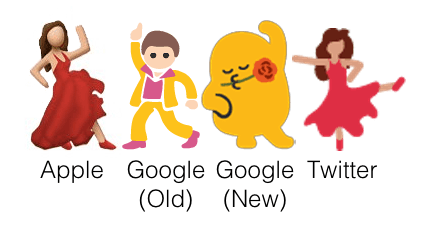


Figure 2 - Emoji Operating System Interpretation

The usage of emojis has drastically increased over the last few years. Roughly 92% of the online population uses them and brands have also standard incorporating them into marketing. According to Swyft Media, in the U.S in 2016, 41.5 billion messages and 6 billion emojis were sent every day on mobile messaging apps. Nearly half of all Instagram posts contains at least one emoji, and Facebook has change it’s ‘reactions’ feature to emojis. Research from the Emogi’s Emoji Report showed that 63 percent of people are use emojis frequently, 29 percent are occasional users, and 8 percent don’t use them at all. (Dua, 2016) (Emogi, 2016)

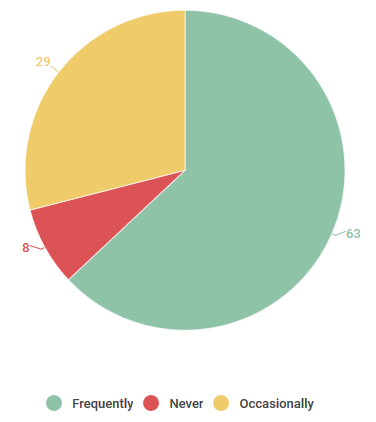


Figure 3 - Emoji Usage Statistics (U.S 2016)

There are in total, 2,666 emojis in the Unicode Standard as of June 2017. This number has increased due to the introduction of the Fitzpatrick Scale of skin tones. Due to ethnic and cultural issues surrounding emojis, human or body orientated emojis have received updated that add various colours of skin tone from the Fitzpatrick Scale. (Emojipedia, 2018)

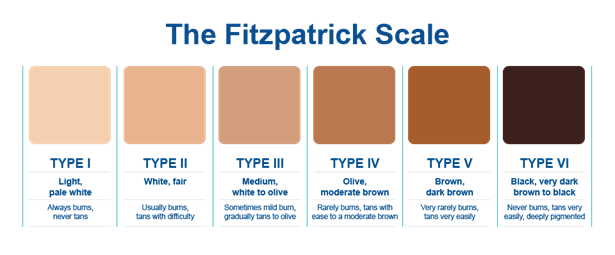


Figure 4 - Fitzpatrick Skin Phototype Scale

Due to this surge of emoji usage, subject to the Twitter user, a lot of Tweets downloaded for this project will have emoji’s in their text element. This adds a huge level of complexity to the structure of the Tweets as the text element would need to be queried for Unicode symbols and then cross-referenced across a library of Unicode emoji’s to detect which image they should portray.

## [2.3] Robocode

### [2.3.1] Game Physics, Coordinates & Directional Systems

Robocode uses the Cartesian coordinate system, which means that that the (0, 0) coordinate is located at the bottom-left corner of the battlefield. The rotational directional system uses a clockwise direction convention where 0°/360° is north, 90° is east, 180° is south, and 270° is west.

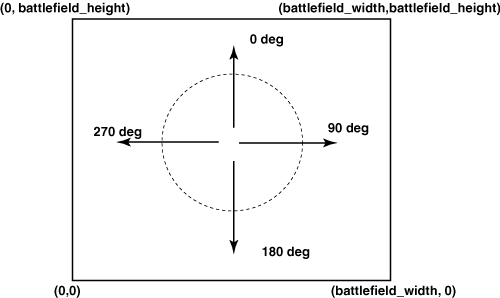


Figure 5 - Robocode Directional & Rotational Convention

Robocode time is measured in "ticks". Each robot gets one turn per tick. Regarding distance measurement, Robocode's distance units are measured with double precision, so you can move a fraction of a unit. Generally, 1 Robocode distance unit is one pixel, except when Robocode automatically scales down battles to fit on the screen. Acceleration is measured at the rate of one pixel per turn, every turn. Robots decelerate at the rate of two pixels per turn, every turn. The game itself determines the acceleration based on the distance the robot is trying to move. Velocity is measured as (acceleration x time). Velocity cannot exceed eight pixels per turn. Given that velocity is a vector, meaning it has both magnitude and direction, Robocode simply assumes the direction of the vector to be the robot’s heading. Distance is again based off real physics using (velocity x time). (Robowiki, 2017)

### [2.3.2] Anatomy of a Robot

Robocode robots consist of three individual parts. The body, which carries the gun with the radar on top. The body is used for moving the robot back and forth, as well as turning left or right. The gun, which is mounted on the body and is used for firing energy bullets. The gun can turn left or right and carries the radar on top. Finally, the radar, which is mounted on the gun. It is used to scan for other robots when moved. It can turn left or right and generates **onScannedRobot()** events when robots are detected. (Robowiki, 2017)

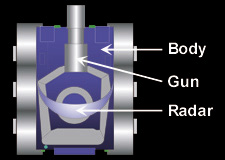


Figure 6 - Robocode Robot Anatomy

### [2.3.3] Robocode Processing Loop

Robocode runs a series of consecutive processing actions, indefinitely, in a loop until the match is over. It is important to understand how this loop works in order to integrate my own programs code with Robocode. The order that Robocode runs is as follows:

1. The battle view is (re)painted.
2. All robots execute their code until they take action (and are then paused).
3. Time is updated (time increments by one).
4. All bullets move (including the bullet fired in the last tick) and are checked for collisions.
5. All robots move (Gun, Radar, Heading, Acceleration, Velocity, Distance, in that order. Gun heat is also decreased in this step).
6. All robots perform scans (and collect team messages).
7. All robots are resumed to take new action.
8. Each robot processes its event queue.

One problem with this loop is that because bullets are fired before the gun moved, calling **setFire()** will cause the bullet to leave the gun at its current heading. For most of the time, this error will be negligible, but it can nonetheless affect robot accuracy. The Robocode wiki provides an example code structure on how to get the bullet to leave after turning the gun. (Robowiki, 2017)



Figure 7 - Avoiding Firing Pitfall in Robocode Processing Loop

### [2.3.4] Creating My Own Robot

Normally, when creating your own robot in Robocode, you would use either the built-in editor, or an IDE of your choice to write the code that dictates how the robot acts in certain situations. You would declare a new class, extend either the Robot or AdvancedRobot interface and add a **run()** method that loops and executes every game tick.

However, because Robocode simply accepts a .class file inside the ‘robots’ directory of the source file root, it makes it very difficult to develop the specific custom functionality for this project. Robocode handles the compilation of the class into a .java file and then manages the robots itself. This means that when writing my TwitterRobot class, the one that will be used with random values from Twitter, I must consider that an Object of that class can never be instantiated in my code. This prevents me from invoking both accessor and mutator methods upon the TwitterBot Objects, as it would never *really* exist from the developers’ point of view. This adds a huge layer of academic challenge to the project as it strips away a lot of the core features and benefits of using an object-orientated programming language such as Java.

To overcome the .class file directory problem, I copied the entire Robocode source file master repository into my IntelliJ project and then set the project ‘production out’ directory to ‘robocode\_master/robots’ so that upon building the project, IntelliJ would put the .class files from my project source into the correct ‘robots’ folder so Robocode can see them.

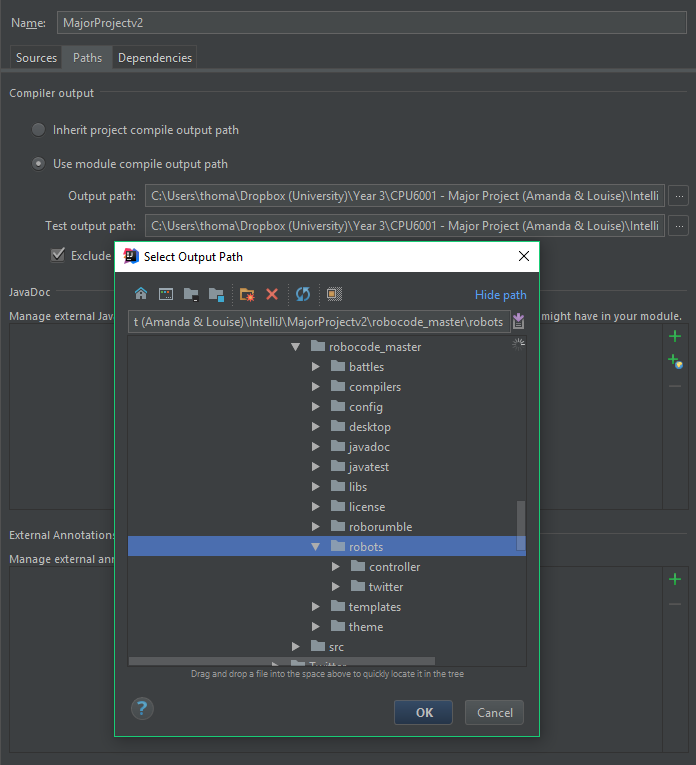


Figure 8 - MajorProjectv2 Compile Output Path

### [2.3.5] Robots vs Advanced Robots

The Robocode API provides two classes which can be extended by user created Robot classes. These are ‘Robot’ and ‘AdvancedRobot’. How do they differ?

**Robot Class**

The standard ‘Robot’ class allows only one action per turn. It cannot access the **setXXX()** methods. For example, a class inheriting from the Robot class can only call **fire()**, not **setFire()**. (Robowiki, 2017)

**AdvancedRobot Class**

The more complex ‘AdvancedRobot’ class allows multiple actions per turn. It also has the ability to call **setXXX()** methods. These types of methods end the turn when you call them and block the robot’s thread until the command finishes. This class also implements several interfaces that extend the AdvancedRobots functionality even more, making it a lot more versatile than the regular Robot interface. However, there are no significant advantages on the battlefield, these classes simply change the development of the robot. You can view the AdvancedRobot Class Documentation at the API [here](http://robocode.sourceforge.net/docs/robocode/robocode/AdvancedRobot.html). (Mathew Nelson, 2017)

## [2.4] Data Visualisation & NodeJS

### [2.4.1] Past Experience

I have had lots of prior experience with NodeJS and the MEAN Stack in general. The MEAN Stack is a collection of JavaScript technologies including MongoDB, ExpressJS, AngularJS and NodeJS. MongoDB is a document-based database models. ExpressJS is a framework built for NodeJS. AngularJS is a JavaScript library used for front-end development. Finally, NodeJS is a runtime environment built upon Google Chromes’ V8 JavaScript engine.

I deliberately chose this stack of technologies for this part of the project as I was familiar with it. I felt that the rest of the project provided more than enough academic challenge and that picking something entirely new for this part would be ultimately detrimental to the project.

### [2.4.2] Model, View, Controller Architecture

MVC, or Model-View-Controller is an architecture model used in software engineering and programming. It splits the contents of an application into the three categories. Models represent knowledge, usually via an object or a structure of objects. There should be a one-to-one correspondence between the models’ parts, and how that object would be perceived by the owned in the real world. A view is a visual representation of its model. A view queries its model for the necessary data required for presentation and may update it by sending appropriate messages. Finally, a controller is the link between a user and the system. It provides them with an input by arranging relevant views in such a way that it provides means for user output. This is usually done via menus, buttons and other means of giving commands and data. The controller receives user output and translates it into the appropriate messages for the views. (Atwood, 2008)

## [2.5] Mathematics of Randomness

# [3] Approach & Methodology

## [3.1] Agile Software Development

The project was split into 4 major sections in order to make it more manageable and adhere to the chosen ‘Agile’ development methodology. The project was split as follows;

[**Section 1**] Twitter & String Parsing

Twitter.com was the source of control and randomness for the Robocode Java code. Functionality was needed in order to access Tweets from given account usernames. The Tweets would then need to be parsed so that Robocode could send instructions to the virtual robots. (Anon., 2017)

[**Section 2**] Robocode & Java

Robocode was the intermediary platform that was fed input from Twitter and would provide the output of data to be analysed. Research into the workings of Robocode was necessary to understand how the underlying Java code controls the robots. (Anon., 2017)

[**Section 3**] Mathematics of Randomness

The study of randomness is the core element of the project. Understanding the differences between Pseudo-Random Number Generation (PRNG) and Truly-Random Number Generation (TRNG) was crucial, as differentiating between them is a common misconception. Research into further understanding how randomness differs on a mathematical level was required as it was ultimately used to conclude the project. (Anon., 2017)

[**Section 4**] Data Visualisation & JavaScript

Visualising the data was required in order to graph the final data. Visualising the data made it possible to mathematically analyse the results and produce a conclusion. Research into dynamically graphing data with the D3.JS JavaScript library had to be done. (Bostock, 2017)

Research involved creating test projects within the relevant platforms and writing the basic functionality to compete the given task.

# [4] Analysis & Design

This section illustrates the design stages of the project whereby I considered future development and how, therefore, programs should be designed. There are sub-chapter for each stage in the agile software development process, all of which have their own respective sections.

## [4.1] Twitter

### [4.1.1] Java Standard Edition

Oracle is a global provider of enterprise cloud computing solutions and provides leading-edge capabilities in software, platform, infrastructure and data as a service. (Oracle, 2017) They acquired Sun Microsystems in 2010 along with their Java platform which was original created by James Gosling and released in 1995. The Java platform has evolved over the last 20 years as teams of software engineers have improved and extended the languages functionality. The original JDK Beta, 1.0 and 1.1 were released in 1995-1997. The next release was renamed to J2SE through versions 1.2 to 5.0 in 2004. Finally, the Java SE (Standard Edition) version 6 was released in 2006 and it has been incrementing ever since. Currently, as of writing this report, the latest commercially available and stable version is Java SE 9.

Each SE Version of the language introduced new coding standards, programming syntax and functionality. As software engineering practices and methods evolve, so does the etiquette. To illustrate, Java SE 8 introduced Lambda expressions which is Java’s first step into functional programming. A Java lambda expression is a function that can created without belong to any class and can therefore be passed around as if it was an object. To illustrate, the following **.addStateListener()** method invocation contains a lambda expression. The expression is matched again the parameter type of the interface that the StateOwner Class implements.



Figure 9 - Java SE 8 Lambda Expression

The hypothetical Interface in the above example has a single method that is called whenever the state of the StateOwner Class changes. Previously, in Java SE 7, you would have to implement the interface and create use an event listener with an anonymous interface implementation. The Java compiler is essentially using type inference on the method signature to guess the parameter type. (Jenkov, 2015)

I have adhered the latest Java standards where possible in my program.

### [4.1.2] Association, Aggregation & Composition

There are myriad types of relationships in Object-Orientated Programming (OOP) and by extension, Universal Markup Language (UML) types. Three commonly associated relations ships are Association, Aggregation and Composition. Although technically, Aggregation and Composition are subsets of association, meaning they are specific cases of association.

Both Aggregation and Composition refer to a relationship whereby one object has ownership of another class. In the case of Aggregation, the child can exist independently from the parent. For example, if the parent was a lecture, and the child was a student. Removing the lecture class would mean the student would still exist. However, in the case of Composition, it implies the opposite. The child cannot exist independently without the parent. (VisualParadigm, 2017)

These relationships are important to keep in mind when designing applications. Before any code is written, UML diagrams are drawn up to visualise the structure of the program and decide the fundamental elements such as methods and return types. The following UML diagram represents the early, original program structure I envisioned. However, the design was superseded by a different one when changes were made.

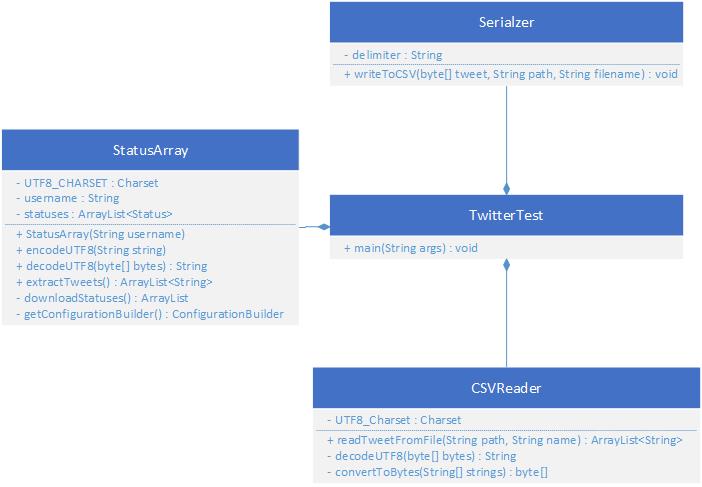


Figure 10 - Original Twitter Java Program UML

ONCE TWITTER PROGRAM COMPLETE, PUT UML FOR IT HERE

### [4.1.3] Single Responsibility Principle (SRP)

As discussed in the [Literature Review](#_[13.2]_Literature_Review), The Single Responsibility Principle is one of the five fundamental S.O.L.I.D Principles. To quote such research;

“The Single Responsibility Principle says that a class should have one and only one reason to change, meaning that a class should only have one job.”

I therefore ensured that each Class in the program had only one role. As I regularly revised the codes structure, I re-arranged particular methods and split up classes where possible in order to adhered to the principle.

### [4.1.4] Formatting & Sanitisation of Tweets

What is a tweet? By definition, it is the chirp of a small young bird. However, in the context of the social media platform ‘Twitter’, it refers to a post on the microblogging service. Tweets aren’t very long, limited to a maximum of 280 characters. They are short and sweet like the chirp of a bird. (Nations, 2017)

On the surface, tweets seem relatively simple. However, they aren’t simply just text, they can contain images, videos, geo-location, mentions, re-tweets and polls. Regarding the tweets use-case in this project, the ‘types’ of tweets aren’t concerning, as the text will be extracted from the tweet anyway. The problem lies in the myriad of possible characters that could appear in the tweet itself. Below is an example of a JSON object retrieved from a Twitter users’ timeline. It contains information such as the date posted, the unique id number of the user, and of course, the text element of the tweet itself. (Nations, 2017)



Figure 11 - Twitter Tweet JSON Object

The ‘text’ property of the object is the element which will be parsed for Robocode to autonomously control its robots. As previously mentioned, the text can contain up to a maximum of 280 characters. However, the definition of a ‘character’ is a somewhat ambiguous. For the vast majority of tweets, each character is a single byte. Using anything else beyond basic letters, numbers and punctuation, causes things to get a little more complicated. The example below is taken from the Twitter Developer documentation and explains one exception to this rule;

|  |  |  |
| --- | --- | --- |
| **Word** | **UTF-8 Bytes (Hex)** | **Description** |
| café | 0x63 0x61 0x66 0xC3 0xA9 | Using the “é” character, called the “composed character”. |
| café | 0x63 0x61 0x66 0x65 0xCC 0x81 | Using the combining diacritical, which overlaps the “e” |

Figure 12 - Twitter Hexadecimal Byte Sequence Differentiation

The word “café” actually has two byte sequences that look exactly the same, but use a different number of bytes. To the human eye, both instances of the word “café” have a character length of four. However, they *really* have lengths of 5 and 6 bytes respectively. Thankfully, Twitter doesn’t want to penalise users for the fact they have chosen the UTF-8 Charset and so regardless of which representation is used, it will always be counted as 4 characters. (Twitter, 2017)

Considering the vast number of supported language, and by virtue, the number of possible characters available, it was decided that only Twitter accounts using the English language will be used. Although this reduces the possible number of characters and combinations, it vastly reduces the complexity of the programming and error handling involved in the project, making it manageable for a single developer and their time constraints.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Twitter Constraints** | |  | **Developer Constraints** | |
| Tweet Length | 280 Characters |  | Language | English Only |
| Character Set | UTF-8 Charset |  | Whitespaces? | Remove All |

I also needed to decide on which attributes I wished to take from the Twitter status objects. MongoDB automatically creates a unique **\_id** field. I then decided to store the Twitter Tweet ID, the post date, the text itself, the username and the language. The stored document object would look something like this;



Figure 13 - Initial Design of Tweet Object Document

### [4.1.5] Java Documentation (Javadoc)

Javadoc is an extensible documentation generation system which reads specially formatted comments in Java source code and generates compiled documentation. It is typically used to produce API documentation in the form of HTML web pages. (StackOverflow, 2018) IDEs, such as JetBrains IntelliJ that is being used for this project, can generate contextual API descriptions for user-defined methods. This means that when invoking a method, for example, inputting Ctrl + Q in IntelliJ will display a window explaining the method, what it does, and what arguments (with data types) that it accepts. These features can increase programming productivity and efficiency if utilised properly.

Javadoc is similar to multiple line comments, except it is denoted by the double asterisk after the initial forward slash. Most IDE’s automatically open a Javadoc comment with the parameter names when the user types /\*\* and returns. The example below shows the constructor for the TweetHandler Class, with Javadoc explaining what it does.



Figure 14 - Javadoc Example Method

When instantiating a TweetHandler Object from the above Class, hitting Ctrl + Q in IntelliJ then displays the Javadoc in a pop-up window for the user.

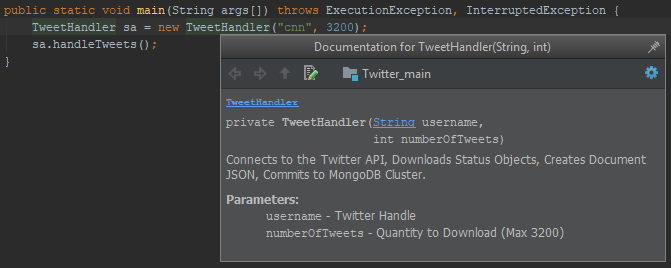


Figure 15 - IntelliJ Contextual API Documentation

## [4.2] Robocode Programming Game

### [4.2.1] Parsing Tweets from the TweetReader

|  |  |  |
| --- | --- | --- |
| **Robocode Constraints** | | |
| Development Language | Java or .NET |
| Fire Power | 0.1(Min) to 3.0 (Max) |
| Starting Position | Has to be within battlefield dimensions |
| Starting Health | 100 (Default) |
| Energy | Returns 3 x Fire Power to firing Robot |
| Energy Return | 3 x Fire Power (After hitting another Robot) |
| Bullet Damage | (Fire Power x 4) + (max(0, Fire Power - 1) x 2) |
| Gun Heat | 1 + (Power / 5) after firing a bullet |
| Bullet Velocity | 20 – (3 x Fire Power) |
| Robot Velocity | 8 Pixels per Turn |
| Robot Acceleration | 1 Pixel per Turn |
| Robot Deceleration | 2 Pixels per Turn |
| Robot Base Rotation | Max (10 – 0.75 x abs(velocity)) Degrees per Turn |
| Gun Rotation | Max 20 Degrees per Turn |
| Radar Rotation | Max 45 Degrees per Turn |
| Collision with Robot | 0.6 Damage |
| Collision with Wall | AdvancedRobots take abs(velocity) x 0.5 – 1 damage |

|  |  |
| --- | --- |
| **Developer Constraints** | |
| Battlefield Size | Max 1500x1500px |
| No. of Rounds | Max 10 |
| No. of Robots | 2 (1v1) |
| Gun Cooling Rate |  |
| Inactivity Time |  |
| Sentry Border Size |  |

### [4.2.2] Battle Results Data Structure

In order to store the results of a Robocode battle in the MongoDB Cluster, a data structure or ‘model’ is required. Each battle can contain *n* robots, and each robot has a number of attributes relating to the results that can be accessed upon completing a battle.

These attributes are;

1. **Team Leader Name:** packageName.RobotName (e.g. sample.Corners)
2. **Bullet Damage:** Robots score 1 point for each point of damage done to enemies
3. **Bullet Damage Bonus:** When a robot kills an enemy, it scores an additional 20% of all the damage it did to that enemy.
4. **Ram Damage:** Robots score 2 points for each point of damage they cause by ramming enemies.
5. **Ram Damage Bonus:** When a robot kills an enemy by ramming, it scores an additional 30% of all the damage it did to that enemy.
6. **Survival Score:** Each robot that's still alive scores 50 points every time another robot dies.
7. **Last Survivor Bonus:** The last robot alive scores 10 additional points for each robot that died before it.
8. **Score:** This is everything else added up, and determines each robot's rank in this battle.
9. **Rank:** A position based upon the score value.
10. **Firsts:** Number of rounds the robot finished in first place at the end of a round.
11. **Seconds:** Number of rounds the robot finished in second place at the end of a round.
12. **Thirds:** Number of rounds the robot finished in third place at the end of a round.

The data structure would contain *n* number of objects in an array, where each object has 12 key value pairs. Each key value pair is such that the key is the attribute name, and the value is the respective value. Each battle will output a single document with a single attribute called “results” whose key is the aforementioned array of objects.

The figure to the left shows an example of this data structure. The battle contained only two robots, “sample.Corners” and “sample.Crazy”. The array therefore contains two objects, each with the 12 attributes.

Figure 16 - Robocode Results Data Structure Example

## [4.3] Data Visualisation & NodeJS

### [4.3.1] MVC Folder Structure

The ExpressJS Framework utilises MVC (Model, View, Controller). MVC splits the program into 3 main, inter-connecting parts that makes managing and developing the project easier. The file structure for this NodeJS project is as follows;

* /bin
* /db (MODEL)
* /node\_modules
* /public (CONTROLLER)
  + js (JavaScript functionality here)
  + css
* /routes
* /views (VIEW)
* app.js

# [5] Program Development

This section documents the development stages throughout the project. Any Object-Orientated programming is written in Java, and the code snippets are taken from the program written in IntelliJ IDEA.

## [5.1] Twitter

### [5.1.1] Twitter Authentication & API Connection

The first task was to connect to the Twitter API so that the program has access to its methods. As previously mentioned in the background research, the Twitter4J Library was selected for this purpose. I downloaded the .JAR files from their site and installed on the classpath by adding them as a new module to the project in IntelliJ.

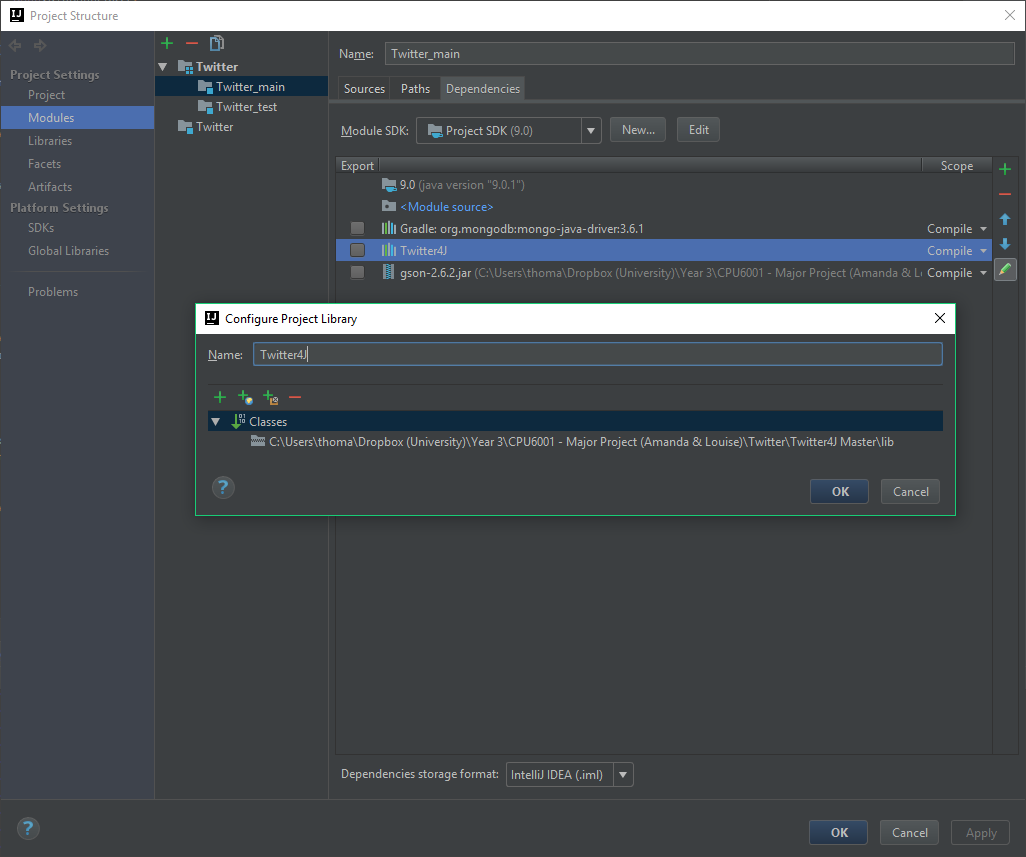


Figure 17 - Adding Twitter4J .JAR to IntelliJ Project

I then included the import statements at the top of my Class and used the wildcard asterisk character to include all available classes in the .JARs. This meant I could now invoke all necessary methods from the Twitter4J Library in my Class.



Figure 18 - Java External Library Import Statements

Twitter utilises the Open Authorisation (OAuth) 2.0 Framework, an open protocol used to protect data on web applications and server-side API’s (OAuth, 2017). Developers are provided with an access token and a secret token that can be used to make API requests to Twitter via an ‘Application’ (Twitter, 2017). The screenshot below shows both the Access Token and Access Token Secret keys. This also allows for the OAuth authentication to function correctly as previously mentioned in the background research.

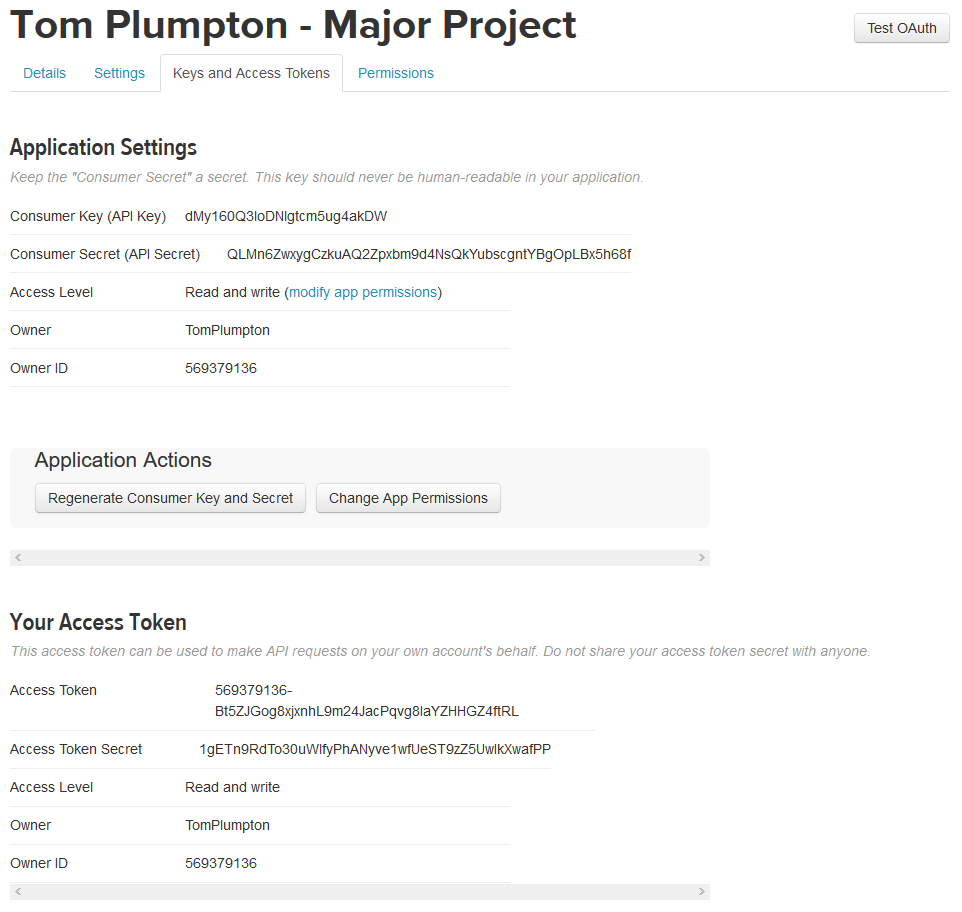


Figure 19 - Twitter Developer Application Access Tokens

Now that I had my access token, the documentation states that I had to create a ConfigurationBuilder. I wrote a method that instantiates a new ConfigurationBuilder Object, sets all the relevant keys, and then returns it. This method will be invoked upon in other methods further on, facilitating the TwitterFactory.



Figure 20 - Twitter ConfigurationBuilder Method

### [5.1.2] Mongo Atlas Cluster Connection & Inserting Documents

During the early stages of project development, I decided that it would be better to simply store the tweets in the Mongo Database Cluster, rather than serialising them into .CSV files on local storage. This would mean the data is available online anywhere, making it easier to develop when using multiple different computers. Furthermore, given the significant quantity of stored data, file sizes would get rather large and start to bloat the local disk.

I decided to install ‘Gradle’ as a module in IntelliJ for the Twitter project. Grade is an open source build automation system that uses domain-specific language (DSL) instead of XML to declare project configuration. (Wikipedia, 2017) IntelliJ has Gradle integration out-of-the-box, meaning that I simply had to navigate to **New**>**Module**>**Gradle** and configure the settings as necessary. I added the latest version of MongoDB Driver to the dependencies object in **build.gradle** which gave my program access to the MongoDB Driver API. The library could have been installed on the classpath manually, but installing Gradle made it, and future libraries, easier to install and manage.

Firstly, I registered an account with MongoDB Atlas, a free cloud database service for MongoDB, and created a new cluster called ‘major-project’. All the back-end configuration is handled automatically on the server-side and is ready to use in a few minutes. I then installed Mongo Compass, a free program by MongoDB that allows you to connect to a cluster and view the database document in a GUI.

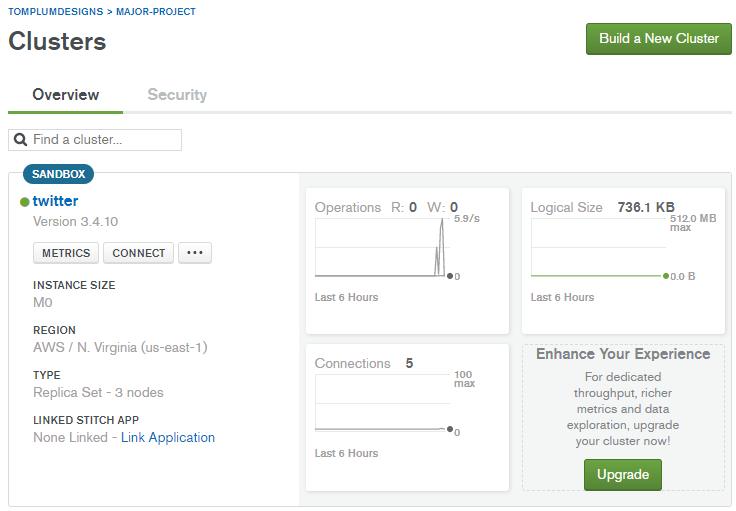


Figure 21 - MongoDB Atlas Cluster

Now that the database was setup, I wrote a Java Class that connected to Twitter, downloaded Tweets, converted them to JSON Objects, then committed them to the MongoDB. Considering these operations are all time-consuming, non-blocking asynchronous code is required to allow time for each operation to complete and callback. I made use of the ‘CompleteableFuture’ Class to achieve this. The following code snippet shows the method that I wrote to handle this functionality. The method invokes four asynchronous operations. Download the Twitter Status Objects, which contain all the information. Extract the relevant information that I wish to store and create an Object of type Document. Insert them into the Mongo database. Then finally, print out some statistics and confirmation, the close the connection to the cluster.



Figure 22 - Asynchronous CompletableFuture Method

The **handleTweets()** method uses the ‘conn’ variable which is declared at the top of the Class like so;



Figure 23 - Private Visibility MongoConnection Member

This is using composition to instantiate an Object of type ‘MongoConnection’ which is derived from the Class I wrote to connect to the database. This code snippet below shows the MongoCollection Class. It declares a String that holds the URI of the Mongo Atlas Cluster. It also declares a new MongoClient Object, available from the API that was configured in Gradle, and passes it a new MongoClientURI Object with the String. I the declared the relevant methods for retrieving the collection from the database, inserting documents and disconnecting when finished.



Figure 24 - User Defined MongoConnection Class

Once everything was cleaned up, I wanted to insert the maximum number of Tweets from an account in one API call, which was 3200. I pass the TweetHandler Object Constructor the name “CNN” and ran the program. It successfully stored all 3200. For good measure, I tried it with another account “BBC” and successfully had a collection of 6400 documents. The screenshot below shows the GUI view in MongoCompass. However, I immediately noticed that the formatting was off in some of the text attributes, and emojis were also being parsed. This was not something I anticipated during early research and planning phases and would now need to be dealt with. I aimed to write a method that sanitises the text strings before they’re appended to the Documents.

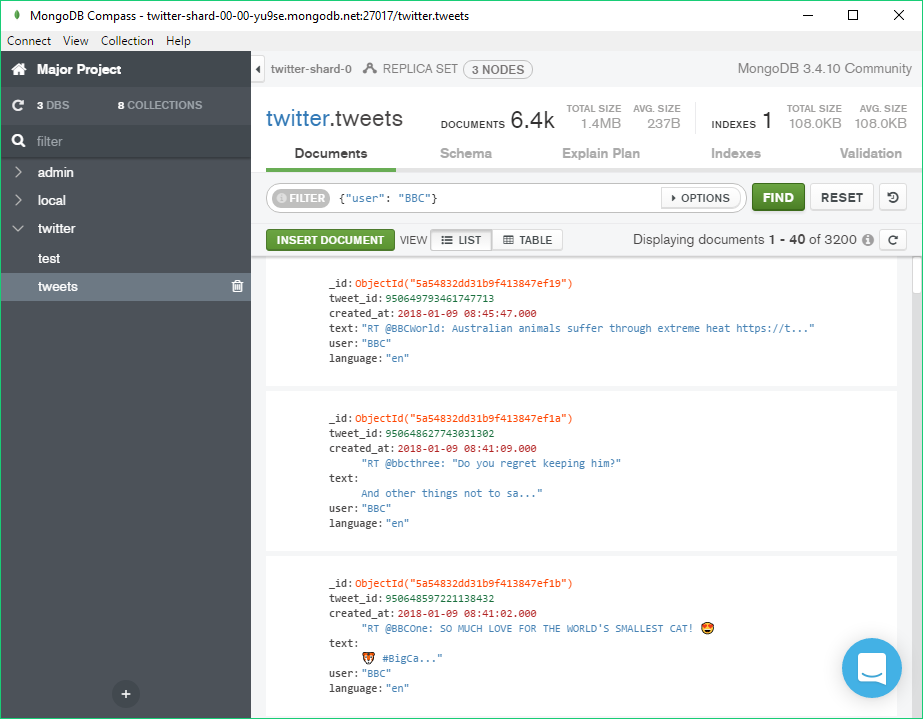


Figure 25 - MongoDB Compass Collection with 6400 Tweets Inserted

### [5.1.3] Tweet String Sanitisation

As previously mentioned in the last sub-section of this chapter, the Tweet strings needed to be sanitised to remove or re-format it in such a way that it could be properly parsed by Robocode in the future. This includes bytes such as white-spaces, line-breaks, emojis and other symbols. A new TweetSanitiser Class was created with methods that re-format Tweet strings in some way. For example, remove all white-spaces, emojis and line-breaks.

The code snippet below shows three methods that achieve them respectively. **removeWhiteSpaces()** and **removeLineBreaks()** use regex (Regular Expressions) to identify what to replace.



Figure 26 - TweetSanitiser Class Snippet

### [5.1.4] Tweet String Parsing

For the first iteration of development, I decided to keep the complexities to a minimum by keeping the logic of the algorithms simple. I used to the TweetSanitiser class to strip the Tweet string of all emojis, symbols and numbers so that only alpha-numeric characters remained. I then wrote a method called **getValue()** which accepts a letter and finds the letter in a array containing the alphabet. It then returns the respective index value from 0-25.



Figure 27 - TweetParser Class

## [5.2] Robocode

After finishing the Twitter related stage of development and testing each of the methods and functionality that I had written, I started on the second stage. This involved integrating the Robocode application within my IntelliJ project, and creating a new class that extends the Robocode AdvancedRobot Interface and interacts with the Twitter related classes from the previous section.

### [5.2.1] Integrating Robocode with IntelliJ

As previously mentioned in the [Background Research](#_[2]_Background_Research) chapter, I copied the entire Robocode master repository into my IntelliJ Project to overcome a problem with output paths. This wasn’t technically necessary to run Robocode from the project, it would just mean that each computer the project was used on had Robocode installed in the same local directory. However, by keeping it local to the project source itself, it could be run anywhere during development.

To get Robocode running, I created a new ‘Application’ configuration in IntelliJ. The configurations menu can be accessed from the upper pane in the IntelliJ window.

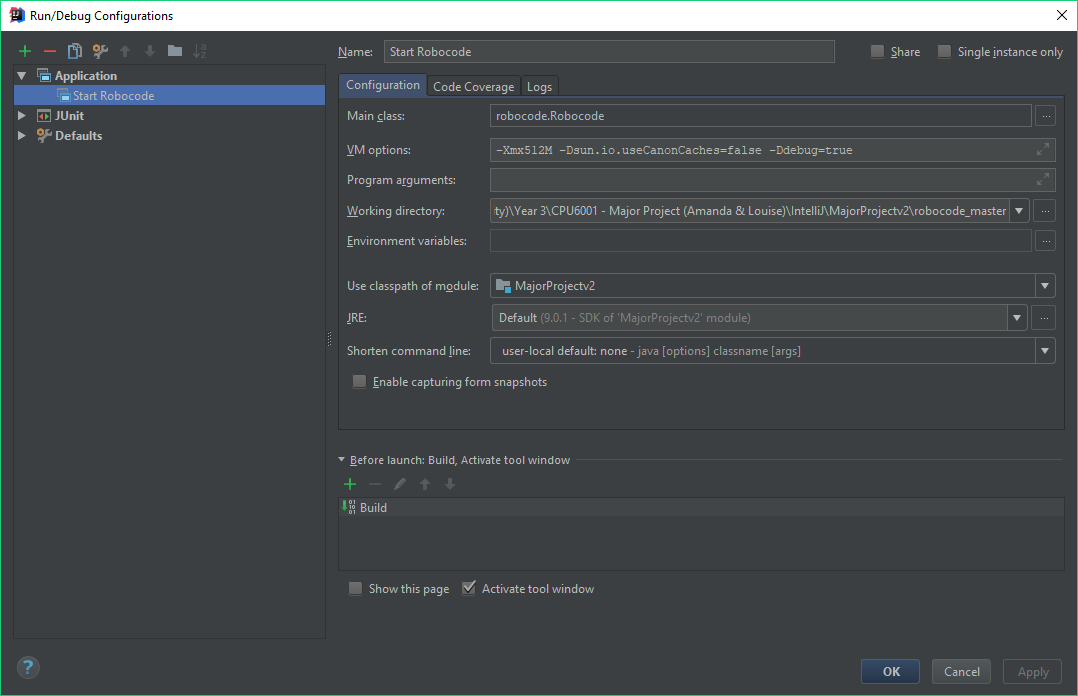


Figure 28 - Start Robocode Application Run Configuration

The screenshot above shows the configuration of the Application. The three configuration options I used are the ‘Main class’, the ‘VM Options’ and the ‘Working directory’.

**Main Class:** robocode.Robocode

This line is pointing to the ‘robocode’ package that I have on the project classpath. This is the main robocode.jar file that contains all of the classes and interfaces that make up the game itself, allowing my program to build and compile. It then drills down further into the root of the package and points to the ‘Robocode’ class. This is the main class that contains the functionality for starting the game.

**VM Options:** -Xmx512M -Dsun.io.useCanonCaches=false -Ddebug=true

The VM (Virtual Machine) options specifies what the JVM (Java Virtual Machine) should do differently than its default configuration. The ‘-Xmx512M’ part tells the JVM that it can use up to a maximum of 512MB of RAM off the heap. ‘-Dsun.io.useCanonCaches=false’ is a workaround to prevent SecurityExceptions from occurring when a robot tries to access a file. Finally, ‘-Ddebug=true’ tells Robocode that is should not give skipped turns just because the developer paused the robot in debugger mode. (Robowiki, 2017)

**Working Directory**

As the name suggests, this is the folder directory in which the application should look for the files that it needs. Since I added the Robocode source files to the project path, I simply pointed the working directory to the ‘robocode\_master’ folder in IntelliJ.

### [5.2.2] Creating an Advanced Robot Class

Initially, I created a new class called ‘TwitterRobot’ and created the common layout used by many Robocode robots.



Figure 29 - TwitterRobot Class with Default Template

As previously mentioned in the background research, I had changed the output path in IntelliJ to the ‘robots’ folder in the root of the Robocode master repository. The screenshot bellows shows the project file structure. You can see that IntelliJ has created directories for the respective packages in my source root.

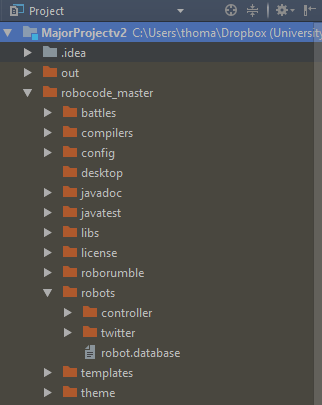


Figure 30 - IntelliJ Project Structure Pane

I started up Robocode from IntelliJ, created a new battle and could see that it had read the controller folder and found the TwitterRobot class in the correct directory. I then had the ability to add new TwitterRobots to the battle.

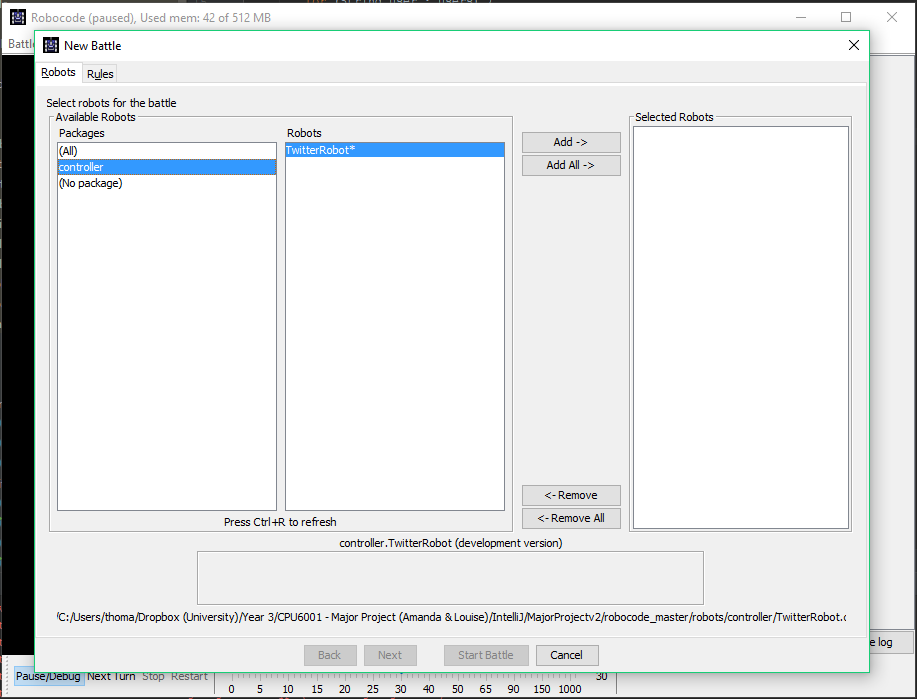


Figure 31 - Adding TwitterRobot's to Battle

### [5.2.3] Overcoming Security Exceptions

Robocode has built in security that prevent Robots from reading and writing files that are not core to the game. This is to stop Robots from cheating in RoboRumble battles, and to ensure that community downloaded robots cannot cause a threat to your system by writing malicious code onto the local computer.

Because my TwitterRobot class was instantiating objects of classes that are not part of core Robocode, the security manager was banning my robots from the game and destroying them upon starting a new battle.

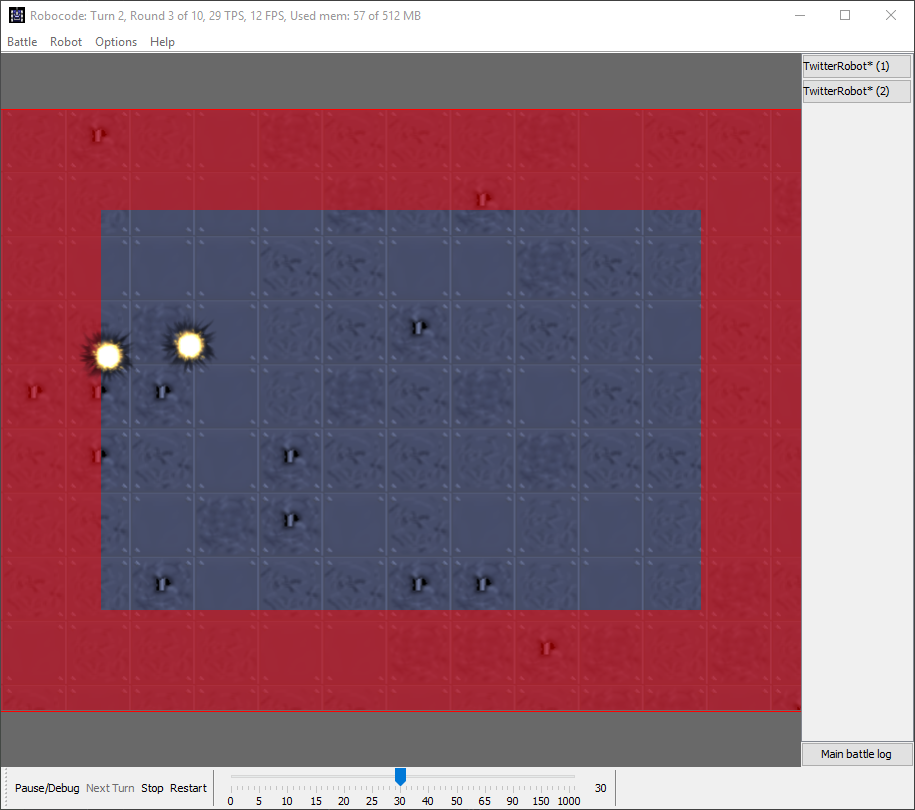


Figure 32 - Robocode Security Manager Destroying Robots

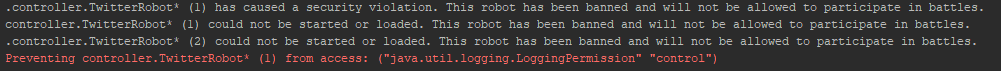


Figure 33 - Robocode Security Manager Logging Violation

To overcome this issue, I simply had to add an argument to the JVM Options in the application run configuration for Robocode in IntelliJ. This argument is **-DNOSECURITY=true**.

### [5.2.4] Controlling the Robots

## [5.3] Data Visualisation & NodeJS

### [5.3.1] Configuring NodeJS & ExpressJS Framework

The first step was to configure the HTTP Server for NodeJS and point it to the main app.js file which will initialise all the middleware and page routes for the application. This file is called ‘www’ and is stored in the bin folder and is what will be run by NodeJS.



Figure 34 - Configuring HTTP Server in NodeJS

I installed the relevant packages for the project. Including the ExpressJS Framework, MongoDB, Connect, BodyParser etc. This was done from the command line terminal using the command **npm install *package-name* –save**. The save flag ensured that the dependencies are updated in the package.json file.

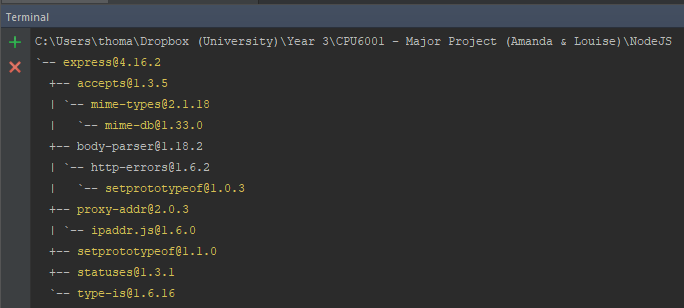


Figure 35 - Installing ExpressJS Framework via NPM Command Line

Then I created the app.js file in the root directory so that the aforementioned www file can see it. I imported the required npm packages that were installed earlier. I setup the pug view engine so that .pug files are used instead of .html. Then, I configure the ‘public’ folder as a static directory for all assets and controller code. Next, I add the required middleware to the app variable. Finally, I configure the first page routing file.



Figure 36 - Configuring Middleware Stack & Page Routing in NodeJS

The following code snippet shows the package.json file for the project. The object defines some attributes for the project, including name, description, version etc. Then it has a dependencies object which is update each time a new npm package is installed with the **–save** flag.



Figure 37 - NodeJS package.json file contents

The property called “scripts” is an object contains key:value pairs. The key is an npm command, and the value is what to do with it. I declare a new pair with the key “start” and the value “node ./bin/www”. This tells npm to run the www file with the server configuration upon the command **npm start bin/www** from the project root directory.

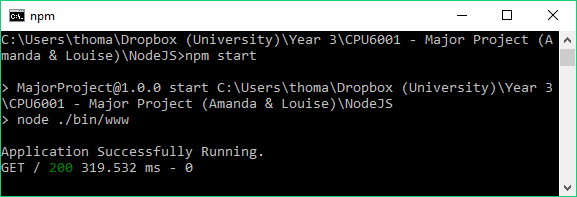


Figure 38 - Running NodeJS Application from NPM Command Line

### [5.3.2] Automating NodeJS Runtime in JetBrains WebStorm

In the previous sub-section, it was noted that to run the NodeJS application, the developer had to open a NodeJS Command Prompt, navigate to the project root directory, and execute the command **npm start bin/www**. WebStorm provides an easier method of doing this, while also provides further benefits.

I created a new Node.js Run Configuration in WebStorm. I set the working directory to the project root directory, and the JavaScript file to bin/www. This meant I could now run the application from within WebStorm, as it has access to the NodeJS interpreter application and knows where the www file is.

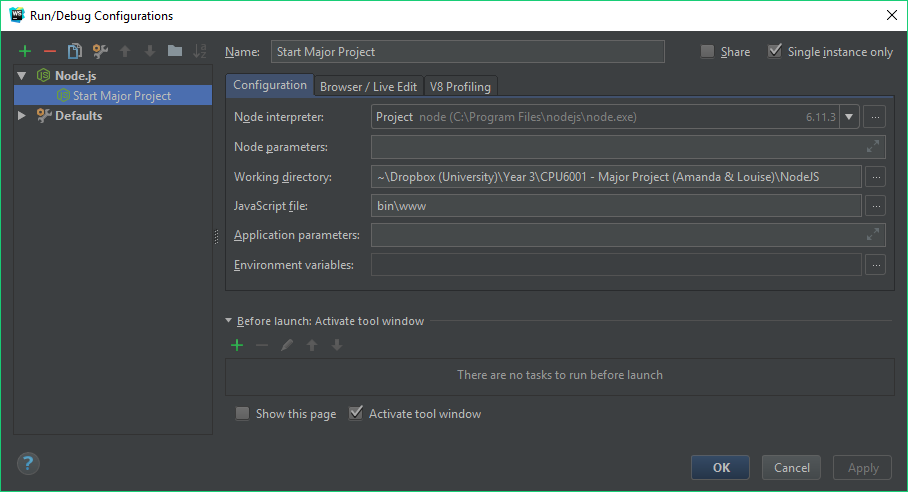


Figure 39 - JetBrains WebStorm NodeJS Application Run Configuration

Simply clicking the green play button in WebStorm causes the application to run successfully. Any command line output is printed to the run console within the IDE, now eliminating the need to use a separate NodeJS Command Line window.

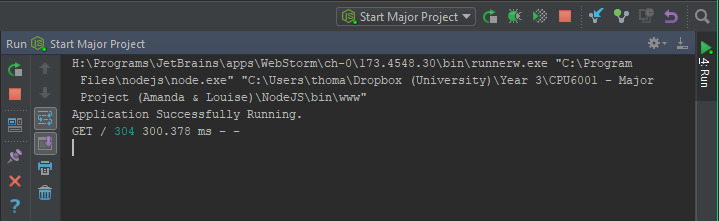


Figure 40 - Running NodeJS Application from WebStorm

### [5.3.3] Creating the Page & Integrating D3.JS

## [5.4] Mathematical Analysis

# [6] Testing Phase

This section documents the various testing phases that were undergone at the end of each stage of development. Furthermore, any issues that were discovered as a result, are documented along with any changes that were made in correction.

## [6.1] Twitter

### [6.1.1] Accessing the MongoDB Atlas Cluster

After writing the first draft of code for storing the Tweets in the database, I tested it by inserting a single document with a single attribute into a test collection. I simply added a main method to the MongoConnection Class, so it was runnable.



Figure 41 - Testing with MongoConnection Main Method

After the first stage of development was complete and functional, the program was tested with some randomly selected accounts to see if the JSON objects would successfully commit to the Mongo Cluster where they could be viewed in Mongo Compass. The program ran successfully, and the test document appeared in Compass.

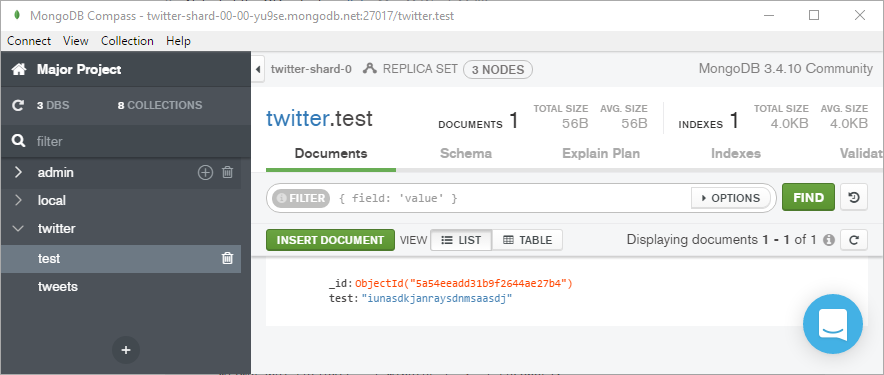


Figure 42 - MongoDB Compass Test Document

### [6.1.2] Committing Twitter Status Objects to the MongoDB Atlas Cluster

To ensure that the program worked correctly, I tried inserting 3200 Tweets for several different Twitter users. The first one I tried as I wrote the program was “cnn”. The Tweets were committed successfully but I noticed the user field was “CNN” (capital casing), just how the handle was displayed (@CNN) on the profile. I questioned the **status.getUser().getName()** method invocation I was using to store the username, so I tried a few other users.

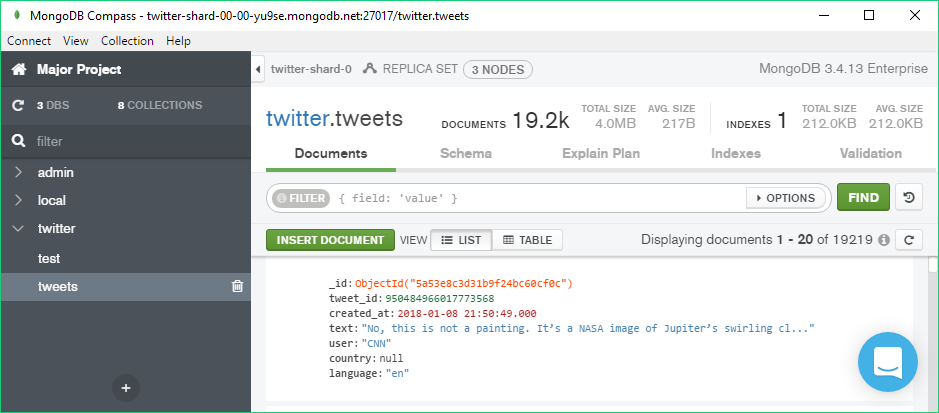


Figure 43 – MongoDB Compass Displaying 19200 Documents

### [6.1.3] Sanitising Tweets

I wrote a TweetSanitiserTest class with a series of Junit 5 tests. I declared a **downloadTweets()** method that is to be run before all tests. It downloads a list of usernames from the MongoDB Atlas Cluster, picks a pseudo-random one, and downloads all the Tweets from that user.



Figure 44 - TweetSanitiser JUnit Test Method - downloadTweets()

I then wrote several tests that verify the working functionality of each method in the TweetSanitiser Class. Below are examples of a test passing and failing.

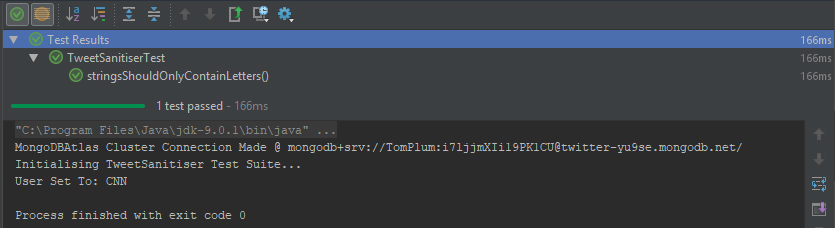


Figure 45 - TweetSanitiser JUnit Test Pass

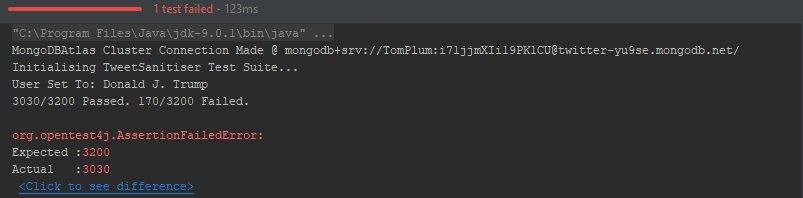


Figure 46 - TweetSanitiser JUnit Test Fail

## [6.2] Robocode

### [6.2.1] Adding Custom Robots to Robocode

### [6.2.2] Robot Controller Random Values

# [7] Results

# [8] Evaluation

# [9] Conclusions

# [10] Glossary of Terms

|  |  |
| --- | --- |
| **Term** | **Definition** |
| Agile | Relating to or denoting a method of project management, used especially for software development, that is characterized by the division of tasks into short phases of work and frequent reassessment and adaptation of plans. |
| API | A set of functions and procedures that allow the creation of applications which access the features or data of an operating system, application, or other service. |
| Cache | An auxiliary memory from which high-speed retrieval is possible. |
| Classpath | Classpath is a parameter in the Java Virtual Machine or the Java compiler that specifies the location of user-defined classes and packages. The parameter may be set either on the command-line, or through an environment variable. |
| CSV | A CSV is a comma separated values file which allows data to be saved in a table structured format. CSVs look like a garden-variety spreadsheet but with a .csv extension. Traditionally they take the form of a text file containing information separated by commas, hence the name. |
| D3.JS | A JavaScript library for producing dynamic, interactive data visualizations in web browsers. |
| DSL | Domain-Specific Language. A computer language specialized to a particular application domain. This is in contrast to a general-purpose language (GPL), which is broadly applicable across domains. ... Simpler DSLs, particularly ones used by a single application, are sometimes informally called mini-languages. |
| GET | GET is a HTTP request usually intended to retrieve some data, and is expected to be idempotent (repeating the query does not have any side-effects) and can only send limited amounts of parameter data to the server. |
| Gradle | An open-source build automation system that builds upon the concepts of Apache Ant and Apache Maven and introduces a Groovy-based domain-specific language (DSL) instead of the XML form used by Apache Maven for declaring the project configuration. |
| GUI | Graphical User Interface. A type of user interface that allows users to interact with electronic devices through graphical icons and visual indicators such as secondary notation, instead of text-based user interfaces, typed command labels or text navigation. |
| HTTP | Hyper Text Transfer Protocol. HTTP is the underlying protocol used by the World Wide Web and this protocol defines how messages are formatted and transmitted, and what actions Web servers and browsers should take in response to various commands. |
| Java | A general-purpose computer programming language designed to produce programs that will run on any computer system. |
| JVM | Java Virtual Machine. |
| JavaScript | An object-oriented computer programming language commonly used to create interactive effects within web browsers. |
| OAuth | OAuth (Open Authorization) is an open standard for token-based authentication and authorization on the Internet. |
| PRNG | Pseudo-Random Number Generator. A program written for, and used in, probability and statistics applications when large quantities of random digits are needed. |
| Robocode | Robocode is a programming orientated game, where the goal is to develop a robot battle tank to battle other tanks in Java or Microsoft .NET. |
| Server | A computer or computer program which manages access to a centralized resource or service in a network. |
| TRNG | True-Random Number Generator. A mathematical construct, either computational or as a hardware device, that is designed to generate a random set of numbers that should not display any distinguishable patterns in their appearance or generation. |
| Twitter | An online news and social networking service where users post and interact with messages, "tweets", restricted to 140 characters. Registered users can post tweets, but those who are unregistered can only read them. |
| VM | Virtual Machine. |
| XML | Extensible Markup Language. A metalanguage which allows users to define their own customized markup languages, especially in order to display documents on the Internet. |

# [11] Bibliography

360logica, n.d. *Agile Development – Advantages, Disadvantages and when to use it?.* [Online]   
Available at: http://www.360logica.com/blog/agile-development-advantages-disadvantages-and-when-to-use-it/  
[Accessed 30 11 2017].

Anon., 2017. [Online]   
Available at: https://twitter.com/

Anon., 2017. [Online]   
Available at: http://robocode.sourceforge.net/

Anon., 2017. *Introduction to Randomness and Random Numbers.* [Online]   
Available at: https://www.random.org/randomness/

Anon., 2017. *The Importance and Effect of Sample Size.* [Online]   
Available at: https://select-statistics.co.uk/blog/importance-effect-sample-size/

Anon., n.d. *The frequency of the letters of the alphabet in English.* [Online]   
Available at: https://www3.nd.edu/~busiforc/handouts/cryptography/letterfrequencies.html  
[Accessed 19 01 2018].

Anon., n.d. *Linear Regression.* [Online]   
Available at: http://www.stat.yale.edu/Courses/1997-98/101/linreg.htm  
[Accessed 19 12 2017].

Bell, L., 2017. *What is caching and how does it work?.* [Online]   
Available at: http://www.wired.co.uk/article/caching-cached-data-explained-delete

Bostock, M., 2017. [Online]   
Available at: https://d3js.org/

Bostock, M., 2017. *Data-Driven Documents.* [Online]   
Available at: https://d3js.org/  
[Accessed 06 12 2017].

Cunningham, A., 2014. *The bizarre, mind-numbing, mesmerizing beauty of “Twitch Plays Pokémon”.* [Online]   
Available at: https://arstechnica.com/gaming/2014/02/the-bizarre-mind-numbing-mesmerizing-beauty-of-twitch-plays-pokemon/  
[Accessed 12 03 2017].

Dua, T., 2016. *The year in emojis, in 5 charts.* [Online]   
Available at: https://digiday.com/careers/year-emojis-5-charts/  
[Accessed 09 01 2018].

Emogi, 2016. *2016 Emoji Report.* [Online]   
Available at: http://cdn.emogi.com/docs/reports/2016\_emoji\_report.pdf  
[Accessed 18 01 2018].

Emojipedia, 2018. *FAQ.* [Online]   
Available at: https://emojipedia.org/faq/  
[Accessed 18 01 2018].

Foote, K., 2016. *A Review of Different Database Types: Relational versus Non-Relational.* [Online]   
Available at: http://www.dataversity.net/review-pros-cons-different-databases-relational-versus-non-relational/  
[Accessed 12 12 2017].

Gee, T., 2014. *Getting Started with MongoDB and Java: Part I.* [Online]   
Available at: https://www.mongodb.com/blog/post/getting-started-with-mongodb-and-java-part-i  
[Accessed 08 01 2018].

Haahr, D. M., 2017. *Pseudo-Random Number Generators (PRNGs).* [Online]   
Available at: https://www.random.org/randomness/  
[Accessed 06 12 2017].

Heller, B., 2016. *Analytics 101: Choosing the right database.* [Online]   
Available at: https://reflect.io/blog/analytics-101-choosing-the-right-database/  
[Accessed 25 12 2017].

Hern, A., 2015. *Don't know the difference between emoji and emoticons? Let me explain.* [Online]   
Available at: https://www.theguardian.com/technology/2015/feb/06/difference-between-emoji-and-emoticons-explained  
[Accessed 18 01 2018].

itinfo, 2017. *Software Development Methodologies.* [Online]   
Available at: http://www.itinfo.am/eng/software-development-methodologies/  
[Accessed 26 11 2017].

Jenkov, J., 2015. *Java Lambda Expressions.* [Online]   
Available at: http://tutorials.jenkov.com/java/lambda-expressions.html  
[Accessed 11 01 2018].

JetBrains, 2017. *DataGrip.* [Online]   
Available at: https://www.jetbrains.com/datagrip/  
[Accessed 18 12 2017].

JetBrains, 2017. *Gradle.* [Online]   
Available at: https://www.jetbrains.com/help/idea/gradle.html  
[Accessed 08 01 2018].

JetBrains, 2017. *IntelliJ IDEA.* [Online]   
Available at: https://www.jetbrains.com/idea/  
[Accessed 13 12 2017].

JetBrains, 2017. *jetbrains.com.* [Online]   
Available at: https://www.jetbrains.com/  
[Accessed 22 12 2017].

JetBrains, 2017. *WebStorm.* [Online]   
Available at: https://www.jetbrains.com/webstorm/  
[Accessed 13 12 2017].

Leonhard, M., 2010. *StackOverflow.* [Online]   
Available at: https://stackoverflow.com/questions/88838/how-to-convert-strings-to-and-from-utf8-byte-arrays-in-java  
[Accessed 08 11 2017].

Mathew Nelson, F. L. R. M. P. S., 2017. *Class AdvancedRobot.* [Online]   
Available at: http://robocode.sourceforge.net/docs/robocode/robocode/AdvancedRobot.html  
[Accessed 22 02 2018].

MathsIsFun, 2017. *Standard Deviation and Variance.* [Online]   
Available at: http://www.mathsisfun.com/data/standard-deviation.html  
[Accessed 19 12 2017].

McLaughlin, M., n.d. *What Is Agile Methodology?.* [Online]   
Available at: https://www.versionone.com/agile-101/agile-methodologies/  
[Accessed 30 11 2017].

Microsoft, 2017. *.Net Framework system requiremets.* [Online]   
Available at: https://docs.microsoft.com/en-us/dotnet/framework/get-started/system-requirements  
[Accessed 16 12 2017].

MongoDB, 2018. *Java Driver Example.* [Online]   
Available at: https://docs.atlas.mongodb.com/driver-connection/#java-driver-example  
[Accessed 08 01 2018].

MongoDB, 2018. *MongoDB Java Driver.* [Online]   
Available at: https://mongodb.github.io/mongo-java-driver/  
[Accessed 08 01 2018].

mykong, 2013. *Mykong.com.* [Online]   
Available at: https://www.mkyong.com/java/how-to-read-and-parse-csv-file-in-java/  
[Accessed 08 11 2017].

Nations, D., 2017. *What Is a Tweet on Twitter?.* [Online]   
Available at: https://www.lifewire.com/what-is-a-tweet-3486211  
[Accessed 03 12 2017].

Nelson, F., 2013. *ReadMe for Robocode.* [Online]   
Available at: http://robocode.sourceforge.net/docs/ReadMe.html  
[Accessed 16 12 2017].

Nivas, 2015. *StackOverflow.* [Online]   
Available at: https://stackoverflow.com/questions/3666007/how-to-serialize-object-to-csv-file  
[Accessed 08 11 2017].

OAuth, 2017. [Online]   
Available at: https://oauth.net/

Oloruntoba, S., 2015. *S.O.L.I.D: The First 5 Principles of Object Oriented Design.* [Online]   
Available at: https://scotch.io/bar-talk/s-o-l-i-d-the-first-five-principles-of-object-oriented-design  
[Accessed 18 12 2017].

Oracle, 2017. *Oracle Fact Sheet.* [Online]   
[Accessed 11 01 2018].

Powell-Morse, A., 2016. *Rapid Application Development (RAD).* [Online]   
Available at: https://airbrake.io/blog/sdlc/rapid-application-development  
[Accessed 06 12 2017].

Raj, J., 2014. *An Introduction to the MEAN Stack.* [Online]   
Available at: https://www.sitepoint.com/introduction-mean-stack/  
[Accessed 23 12 2017].

Random.org, 2017. *Analysis.* [Online]   
Available at: https://www.random.org/analysis/  
[Accessed 19 12 2017].

Robocode, 2017. *Robocode Home.* [Online]   
Available at: http://robocode.sourceforge.net/  
[Accessed 12 12 2017].

Robowiki, 2017. *Frequently Asked Questions.* [Online]   
Available at: http://robowiki.net/wiki/Robocode/FAQ  
[Accessed 22 02 2018].

Robowiki, 2017. *Game Physics.* [Online]   
Available at: http://robowiki.net/wiki/Robocode/Game\_Physics  
[Accessed 19 01 2018].

Robowiki, 2017. *Robocode/Robot Anatomy.* [Online]   
Available at: http://robowiki.net/wiki/Robocode/Robot\_Anatomy  
[Accessed 19 01 2018].

Robowiki, 2017. *Running Robocode From Eclipse.* [Online]   
Available at: http://robowiki.net/wiki/Robocode/Eclipse/Running\_from\_Eclipse  
[Accessed 22 02 2018].

Robowiki, 2018. *AdvancedRobot.* [Online]   
Available at: http://robowiki.net/wiki/AdvancedRobot  
[Accessed 22 02 2018].

Singhal, V., 2015. *20 best JavaScript charting libraries.* [Online]   
Available at: https://thenextweb.com/dd/2015/06/12/20-best-javascript-chart-libraries/  
[Accessed 18 12 2017].

Singh, R., 2017. *Java 8 CompletableFuture Tutorial.* [Online]   
Available at: https://www.callicoder.com/java-8-completablefuture-tutorial/  
[Accessed 08 01 2018].

Skau, D., 2013. *Why D3.js is So Great for Data Visualization.* [Online]   
Available at: https://visual.ly/blog/why-d3-js-is-so-great-for-data-visualization/  
[Accessed 01 01 2018].

StackOverflow, 2018. *Javadoc.* [Online]   
Available at: https://stackoverflow.com/questions/19172015/what-exactly-is-javadoc  
[Accessed 18 01 2018].

TheEclipseFoundation, 2017. *Desktop IDE's.* [Online]   
Available at: https://www.eclipse.org/ide/  
[Accessed 16 12 2017].

tutorialspoint, 2017. *Java - Serialization.* [Online]   
Available at: https://www.tutorialspoint.com/java/java\_serialization.htm  
[Accessed 08 11 2017].

tutorialspoint, 2017. *MongoDB - Advantages.* [Online]   
Available at: https://www.tutorialspoint.com/mongodb/mongodb\_advantages.htm  
[Accessed 13 12 17].

Twitch, 2017. *About.* [Online]   
Available at: https://www.twitch.tv/p/about  
[Accessed 08 03 2017].

Twitter, 2017. [Online]   
Available at: https://apps.twitter.com

Twitter, 2017. [Online]   
Available at: https://developer.twitter.com/en/docs/basics/rate-limiting

Twitter, 2017. *Counting characters.* [Online]   
Available at: https://developer.twitter.com/en/docs/basics/counting-characters  
[Accessed 03 12 2017].

Twitter4J, 2017. *Twitter4J.* [Online]   
Available at: http://twitter4j.org/en/index.html  
[Accessed 16 12 2017].

TwitterDev, 2017. *Tweet data dictionaries.* [Online]   
Available at: https://developer.twitter.com/en/docs/tweets/data-dictionary/overview/tweet-object  
[Accessed 03 12 2017].

TwitterDeveloper, 2017. *Rate Limiting.* [Online]   
Available at: https://developer.twitter.com/en/docs/basics/rate-limiting  
[Accessed 16 12 2017].

TwitterDeveloper, 2017. *Twitter libraries.* [Online]   
Available at: https://developer.twitter.com/en/docs/developer-utilities/twitter-libraries  
[Accessed 16 12 2017].

TwitterInc., 2018. *Streaming API.* [Online]   
Available at: https://developer.twitter.com/en/docs/tweets/filter-realtime/api-reference/post-statuses-filter  
[Accessed 22 01 2018].

Veracode, 2017. *WHAT IS AN INTEGRATED DEVELOPMENT ENVIRONMENT (IDE)?.* [Online]   
Available at: https://www.veracode.com/security/integrated-development-environments  
[Accessed 12 12 2017].

versionone.com, 2017. *What Is Agile Software Development?.* [Online]   
Available at: https://www.versionone.com/agile-101/  
[Accessed 28 11 2017].

VisualParadigm, 2017. *UML Association vs Aggregation vs Composition.* [Online]   
Available at: https://www.visual-paradigm.com/guide/uml-unified-modeling-language/uml-aggregation-vs-composition/  
[Accessed 11 01 2018].

VSauce, 2014. *What Is Random?.* [Online]   
Available at: https://www.youtube.com/watch?v=9rIy0xY99a0  
[Accessed 06 12 2017].

Wikipedia, 2017. *Gradle.* [Online]   
Available at: https://en.wikipedia.org/wiki/Gradle  
[Accessed 08 01 2018].

Wikipedia, 2017. *Java (programming language).* [Online]   
Available at: https://en.wikipedia.org/wiki/Java\_(programming\_language)  
[Accessed 11 01 2018].

Wikipedia, 2017. *Java version history.* [Online]   
Available at: https://en.wikipedia.org/wiki/Java\_version\_history  
[Accessed 11 01 2018].

Wikipedia, 2017. *JetBrains.* [Online]   
Available at: https://en.wikipedia.org/wiki/JetBrains  
[Accessed 22 12 2017].

Wikipedia, 2017. *Twitch Plays Pokémon.* [Online]   
Available at: https://en.wikipedia.org/wiki/Twitch\_Plays\_Pok%C3%A9mon  
[Accessed 05 03 2017].

Wikipedia, n.d. *Twitch Plays Pokémon.* [Online]   
Available at: https://en.wikipedia.org/wiki/Twitch\_Plays\_Pok%C3%A9mon  
[Accessed 05 03 2017].

# [12] Appendices

## [12.1] Terms of Reference

Student Details

**Name:** Thomas Plumpton

**Identification Number:** 1500936

**Email:** tp2amt@bolton.ac.uk

Project Title

Analysing the Random Nature of Social Media

Date of Production

13/10/2017

Background & Overview

Randomness is a mathematical concept that is commonly misunderstood. Computer-generated random numbers are not *truly* random, they are actually *pseudo*-random. This means that although they appear to be random, they are really pre-determined. Although Pseudo-Random Number Generation (PRNG) is efficient and deterministic, these are not desirable attributes in scenarios that require True-Random Number Generation (TRNG). Such scenarios include the generation of data encryption keys and cryptography or simulation of natural phenomena like weather.

This project aims to analyse and determine the random nature of the social media platform ‘Twitter’ by parsing a number of Tweets and using them to control the virtual robots in a game called ‘Robocode’. The data produced will be analysed to determine whether it is pseudo or truly random in an attempt to produce a new method of generating truly random numbers. The inspiration for this project originates from a social experiment called ‘Twitch Play Pokémon’.

Twitch is the world’s leading social video platform for streaming gaming oriented content around the world. It was founded in June 2011 and currently amasses nearly 10 million visitors every day. The site is popular among both veteran and aspiring YouTubers and attracts the attention of big e-sports players and tournaments. (Twitch, 2017)

Twitch Plays Pokémon (TPP) was a social experiment by an anonymous Australian programmer and involves streaming a game of Pokémon on Twitch in the Game Boy emulator ‘VisualBoyAdvance’. The anonymous programmer developed an Internet Relay Chat (IRC) bot, written in Python, which captures messages from the Twitch chat and parses those strings directly into commands for the game. The stream soon became popular and reached approximately 60,000 concurrent playing users. It took nearly 16 continuous days to complete the first game ‘Pokémon Red’. Since then, numerous other Pokémon games from the franchise have been hosted and streamed on Twitch for the users to collaborate on. (Wikipedia, n.d.) (Cunningham, 2014)

Robocode is a lightweight, open-source, programming game developed by Mathew A. Nelson as a personal endeavour in 2000. The goal is to create a virtual robot by writing the code yourself in Java or Microsoft .NET. The game acts as an intermediary between Twitter and the data visualisation by converting the parsed Tweets into instructions to control the robots and run the game.

Mathematics plays an important role in the conclusion of the project. Methods such as statistical regression and standard deviation should be used to determine the random nature of the results output from Robocode and graphed via D3.JS. This will conclude whether Twitter can be used as a TRNG.

Objectives

1. Design & implement a program to download Tweets from Twitter.
2. Investigate how to parse the downloaded Tweets for Robocode.
3. Investigate how to build upon the existing Robocode source code.
4. Create Java code for Robocode and investigate User Experience (UX).
5. Investigate Java connection to databases to store data output.
6. Create web application in NodeJS to host D3.JS Data Visualisation.
7. Connect database to NodeJS and graphically visualise data.
8. Use statistical techniques to determine random nature of the output data.
9. Conclude and evaluate results, consider real world applications.

Constraints

* Robocode requires Java or Microsoft .NET
* Twitter REST API allows only 15 calls every 15 minutes. It also restricts the request to a maximum of 3200 tweets per call.
* Consent to access public profiles

Resources

All software and online resources are open-source and therefore freely available, with the exception of the JetBrains Suite which is free to students.

Software:

* Java IDE (JetBrains IntelliJ IDEA)
* Editor / Web Development Environment (JetBrains WebStorm)
* SQL IDE (JetBrains DataGrip)
* Internet Connection

Technical Skills:

* Java 8
* HTML5 / Pug / Jade (Web Mark-up Languages)
* CSS3 (Cascading Stylesheet)
* JavaScript (Client & Server Side)
* NodeJS (Runtime Environment)
* D3.JS (Data Driven Documents) JavaScript Library

Services:

* Twitter Development API
* Robocode Development API
* Twitter4J Development API

Reporting

1. **Name:** Abdul Razak

**Role:** Project Supervisor

**Responsibilities:** Overseeing project. Weekly meetings will be held to ensure the work is on track and completed in a competent manner.

1. **Name:** Andrew Parker

**Role:** Research Guidance

**Responsibilities:** Provide guidance and helpful tips on database and object-orientated related tasks.

1. **Name:** Amanda Dewhurst & Louise Ashby

**Role:** Report & Literature Guidance

**Responsibilities:** Fortnightly meetings to review current project documents and ensure deadlines are correct and on track.

## [12.2] Literature Review

[1] Introduction

This project aims to analyse and determine the random nature of the social media platform ‘Twitter’ by parsing Tweets and using them to control the virtual robots in a programming game called ‘Robocode’. The data produced will be analysed to determine whether it is pseudo or truly random in an attempt to produce a new method of generating truly random numbers. The inspiration for this project originates from a social experiment called ‘Twitch Play Pokémon’.

Twitch is the world’s leading social video platform for streaming gaming-oriented content around the world. It was founded in June 2011 and currently amasses nearly 10 million visitors every day. The site is popular among both veteran and aspiring YouTubers and attracts the attention of big e-sports players and tournaments. (Twitch, 2017)

Twitch Plays Pokémon (TPP) was a social experiment by an anonymous Australian programmer and involves streaming a game of Pokémon on Twitch in the Game Boy emulator ‘VisualBoyAdvance’. The anonymous programmer developed an Internet Relay Chat (IRC) bot, written in Python, which captures messages from the Twitch chat and parses those strings directly into commands for the game. The stream soon became popular and reached approximately 60,000 concurrent playing users. It took nearly 16 continuous days to complete the first game ‘Pokémon Red’. Since then, numerous other Pokémon games from the franchise have been hosted and streamed on Twitch for the users to collaborate on. (Wikipedia, 2017) (Cunningham, 2014)

Robocode is a lightweight, open-source, programming game developed by Mathew A. Nelson as a personal endeavour in 2000. The goal is to create a virtual robot by writing the code yourself in Java or Microsoft .NET. The game acts as an intermediary between Twitter and the data visualisation by converting the parsed Tweets into instructions to control the robots and run the game. (Robocode, 2017)

Mathematics plays an important role in the conclusion of the project. Methods such as statistical regression and standard deviation should be used to determine the random nature of the results output from Robocode and graphed via D3.JS. This will conclude whether Twitter can be used as a TRNG.

This literature review intends to report on knowledge and ideas established from the given research findings. It assesses the relative strengths and weaknesses of such research and provides an opinion on the various findings. The latter end of the review evaluates and reflects on the research findings and justifies the choices that were ultimately made.

[2] Approach & Methods

The approach to the project requires a high degree of technical and mathematical knowledge. It makes extensive use of both object-orientated and procedural programming. Web development knowledge is also required which warrants the need for languages such as HTML, CSS, JavaScript and other libraries that provide helpful ‘out-of-the-box’ functionality. The required mathematical knowledge is mainly statistical. The author of this project has prior knowledge and experience with high-level statistics from studying Mathematics at A-Level and such skills are directly transferrable to this project.

In order to write a literature review, it is essential that a wide range of relevant literature is assessed to produce research on the given topic. This includes sources such as websites, books, journals and sampled research such as questionnaires. However, given the obscurity of this projects' specification, there is little scholarly research available online that is directly relevant to the scope of the project. This is not the case for all of the research topics covered, nonetheless, the majority of literature sources for this review are available online.

[3] Research Findings

The following section describes the research that was carried out on the project specifics. This includes discovering possible technologies and development methodologies.

[3.1] Twitch Plays Pokémon

Twitch Plays Pokémon is the social experiment that inspired the initial idea for this project. Therefore, it is important to research into the inner workings of the experiment to gain a better understanding of how it works and how it was developed.

The original concept was developed by an anonymous Australian programmer, so there is no official developer documentation available. However, what we do know is that the script responsible for parsing chat commands on Twitch was written in a programming language called Python. The script interfaced with a Gameboy emulator called ‘VisualBoyAdvance’. The script captures directional commands such as “A”, “B”, “Select” and “Start” which are sent to the emulator to control the game. An additional webpage was developed in JavaScript which was used to display a tally of moves that are shown within the stream. (Wikipedia, 2017)

This project takes this loose concept of controlling a game via an external influence but takes a tangential approach. Instead of text input from an IRC, the text contents of a Tweet will be used to control the Robots in Robocode.

[3.2] Software Development Methodologies

A software or system development methodology is a framework used to structure, plan and control the process of the development of an information system. The ‘traditional’ methodology is called ‘Waterfall’ but has since been superseded by newer modern methods that yield far greater efficiency, time management and cost savings. Such methodologies include Agile Software Development, Scrum, Rapid Application Development (RAD) and Extreme Programming (XP). There are many more out there all with their own advantages that cater to specific use cases. (itinfo, 2017)

Considering that all the aforementioned development methodologies are designed for software, *in theory*, any of them could be used and therefore be deemed ‘suitable’. However, given the modularity and highly technical nature of the project, there are a few which stand out from the crowd.

**Agile Software Development**

Agile Development is an umbrella term for several iterative and incremental software development methodologies. Some of the more popular agile methodologies include Extreme Programming (XP), Scrum, Crystal and Feature-Driven Development (FDD). Although there are various agile methodologies that have their own unique approach, they all ultimately have the same vision and core values. The fundamentals of an agile methodology include iteration with continuous feedback to successively refine and deliver a software system. (versionone.com, 2017)

Such methodologies allow changes to be made at any point during development and ensure customer satisfaction. They allow the project to adapt to changes in circumstances meaning the software can be dynamic. This can ultimately save time and money as other methodologies may require the project to be started from the beginning if the requirements change. (360logica, n.d.)

**Rapid Application Development (RAD)**

Rapid Application Development is another software development methodology which heavily emphasizes rapid prototyping and iterative delivery. This method is a sharp alternative to the conventional waterfall model which heavily relies on having absolute system requirements and sequential design practices. (Powell-Morse, 2016)

One of the major advantages of RAD is that the methodology itself was designed around software development. To elucidate, given that software development is malleable and in constant evolution, a methodology that revolves around iterative delivery and experimentation is a perfect match. (Powell-Morse, 2016)

**Feature-Driven Development (FDD)**

The third and final suitable methodology for this project is Feature Driven Development. This is another variant of agile methodology and begins by establishing an overall model shape. However, unlike other methodologies, FDD uses the idea of “design by feature, build by feature”. Short iterations of development produce these small features that are useful to the clients. Developing my feature is one of eight development practices. Others include; Domain Object Modelling; Configuration Management and Inspections. (McLaughlin, n.d.)

[3.3] Software Design Principles

There are many design principles, patterns and general etiquette which a programmer must conform to when writing high quality, extensible code. Referred to as the ‘First Five Object-Orientated Design Principles’, S.O.L.I.D is an acronym that stands for such. It stands for the following principles;

* **S**ingle Responsibility Principle (SRP)
* **O**pen Closed Principle (OCP)
* **L**iskov Substitution Principle (LSP)
* **I**nterface Segregation Principle (ISP)
* **D**ependency Inversion Principle (DIP)

To summarise each of these design principles;

The Single Responsibility Principle says that a class should have one and only one reason to change, meaning that a class should only have one job. The Open Closed Principle says that Objects or entities should be open for extension, but closed for modification. The Liskov Substitution Principle states that functions that use pointers or references to base classes must be able to use objects of derived classes without knowing it. The Interface Segregation Principle says that a client should never be forced to implement an interface that it doesn’t use, or clients shouldn’t be forced to depend on methods they do not use. Finally, the Dependency Inversion Principle states that entities must depend on abstractions, not on concentrations. High-level modules must not depend on the low-level module, but they should depend on abstractions. (Oloruntoba, 2015)

[3.4] Development Tools

Given the technical nature of this project, it is desirable from a development standpoint to choose tools that provide a smooth and efficient workflow. The first part of the project involves Java programming. Therefore, a Java Integrated Development Environment (IDE) is required.

An IDE is an application that facilitates software and application development. They are designed to encompass all programming tasks in one application. Subject to the language they include a code editor for writing source code. This differs from a regular text editor as it can provide intelligent suggestions to improve your code and has syntax highlighting to make code more readable. They have compilers built in and automatically compiles code each time you save, removing the need for the developers to manually compile from the command line. They also include debugging tools that can help developers find and fix problems in the code by ‘stepping through’ each line of execution to find the bug. Finally, some IDE’s have build automation tools that help automate some of the developers’ common tasks to increase workflow efficiency and save time. (Veracode, 2017)

Eclipse is a very popular IDE famous for Java but also supports C, C++ and PHP. It provides all of the desired functionality and advantages that a developer would expect from an IDE. (TheEclipseFoundation, 2017)

JetBrains, formerly IntelliJ, is a software development company who create tools targeted towards software developers and project managers. (Wikipedia, 2017) Their suite of software has vastly expanded and now has a respectable 4 million users and have been adopted by reputable companies such as Valve, Samsung and NASA. (JetBrains, 2017)

[3.5] Twitter Social Network

**Twitter Java Library**

The first major section of the project involves connect to the Twitter API in a Java program so that Tweets can be downloaded from a user’s timeline. This functionality is not natively supported in the Java API and so a third-party, community-written library is required to achieve this. The official Twitter Developer Documentation lists all known functioning libraries that work with their API and there is only one listed under Java. This library is called ‘Twitter4J’, meaning ‘Twitter for Java’ and was published by Twitter handle @yusuke. (TwitterDeveloper, 2017)

Although Twitter4J is an unofficial Java library for the Twitter API, it is open-source and is available to download from GitHub. The website provides all the necessary .JAR files that can simply be copied in the Java programs’ classpath to be used. The site also provides full Java Documentation (JavaDoc) which makes it easy to understand how the library works and how to use it. (Twitter4J, 2017)

**Constraints**

The Twitter API has a restriction called ‘Rate Limiting’. This means that each user and access token has 15 requests per rate limit window. The HTTP GET request window is 15 minutes (Twitter, 2017). To access a user timeline, the ‘getUserTimeline’ method returns an object upon invocation. This object contains useful information such as date posted, username, profile image and tweet text. Furthermore, this particular method only allows a maximum of 3200 objects to be returned. Therefore, assuming every account used has a posted at least 3200 tweets, a maximum of 48,000 (15 x 3200) Tweets can be downloaded per 15 minutes.

To overcome the rate limiting restriction, the Twitter Development Documentation suggests methods to avoid being rate limited. One suggestion is the use of ‘Caching’. This involves storing the API responses in the application or site if you expect to be using them a lot. The data can then be loaded from the local data store to ensure that calls are made to the API as infrequently as possible. (TwitterDeveloper, 2017)

[3.6] Robocode Programming Game

Robocode also imposes its own constraints on the project. Programming in Robocode is limited to either Java or a language from Microsoft’s .NET Framework such as C#. Subject to the chosen language, there are specific system requirements that must be met.

Java requires that the Java 6 Standard Edition (SE) or newer must be installed on the system. Both the Java Runtime Environment (JRE) and the Java Developer Kit (JDK) must be installed. There are also important environment variables that must be set up prior to running Robocode. Detailed specification can be found on the Robocode ReadMe on Sourcefource.net [here](http://robocode.sourceforge.net/docs/ReadMe.html). (Nelson, 2013)

.NET requires an API Plug-In to be installed on top of Robocode. This can be done by downloading the Robocode .Net .JAR file and running it. Furthermore, the .NET Framework itself needs installing which requires you to be running a Microsoft Windows Operating System and that the computer meets their minimum hardware specification requirements. (Microsoft, 2017)

[3.7] Database Technologies

A database is required to store the results from Robocode so they can be used further on in the project, making it a fundamental piece of technology.

Some examples of database types include; RDBMS; OODMS; ORDBMS and NoSQL. One of the more commonly used Database Management System (DBMS) is the Relational model, which can be used to manage transaction-orientated applications (OLTP). Conversely, non-relational databases, more commonly referred to as ‘NoSQL’, use document and column stores opposed to relations. (Foote, 2016)

This project is highly dependent on large datasets. Therefore, a database that is good at handling large datasets is desirable.

[3.8] Data Visualisation & Web Development

As previously mentioned, this project relies on large quantities of data. Therefore, in order to draw a conclusion from the findings, the data needs to be formatted and visualised so that a mathematical analysis can be conducted. Considering the data is stored electronically in a database, it is quick and easy to send the data to a webpage. Therefore, a charting library will need to be chosen to graphically represent the data objects on the webpage.

There is a wide range of programming libraries for charting available online. The vast majority of them are open-source and therefore freely available. There are many JavaScript libraries that are emerging as the most powerful tools for visualising data in the form of easy to understand, interactive charts. Some examples of the popular charting libraries are Chartist.js, FusionCharts, D3.JS, Dygraphs and Highcharts. (Singhal, 2015)

Many of the JavaScript libraries fit the requirements for the project. However, D3 (Data Driven Documents) is powerful, open-source and conforms to the W3C web standards and is compatible across multiple browsers. D3 is a JavaScript Library for manipulating the DOM in order to bring static data to life. The library manipulates HTML markup by creating SVG elements to create re-sizable data visualisations. (Bostock, 2017)

**Visualisation Format**

The format in which the data is visualised is quite important as it affects how the data can be analysed. Due to the nature of this project, and the agile software development methodology that was adopted, a lot of the latter research is still undergoing and has not been carried out yet. As mentioned in the next section ‘Mathematical Data Analysis’ there are particular statistical methods that will warrant certain visualisation formats. These are likely to include line graphs and scatter graphs.

**MEAN Stack Development**

Furthermore, to facilitate the use of the data visualisation library, a runtime environment will need to be chosen in order to run server-side JavaScript files so that data can be sent to the webpage to be rendered on the client-side.

The term MEAN stack refers to a collection of JavaScript based technologies used to develop web applications. MEAN is an acronym for MongoDB, ExpressJS, AngularJS and NodeJS. From the client to the server to the back-end database, MEAN is full stack JavaScript. (Raj, 2014)

Considering that the best database technology for this project is MongoDB, and that it makes use of JavaScript libraries for graphing functionality, it would make sense to use the MEAN Stack for the web development. Because all of the technologies are JavaScript orientated, it means that no further programming languages are required, making development easier and removing the need to learn new language syntax.

[3.9] Random Number Generation

When we think of randomness, most people think of a coin flip or a dice roll, but what does ‘Random’ actually mean? We can define randomness as something that is un-predictable and contains no recognisable patterns. (VSauce, 2014)

**Pseudo-Random Number Generation (PRNG)**

A PRNG is an algorithm that generates ‘random’ numbers. However, unlike a dice roll or a lottery ticket, the outcome is pre-calculated from tables of data to produce a sequence of numbers that appear to random. Modern PRNG algorithms are good enough now such that the number produced look exactly like they are random. (Haahr, 2017)

To illustrate, [Appendix 1](#_[7.1]_Appendix_1) shows two bitmaps produced by different methods. A bitmap is a digital image composed of a matrix of dots. The first bitmap (on the left) is produced by Random.org and uses atmospheric pressure to generate the values used to produce the bitmap. Conversely, the second bitmap (on the right) is produced by the pseudo-random **rand()** function from the PHP API in Microsoft Windows. As you see, this bitmap shows a clear pattern due to the repetition of generated numbers.

**True-Random Number Generation (TRNG)**

A TRNG is similar to a PRNG in as it generates random numbers. However, there is one fundamental difference. The values produced by a TRNG are ‘truly-random’, whereas previously mentioned, those from a PRNG are ‘pseudo-random’.

[3.10] Mathematical Data Analysis

The final section of this project, before a conclusion is formed, is the mathematical analysis of the data. One all the Twitter Tweets have been serialised, passed into the Robocode program, and then exported from the match results, the data will be ready to be visualised and then analysed.

The problem, however, is it is impossible to definitively prove whether a given sequence of numbers is truly-random. The pragmatic approach is to take several sequences of random numbers produced by the RNG in question, then subject them to a series of statistical tests. As the sequences pass more test, our confidence in them (and the generator) increases. However, we also should expect some to fail as random numbers also pose the chance to appear non-random. But if lots of the test fail, we should question the random nature of the generator. (Random.org, 2017)

One approach is a simple visual analysis. As previously mentioned, the data will be graphed via the D3.JS JavaScript Library. Humans are good at spotting patterns; therefore, a visualisation allows use to simply use our eyes and brain to directly analyse the data. Beyond this, statistical attributes such as standard deviation and variance will be measured to gain a more empirical conclusion. Standard deviation is the measure of how spread out numbers are. To calculate it, you simply square root the variance. The variance is calculated by finding the average of the squared differences from the mean. The sample size used is important as the accuracy increases with the size of the dataset. (MathsIsFun, 2017)

Another statistical method useful to the analysis of the data is linear regression. There are different types of regression, but it essentially attempts to model the relationship between two variables by fitting a linear equation (a straight line) to the observed data. The most common method for fitting a regression line is the method of least squares. This method calculates the best-fitting line for the data by minimising the sum of the squares of the vertical deviations from each data point to the line. This method shows any obvious outliers, the general correlation of the data points and how spread they are from the best-fit line. See [Appendix 2](#_[7.2]_Appendix_2) for an example of a line of best fit. (Anon., n.d.)

[4] Evaluation & Reflection

The following section evaluates the previous research findings and reflects on the technologies and methodologies discovered. Conclusions are made on the chosen findings and justification is made to back them up.

[4.1] A Different Direction from Twitch Plays Pokémon

Due to the current direction this project is heading and how its objectives differ vastly from Twitch Plays Pokémon, no further research will be conducted on the experiment as this project now has its own foundation. It is clear that Twitch Plays Pokémon was simply the initial spark for an idea that is now heading in its own direction.

[4.2] Agile Software Development Methodology

During the research, there was one common element that was apparent in every agile software development methodology. This element was a team. Considering that this project is being under-taken by a single developer, a lot of the key aspects of the aforementioned methodologies break down and no longer provide the advantages that they should. Alternatively, a different hybrid approach was taken that involved taking inspiration from lots of different agile methodologies and applying them to the project as necessary.

[4.3] SOLID Software Design Principles

There are quite literally thousands of software design principles that can be chosen and implemented by a program developer. However, given the nature of this project and the small scale of the Java program that will be written, the aforementioned five SOLID design principles will be considered during development. Subject to the size and complexity of the program, some principles may not need to be considered as their implementation may heed negligible results or even detriment to the program performance.

[4.4] JetBrains Software Development Suite

The chosen software for the development of this project was the JetBrains Software Development Suite. The first piece of software is ‘IntelliJ IDEA’. IntelliJ is an IDE specifically designed to maximise developer productivity. It includes useful features such as code completion, framework assistance and VCS integration such as Git. All Java programming and management will be done in IntelliJ. (JetBrains, 2017)

The next piece of software is ‘WebStorm’. This is another IDE/Editor that is designed specifically for a JavaScript ecosystem. It shares many of the same advantages and features as IntelliJ but is tailed more towards web, mobile, server and desktop development. (JetBrains, 2017)

Finally, ‘DataGrip’ is a database IDE that is designed for SQL developers. It also shares many of the features and advantages of an IDE, but is tailored towards database development. DataGrip allows the developer to add multiple database connections into the software where they can be managed and manipulated. Currently, DataGrip does not have native NoSQL support, but does have a plug-in that provides this feature. (JetBrains, 2017)

[4.5] Object Serialisation & Comma Separated Values

As mentioned earlier in the findings, there are limitations to the Twitter API. The major issue is with the rate limiting then prevents developers from making more than 15 calls to the API every 15 minutes. There are temporary solutions such as caching which stores the commonly used data in memory to reduce the number of calls made in the session window. However, considering that a large sample size is desirable with regards to data accuracy, a long-term solution is required.

The decision to ‘serialise’ the download Tweets was made. This means that the data is stored locally on the computers disk storage. The data will be formatted in CSV (Comma Separated Value) files. Each line of the file will contain the text from a single Tweet, where each character is delimited by a comma. Each file will contain 48,000 tweets from a single API window. A script can be configured to automate the process and could run every 15 minutes in order to collect a greater number of Tweets.

[4.6] Java Object Orientated Programming

The first major element of this project is handled via Object Orientated Programming. As mentioned in the findings, the virtual robots in Robocode can be programmed in either Oracle’s Java, or in Microsoft’s .NET Framework which utilises languages such as C#, F# or Visual Basic.

Both options provide no realistic performance advantage over the other, therefore, I decided to use Java as I am familiar with the language, it’s syntax, and the API. Furthermore, my chosen development suite, by JetBrains, contains a piece of software called ‘IntelliJ IDEA’ which natively supports Java meaning I can reap the benefits that the IDE provides. Conversely, opting with the .NET Framework would mean learning am entirely new API and having to download and use a separate piece of software to develop in.

[4.7] MongoDB Non-Relational Database

There are quite literally thousands of different database technologies freely available online to choose from. Fortunately, the use-case for this database isn’t very complex. This means that the usual rules you would follow when choosing a database, aren’t too relevant. For example, it is normally important to consider the end result. This can reduce the change of any unpredictable queries popping up and causing serious performance problems. Considering that the queries used in this project are simply dumps of JSON data, consideration of the end result is not important. (Heller, 2016)

The document database ‘MongoDB’ was chosen for the project as it had many advantages over the conventional relational database. Firstly, MongoDB is ‘schema-less’, meaning it does not use schema models. This means that the structure of the data is not strict, and so each datum can have a different number of fields, content and sizes from the others. (tutorialspoint, 2017)

Moreover, MongoDB has a deep query ability using its document-based query language and also scales very well with size. Considering the huge quantity of Tweets that will be stored, of varying lengths, MongoDB caters for it very well.

To summarise, the research has shown that MongoDB is good for big data, clear object structure and deep querying and has therefore been selected as the database of choice for this project.

[4.8] D3.JS Data Visualisation JavaScript Library

D3.JS (Data Driven Documents – JavaScript Library) was chosen for a number of reasons. Firstly, it is very flexible and works seamlessly with existing web technologies and can manipulate any part of the Document Object Model (DOM). This gives it huge advantages over other libraries because it can be displayed however you like and isn’t limited to small regions of a webpage. (Skau, 2013)

D3 is also quick and easy to use as it can be linked and used in an HTML file using one line of markup. Finally, as mentioned earlier in the research findings, D3 also conforms to the W3C web standards and is compatible across all browsers which will ultimately eliminate any potential issues regarding compatibility. (Singhal, 2015)

[4.9] Differentiating PRNG and TRNG

Now we know the difference between pseudo and truly-random number generation, it is important to keep this information in mind during the project. This is especially the case towards the latter end of the project when all the data has been collected and is stored in the database. The mathematical analysis will produce lots of empirical data that will ultimately form the conclusion. However, the values produced from the statistical analysis won’t *really* mean anything without thinking about what they mean and placing them somewhere on the line between pseudo and true.

[4.10] Linear Regression & Standard Deviation

All of the methods in the research findings will be employed for the mathematical analysis of the data. This includes the aforementioned statistical methods such as Regression and Standard Deviation. At the time of writing this, the ‘method of least squares’ is planned to be used. However, future research may warrant the use of another type of regression. This applies to the mathematical analysis in general.

As research shows, there are lots of methods for determining the random nature of a set of numbers, so future research will dictate which methods are chosen to conclude the project.

[5] Version Log

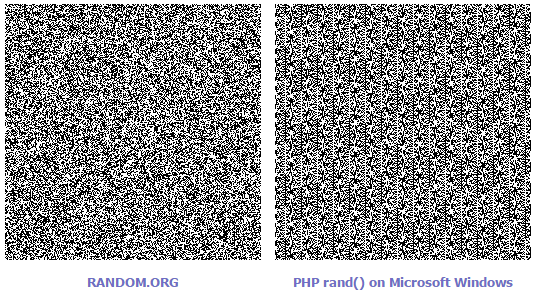
|  |  |  |
| --- | --- | --- |
| **Version** | **Description** | **Date** |
| 1 | First Publication |  |

[6] Glossary of Terms

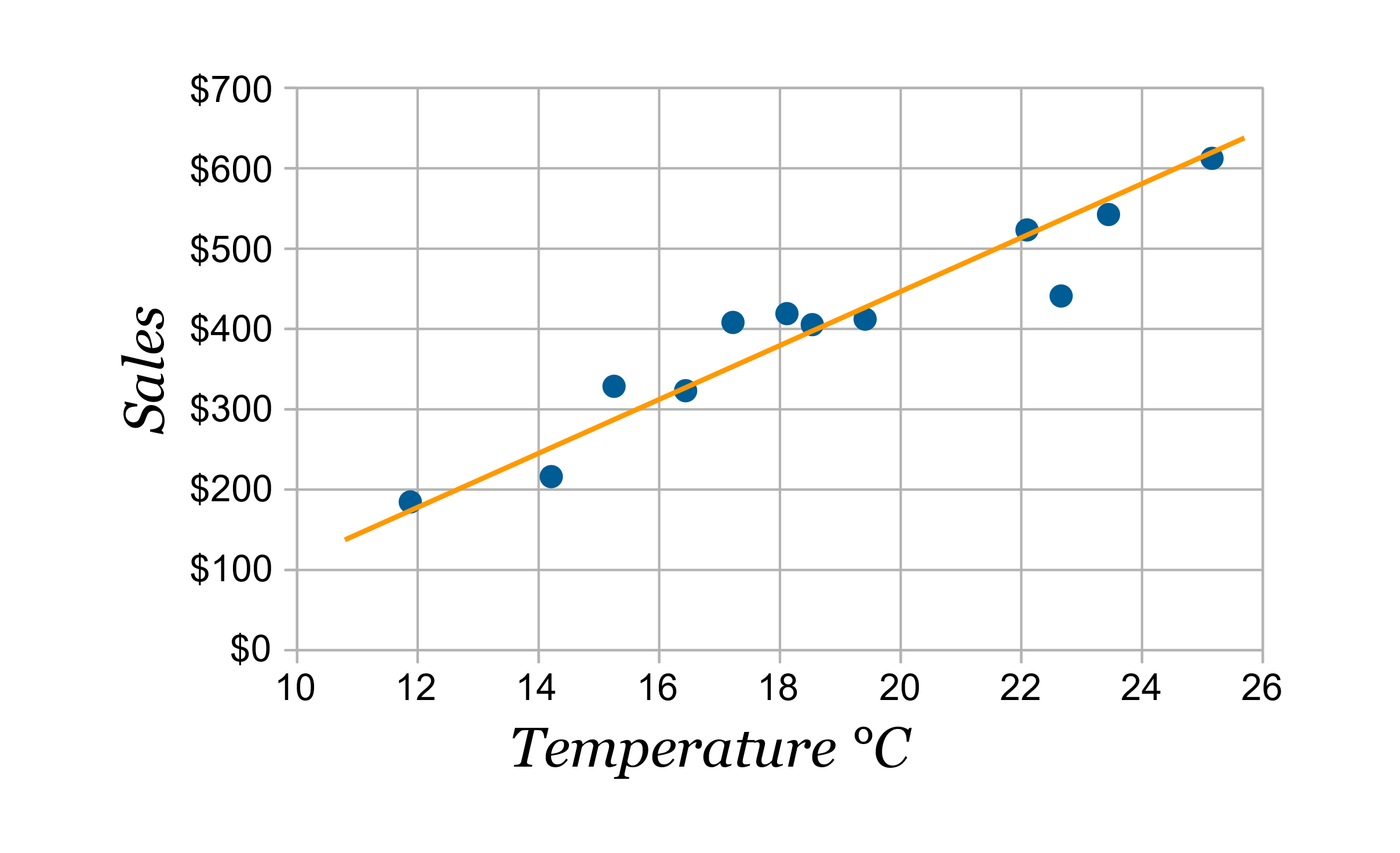
|  |  |
| --- | --- |
| **Term** | **Definition** |
| AAA | Pronounced ‘Triple-A’, refers to the information classification of mid to major publishers of video games. |
| Agile | Relating to or denoting a method of project management, used especially for software development, that is characterized by the division of tasks into short phases of work and frequent reassessment and adaptation of plans. |
| API | Application Programming Interface. A set of functions and procedures that allow the creation of applications which access the features or data of an operating system, application, or other service. |
| C | C is a high-level and general-purpose programming language that is ideal for developing firmware or portable applications. |
| C# | Pronounced “C-Sharp”. An object-oriented programming language from Microsoft that aims to combine the computing power of C++ with the programming ease of Visual Basic. C# is based on C++ and contains features similar to those of Java. |
| C++ | ++ is a general-purpose object-oriented programming (OOP) language, developed by Bjarne Stroustrup, and is an extension of the C language. It is therefore possible to code C++ in a "C style" or "object-oriented style." |
| Classpath | Classpath is a parameter in the Java Virtual Machine or the Java compiler that specifies the location of user-defined classes and packages. The parameter may be set either on the command-line, or through an environment variable. |
| D3 (.JS) | D3.JS (Data Driven Documents) is a JavaScript library used for creating SVG Data Visualisation on web pages. |
| DOM | Document Object Model. A programming API for HTML and XML documents. It defines the logical structure of documents and the way a document is accessed and manipulated. |
| F# | A strongly typed, multi-paradigm programming language that encompasses functional, imperative, and object-oriented programming methods. |
| FDD | Feature Driven Development. an iterative and incremental software development process. It is a lightweight or Agile method for developing software. FDD blends a number of industry-recognized best practices into a cohesive whole. |
| Game Boy | the first handheld electronic device for playing cartridge games; also called Game Boy Color, Game Boy Advance, Nintendo DS. Usage Note. trademark of Nintendo. Dictionary.com's 21st Century Lexicon. |
| GET | An HTTP command used to request a file from a Web server. GET is widely implemented in HTML files (Web pages) for making database queries that do not involve any updating at the server side. |
| Git | A version control system for tracking changes in computer files and coordinating work on those files among multiple people. It is primarily used for source code management in software development, but it can be used to keep track of changes in any set of files. |
| Handle | In the context of the social media platform ‘Twitter’, Each Twitter handle has a unique URL, with the handle added after twitter.com. Example: http://twitter.com/username. Hashtag -- A Twitter hashtag refers to a topic, keyword or phrase preceded by the # symbol. An example is #skydivinglessons. Hashtags are used to categorize messages on Twitter. |
| HTML | Hyper-Text Markup Language. A standardized system for tagging text files to achieve font, colour, graphic, and hyperlink effects on World Wide Web pages. |
| HTTP | Hyper-Text Transfer Protocol. HTTP is the underlying protocol used by the World Wide Web and this protocol defines how messages are formatted and transmitted, and what actions Web servers and browsers should take in response to various commands. |
| IDE | Integrated Development Environment. A piece of software used for managing and developing applications. |
| IRC | Internet Relay Chat. An application layer protocol that facilitates communication in the form of text. |
| Java | a general-purpose computer programming language designed to produce programs that will run on any computer system. |
| Javadoc | Javadoc (originally cased JavaDoc) is a documentation generator created by Sun Microsystems for the Java language (now owned by Oracle Corporation) for generating API documentation in HTML format from Java source code. |
| JavaScript | An object-oriented computer programming language commonly used to create interactive effects within web browsers. |
| JetBrains | JetBrains is a software development company whose tools are targeted to software developers and project managers. They have a large suite of intelligent software used for programming and web development. |
| MEAN | a collection of JavaScript based technologies used to develop web applications. MEAN is an acronym for MongoDB, ExpressJS, AngularJS and Node.js. From client to server to database, MEAN is full stack JavaScript. |
| NASA | National Aeronautics and Space Administration, the federal agency that is responsible for aerospace research, aeronautics, and the civilian space program. |
| NoSQL | A NoSQL (originally referring to "Non-SQL" or "non-relational") database provides a mechanism for storage and retrieval of data that is modelled in means other than the tabular relations used in relational databases. |
| OLTP | Online Transaction Processing. A class of software programs capable of supporting transaction-oriented applications on the Internet. Typically, OLTP systems are used for order entry, financial transactions, customer relationship management (CRM) and retail sales. |
| OODBMS | Object-Orientated Database Management System. A database management system that supports the creation and modelling of data as objects. OODBMS also includes support for classes of objects and the inheritance of class properties, and incorporates methods, subclasses and their objects. |
| OOP | Object Orientated Programming. A programming language model organized around objects rather than "actions" and data rather than logic. Historically, a program has been viewed as a logical procedure that takes input data, processes it, and produces output data. |
| ORDBMS | Object Relational Database Management System. A database management system (DBMS) similar to a relational database, but with an object-oriented database model: objects, classes and inheritance are directly supported in database schemas and in the query language. |
| PHP | Hyper-Text Processor. A script language and interpreter that is freely available and used primarily on Linux Web servers. |
| PRNG | Pseudo-Random Number Generation. A program written for, and used in, probability and statistics applications when large quantities of random digits are needed. |
| Python | An interpreted, object-oriented programming language developed by Guido van Rossum. The name comes from one of van Rossum's favourite television shows, Monty Python's Flying Circus. Python is very portable since Python interpreters are available for most operating system platforms. |
| RAD | Rapid Application Development. A suite of software development methodology techniques used to expedite software application development. RAD uses predefined prototyping techniques and tools to produce software applications. |
| RDBMS | Relational Database Management System. a database management system (DBMS) that is based on the relational model invented by Edgar F. Codd, of IBM's San Jose Research Laboratory. |
| Robocode | Robocode is a programming orientated game, where the goal is to develop a robot battle tank to battle other tanks in Java or Microsoft .NET. |
| Scrum | A framework for managing software development. It is designed for teams of three to nine developers who break their work into actions that can be completed within fixed duration cycles (called "sprints"), track progress and re-plan in daily 15-minute stand-up meetings, and collaborate to deliver workable software every sprint. |
| SVG | Scalable Vector Graphics (SVG) is an XML-based vector image format for two-dimensional graphics with support for interactivity and animation. The SVG specification is an open standard developed by the World Wide Web Consortium (W3C) since 1999. SVG images and their behaviours are defined in XML text files. |
| TPP | Twitch Plays Pokémon. A "social experiment" and channel on the video game live streaming website Twitch, consisting of a crowdsourced attempt to play Game Freak's and Nintendo's Pokémon video games by parsing commands sent by users through the channel's chat room. |
| TRNG | True-Random Number Generation. A mathematical construct, either computational or as a hardware device, that is designed to generate a random set of numbers that should not display any distinguishable patterns in their appearance or generation. |
| Twitch | Twitch is a live streaming video platform owned by Twitch Interactive, a subsidiary of Amazon.com. In 2015, Twitch announced it had more than 1.5 million broadcasters and 100 million visitors per month. |
| Twitter | witter is a free social networking microblogging service that allows registered members to broadcast short posts called tweets. |
| URL | Uniform Resource Locator. A protocol for specifying addresses on the Internet. |
| VCS | Version Control System. A component of software configuration management, version control, also known as revision control or source control, is the management of changes to documents, computer programs, large web sites, and other collections of information. |
| Visual Basic | Commonly abbreviated to VB, Visual Basic is a programming environment from Microsoft in which a programmer uses a graphical user interface (GUI) to choose and modify preselected sections of code written in the BASIC programming language. |
| Windows | The most widely used operating system for desktop and laptop computers. Developed by Microsoft, Windows primarily runs on x86-based computers (the ubiquitous PC), although versions have run on Intel's Itanium CPUs. |
| XP | Extreme Programming. A software development methodology which is intended to improve software quality and responsiveness to changing customer requirements. |

[7] Appendices

[7.1] Appendix 1 – True-Random Bitmap vs Pseudo-Random Bitmap



[7.2] Appendix 2 – Linear Line of Best Fit



## [12.3] What is Twitch?

Twitch is the world’s leading social video platform for streaming gaming oriented content around the world. It was founded in June 2011 and currently amasses nearly 10 million visitors every day. The site is popular among both veteran and aspiring YouTubers and attracts the attention of big e-sports players and tournaments. (Twitch, 2017)

## [12.4] What is Robocode?

Robocode is a lightweight, open-source, programming game developed by Mathew A. Nelson as a personal endeavour in 2000. The goal is to create a virtual robot by writing the code yourself in Java or Microsoft .NET. The game acts as an intermediary between Twitter and the data visualisation by converting the parsed Tweets into instructions to control the robots and run the game. (Robocode, 2017)

## [12.5] What is Twitch Plays Pokémon?

Twitch Plays Pokémon (TPP) was a social experiment by an anonymous Australian programmer and involves streaming a game of Pokémon on Twitch in the Game Boy emulator ‘VisualBoyAdvance’. The anonymous programmer developed an Internet Relay Chat (IRC) bot, written in Python, which captures messages from the Twitch chat and parses those strings directly into commands for the game. The stream soon became popular and reached approximately 60,000 concurrent playing users. It took nearly 16 continuous days to complete the first game ‘Pokémon Red’. Since then, numerous other Pokémon games from the franchise have been hosted and streamed on Twitch for the users to collaborate on. (Wikipedia, n.d.) (Cunningham, 2014)