A WebGL-based aerobatic visualiser using the OLAN(One letter aerobatic notation) catalogue to provide users with a means of generating interactive 3D representations of manoeuvres.

Final Report for CS39440 Major Project

Author: Craig Heptinstall (crh13@aber.ac.uk)

Supervisor: Prof. Neal Snooke (nns@aber.ac.uk)

29th March 2015

Version: 1.0 (Draft)

This report was submitted as partial fulfilment of a MEng degree in Software Engineering (G601)

Department of Computer Science

Aberystwyth University

Aberystwyth

Ceredigion

SY23 3DB

Wales, UK

Declaration of originality

In signing below, I confirm that:

- This submission is my own work, except where clearly indicated.

- I understand that there are severe penalties for plagiarism and other unfair practice, which can lead to loss of marks or even the withholding of a degree.

- I have read the sections on unfair practice in the Students' Examinations Handbook and the relevant sections of the current Student Handbook of the Department of Computer Science.

- I understand and agree to abide by the University's regulations governing these issues.

Signature ....

Date ....

Consent to share this work

In signing below, I hereby agree to this dissertation being made available to other students and academic staff of the Aberystwyth Computer Science Department.

Signature ....

Date ....

Acknowledgements

I am grateful to...

I'd like to thank...

Abstract & Background

The aim of this project is to create a 3D representation of aerobatic manoeuvres, primarily using the OLAN notation(or one letter aerobatic notation). Both real-scale and remote control aerobatic planes and helicopters use the notation to describe a set of manoeuvres in an overall routine, usually including translating the notations into Aresti symbols. The Aresti symbols came before OLAN, but OLAN was developed to make it possible for pilots to write down quickly their planned routines.

The primary element of this project will involve allowing users to insert their notated routines into the application(via an input box on a web-page) thus producing first the ribbon shapes of the manoeuvres, followed by the ability to see a craft fly the route. Although there is only a finite amount of manoeuvres possible from the Aresti catalogue, each OLAN notation can have its own parameters. This can range from entry length into a loop or turn, to the number of rolls in a section of a manoeuvre. The application will also need to be taking into account flight speeds, and possibly wind and gravitational effects. Due to a long list of possible features, the application will be created in a feature-by-feature means (FDD).

WebGL will be the language used to create this application, with hopefully object-orientated Javascript to power the application. Libraries helping towards the graphical/ visual side of the application may be required to give greater flexibility and better aesthetic value. All considerations will be found in the analysis and design stages of this report.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter 1

Background & Objectives

Before commencing the design of the application and the project planning, it is important to have analysed what I hope to have achieved at the end of my project time, and also what steps I will need to be taking to implement each feature. As I will mention later, choosing the best fitting life cycle methodology will play a big part of how I shape my project and create each feature whether it be by priority, size or difficulty. This section details my understanding for my project requirements, steps I am going to need to take, and as I would prefer my project to be similar to and FDD one; developing an overall model and building the list of requirements and features.

1.1 Background

The choice of undertaking a project such as this one was due to two combining factors: maths and an interest to learn graphical programming. The fact that this application will require me to learn graphics, and how to implement visual effects representing the requirements of the project in ways completely new to myself. As graphics is something I have not had much to do with in the past, this project appears both exciting and daunting task due to the learning curve I will need to take. As for the maths factor, I can assume quite a lot of maths will be involved(especially for creating curves, rolls and turns along most of the manoeuvres) which I enjoy learning about.

In terms of the history of the topic, OLAN was originally developed by Michael Gorden in **2006** and was designed to provide shorthand notation for pilots planning out aerobatic routines without having to draw out the full Aresti diagrams. In recent years, the OLAN notation became used much more until because of licensing issues with the original owner was taken offline. Because of this, a new form of the notation has been created in a more open source way paving the way for applications such as this project's intended aim. Although in this report and my planned application itself will be still referring the notation as OLAN, the new re-make of the language is known as the **'OpenAero language'**. This is based off of the original, yet is open and allows anyone to use it. In combination with this, the creators of the OpenAero language also developed a web-based **application** that allows the conversion of the notations to 2D Aresti diagrams. This is somewhat similar to what I hope to achieve, but alongside plenty more features most importantly the ability to see a plane perform the moves.

As for Aresti, named after its conceiver Jos Luis Aresti Aguirre **https://www.aerobatics.org.uk/aresti** is the diagram format that OLAN achieves, and represent informative diagrams showing the shape

of the routine, direction of travel, rolls and sharpness of turns. Aresti diagrams also can include angles or turns, ranging from 90 degrees to 270. Each diagram usually has a name, relating normally to the shape of the manoeuvre, though some are more commonly known to pilots rather than the regular user. The OLAN notation for each diagram usually attempts to try describe the manoeuvre with the letter used, such as 'o' for a loop, or 'z' for a shark tooth. The full list of manoeuvres, including their OLAN notation and full name can be found on the **OpenAero site**.

Upon starting this project, several meetings with the project supervisor are planned each providing more detail of the initial requirements. Each of the meetings will be found in the Gantt for the project, with a smaller document in the **appendix** showing the results of each meeting. Because I have already attended several meetings at this stage, I can provide a fairly accurate list of initial requirements of the application.

The initial required application can be broken down into an extensive(but less detailed) list of main functional requirements. These are as follows:

1. Provide a web-implemented tool that allows input of the OLAN 1 None-IE due to WebGL capabilities. Will it use a simple JSON file to store notations? characters as a string format, alongside possible click functionality.

2. Relate each notation or set of notations to a certain procedural movement(rotations, movements etc.).

   - Must consider parameters in some of the notations, such as the speed of entry.

3. Provide a means of linking up these movements in such a way into moves, or the angle of the plane. They should produce a fluid manoeuvre.

4. Display this using WebGL. Libraries to consider that could help. Begin by initially testing simple shapes to move and fly around, then add textures, and plane structure.

5. Are libraries OK to use? with some of the movements:

   - glMatrix- JavaScript library for helping with performing actions to matrices- http://glmatrix.net
   - ThreeJS- Another JavaScript library, good with handling cameras and different views- http://threejs.org

6. Allow user to add different effects such as wind, gravity changes and other physics. Could be better to implement these last, as it will be easier to test pure functionality of rolls etc. first, then figure out natural physics.

7. Add functionalities of different viewpoints(on-board views, side views) to application.

8. Possibility to add function to save (using local storage?) users different sets of manoeuvres?

The list shown above also has an **accompanying** report which I created after my first project meeting. This report includes the list of initial requirements, alongside footnotes, and also a detailing of the methodology and process I plan to follow. The document can be found in the appendices section of this report.

INCLUDE PICTURE OF USER STORIES

1.2   Analysis

Before I begin using my chosen life cycle model, It is important I analyse the overall model and requirements of the intended application. I plan to analyse two items: the requirements analysed by time, effort and difficulty, and also a breakdown of the OLAN and Aresti manoeuvres. The second of which I will break down to their primitive forms, hopefully finding out how I can make my application create each manoeuvre as simple and efficiently as possible.

1.2.1   OLAN and Aresti interpretations

The best place to begin with my analysis is to look in more detail at the OLAN manoeuvres individually, by breaking down each manoeuvre into their primitive elements. As with the previous section, alongside I have attached a document to the appendices of this report, detailing the main and most important manoeuvres I believe are key to finding primitive shapes. I began this by firstly organising each OLAN and their corresponding Aresti shapes into sub groups.

Of these, there were:

- Single element- These include manoeuvres that can be described as one fluid movement. For instance, the OLAN letter 'd' would mean diagonal line up, which requires only one action to complete the manoeuvre.

- Two-element- This group includes any manoeuvres that require two separate manoeuvres to complete a given manoeuvres. An example of this could be the 'z' notation, also known as a shark tooth. This shape requires both a diagonal line up followed by a vertical line down.

- Loops- These like the single element moves, consist of a single manoeuvre, and can be combined to make other manoeuvres.

- Loop-line combinations- These are loops that are a combination of a loop and a single or two-element manoeuvre.

- Double-loops- As the name states, these are manoeuvres that contain two loops.

- Humpty-Dumpties, Hammerheads and Tailslides- Each of these manoeuvres represent specific shapes, such as a 'humpty dumpty' which consists of a bump shape comprised of a 180 degree turn to come back down vertically. As the previous, these shapes can simply be combined from single element pieces.

- Complex 3 rolling elements- The naming behind this group of manoeuvres comes from the fact that each contain a set of 3 elements to create the entire figure.

- Special and 'oddball' figures- The group of manoeuvres that are more complex in such a way that they require special sections not available from any of the other groups. One example of this is the OLAN letter 'f' which represents a flick. This comprises of rolling the aircraft 360 degrees along its horizontal line.

Another important part of the OLAN analysis that I had to understand before proceeding was the possible parameters, prefixes and postfixes that can be attached to manoeuvres.

Looking at prefixes first, each one-letter-notation, **some** moves are able to be reversed, or inverted. These are so:

- r - Reverse, meaning to order of how each part of the manoeuvre is done. For example, placing the letter 'r' before 'c', would represent a Cuban loop flown in the opposite order.

- i - Inverse, meaning that each part of the manoeuvre is done in the same order, but inverted in terms of value. For example, the letter 'i' before 'c' would mean that rather than looping upwards, the loop would go down. A diagonal line upwards would become a diagonal line downwards.

- ir - This is simply a mixture of both the previous. The manoeuvre fixed to the end of this postfix would both be inverted and then reversed.

In addition, there are a number of postfixes that can be used with some manoeuvres particularly with roll or turn based figures. For example, manoeuvres containing rolls can be represented with the prefix angle of turn in multiples of 90 degrees, and a postfix of the amount of rolls along the same part of the path. So in one example, the notation '2j2' would represent a 180 degree turn, whilst rolling the aircraft twice. Alike the prefixes for inverting and reversing manoeuvres, the parameters are not available on all the manoeuvres in the OLAN catalogue. Again, the full list can be found on the the OpenArea site, or see the my OLAN understandings in my **appendices**.

One final set of optional parameters that could be required of my application to handle are the positions of manoeuvres. Although not strictly in part of the OLAN catalogue, it is already available in the OpenAero application. These parameters are structured (x,y) with x representing the amount of horizontal distance and y the vertical distance from the end of the previous manoeuvre to the start of the current manoeuvre. These can also be negative values to ensure the user can control the position fully.

The main reason for the need of this is that simply all manoeuvres cannot fully follow each other straight after each other. In real-life if a manoeuvre made the pilot finish near the ground, and the next move required them to perform a diagonal line down, they would hit the ground. Obviously my application will attempt some form of validation and checks, but offering the option to the user is a very useful feature.

From my analysis in the groups listed above, a number of assumptions can be made.

1. I have already deduced that all none-single manoeuvres that do not include loops or rolls should be possible to be made from a set of single element manoeuvres.

2. In total, any manoeuvre can be constructed from one of three primary moves: diagonal and straight lines, curved lines, and turns and rolls. Each of which should be able to carry parameters.

3. Each curve should be possible to be created based on 45 degree increments, as this is the smallest change of angle in any manoeuvre, and all other angles seem to be in multiples of this number. This will shorten the need for multiple commands for different ranges of angles when programming in the manoeuvres.

4. Turns, curves and rolls will all need parameters, as some curves are steeper and shorter than others. The same goes for rolls, where you can choose anything from quarter to 3 rolls, and in turns the angle of turn should be specifiable.

By considering my analysis of the set of manoeuvres, I can now envision what manoeuvres will be possible to create easiest, and prioritise my work better. The next section of this report will group up the functionality of the application with my OLAN manoeuvre findings.

1.2.2 Application functionality interpretations

Upon having my initial meetings with the project supervisor, and on analysis of the OLAN catalogue, I can now make more final judgements on what I want to have been achieved by the end of this project. As I discussed in the background section, I had already created a set of initial assumed requirements. Though, some of the features raised questions, such as the possibilities to use libraries for the physics and graphical side. After some research I can now define a more cohesive and stable list of features and requirements of my application. Rather than in a list format though, here I will group features together and discuss the most important factors.

INCLUDE PICTURE OF USER REQUIREMENTS

Starting with the underlying functionality based on my OLAN interpretations, what I hope to be achievable by the end of this project is the possibility to allow users to enter any length of space separated OLAN characters into an input box, alongside any possible prefixes, postfixes and parameters. I hope to achieve this by making my system as general as possible, and by using very abstract methods that can support any given move input. One way I could think of implementing this now would be to break each manoeuvre into a set of instructions, each one saying which direction to move, the angle and length. By breaking each manoeuvre into these, I could use a single method to construct each manoeuvre on the fly. This is important as it would then allow for the user to play through the manoeuvre in an animated fashion and see the aircraft move.

More generally, one of the questions asked by myself at the start of this project was if libraries could be used to help create the graphics and physics. By now, I have had more meetings with my supervisor and found this was allowable. The main library I have considered after this has been the **three.js** library which acts as a good wrapper for controlling a wide range of objects in a scene. This library will allow me to easily manipulate vectors, cameras and lighting which will form the basis of my application. I will discuss the use of this library when it comes to planning each of the features in my design.

This brings me to the scene and cameras that I hope to have working to a good standard. For the scene, things such as the ground terrain are not so important, and can simply represent a flat land, while lighting could be added, but perhaps after fundamental features are added. As for cameras, I would like it that there is a set of two cameras: one for navigating and viewing flight paths at different locations and angles, and another camera which would be on-board, like a nose-cam.

The save/ loading of flight paths is a lower ranked task, but something I will definitely be planning to have implemented by the end of the project. Having looked into various methods of saving the OLAN entries, the best way I have found is to use a combination. One which would use local storage, and one that would export to JSON. The first of which I found out **here** is particularly useful as my project is web based.

There are four features I would also like to add, but I will make these optional for now, and place these after the previously mentioned features. The first of which would be to validate the manoeuvre entry. Currently in the OpenAero application there is a check that looks how close and where manoeuvres are placed on the canvas, and for my application it would be ideal to have a

check that looks if manoeuvres are actually possible from the current rotation or placement of the last manoeuvres ending position. Another big check would be if the current path of the aircraft was to hit the floor, a check should be made.

Physics is also an option I would like to include, such as wind, type of aircraft(each aircraft could be more difficult to navigate corners, meaning wider curves). And relating back to the function of playing the animation once the manoeuvres are drawn, an idea passed onto me from my supervisor was of a 'movie-reel' function that would show a mini image of the current manoeuvre being played, showing the animation progressing through each one. Again, this is another feature I would prioritise less, and implement after other key features.

Finally, a more smaller task I think I should set myself is to make the application mobile compatible. As most phones also now have WebGL capabilities, making the application run on mobiles should be possible. This is more a GUI centred feature on the site itself though, and should be added nearer the end of the project.

PICTURE OF RANKINGS OF ITEMS a bit like scrum.

1.3   Process

Moving onto more project and time management specific items, the process in which I follow can have a large effect on what features I complete on time.

Chapter 2

Design

You should concentrate on the more important aspects of the design. It is essential that an overview is presented before going into detail. As well as describing the design adopted it must also explain what other designs were considered and why they were rejected.The design should describe what you expected to do, and might also explain areas that you had to revise after some investigation.Typically, for an object-oriented design, the discussion will focus on the choice of objects and classes and the allocation of methods to classes. The use made of reusable components should be described and their source referenced. Particularly important decisions concerning data structures usually affect the architecture of a system and so should be described here.How much material you include on detailed design and implementation will depend very much on the nature of the project. It should not be padded out. Think about the significant aspects of your system. For example, describe the design of the user interface if it is a critical aspect of your system, or provide detail about methods and data structures that are not trivial. Do not spend time on long lists of trivial items and repetitive descriptions. If in doubt about what is appropriate, speak to your supervisor. You should also identify any support tools that you used. You should discuss your choice of implementation tools - programming language, compilers, database management system, program development environment, etc.Some example sub-sections may be as follows, but the specific sections are for you to define.

2.1 Overall Architecture

2.2 Some detailed design

2.2.1 Even more detail

2.3 User Interface

2.4 Other relevant sections

Chapter 3

Implementation

The implementation should look at any issues you encountered as you tried to implement your design. During the work, you might have found that elements of your design were unnecessary or overly complex; perhaps third party libraries were available that simplified some of the functions that you intended to implement. If things were easier in some areas, then how did you adapt your project to take account of your findings?It is more likely that things were more complex than you first thought. In particular, were there any problems or difficulties that you found during implementation that you had to address? Did such problems simply delay you or were they more significant? You can conclude this section by reviewing the end of the implementation stage against the planned requirements.

Chapter 4

Testing

Detailed descriptions of every test case are definitely not what is required here. What is important is to show that you adopted a sensible strategy that was, in principle, capable of testing the system adequately even if you did not have the time to test the system fully.Have you tested your system on real users? For example, if your system is supposed to solve a problem for a business, then it would be appropriate to present your approach to involve the users in the testing process and to record the results that you obtained. Depending on the level of detail, it is likely that you would put any detailed results in an appendix.The following sections indicate some areas you might include. Other sections may be more appropriate to your project.

4.1   Overall Approach to Testing

4.2   Automated Testing

4.2.1   Unit Tests

4.2.2   User Interface Testing

4.2.3   Stress Testing

4.2.4   Other types of testing

4.3   Integration Testing

4.4   User Testing

Chapter 5

Evaluation

Examiners expect to find in your dissertation a section addressing such questions as:

- Were the requirements correctly identified?

- Were the design decisions correct?

- Could a more suitable set of tools have been chosen?

- How well did the software meet the needs of those who were expecting to use it?

- How well were any other project aims achieved?

- If you were starting again, what would you do differently?

Such material is regarded as an important part of the dissertation; it should demonstrate that you are capable not only of carrying out a piece of work but also of thinking critically about how you did it and how you might have done it better. This is seen as an important part of an honours degree. There will be good things and room for improvement with any project. As you write this section, identify and discuss the parts of the work that went well and also consider ways in which the work could be improved. Review the discussion on the Evaluation section from the lectures. A recording is available on Blackboard.

Appendices

Appendix A

Third-Party Code and Libraries

If you have made use of any third party code or software libraries, i.e. any code that you have not designed and written yourself, then you must include this appendix.

As has been said in lectures, it is acceptable and likely that you will make use of third-party code and software libraries. The key requirement is that we understand what is your original work and what work is based on that of other people.

Therefore, you need to clearly state what you have used and where the original material can be found. Also, if you have made any changes to the original versions, you must explain what you have changed.

As an example, you might include a definition such as:

Apache POI library The project has been used to read and write Microsoft Excel files (XLS) as part of the interaction with the clients existing system for processing data. Version 3.10-FINAL was used. The library is open source and it is available from the Apache Software Foundation [2]. The library is released using the Apache License [1]. This library was used without modification.

Appendix B

Code samples

2.1 Random Number Generator

The Bayes Durham Shuffle ensures that the psuedo random numbers used in the simulation are further shuffled, ensuring minimal correlation between subsequent random outputs [6].

```
#define IM1 2147483563
#define IM2 2147483399
#define AM (1.0/IM1)
#define IMM1 (IM1-1)
#define IA1 40014
#define IA2 40692
#define IQ1 53668
#define IQ2 52774
#define IR1 12211
#define IR2 3791
#define NTAB 32
#define NDIV (1+IMM1/NTAB)
#define EPS 1.2e-7
#define RNMX (1.0 - EPS)

double ran2(long *idum)
{
  /*----------------------------------------------------*/
  /* Minimum Standard Random Number Generator           */
  /* Taken from Numerical recipies in C                 */
  /* Based on Park and Miller with Bays Durham Shuffle */
  /* Coupled Schrage methods for extra periodicity      */
  /* Always call with negative number to initialise     */
  /*----------------------------------------------------*/

  int j;
  long k;
  static long idum2=123456789;
```

```
static long iy=0;
static long iv[NTAB];
double temp;

if (*idum <=0)
{
  if (-(*idum) < 1)
  {
    *idum = 1;
  }else
  {
    *idum = -(*idum);
  }
  idum2=(*idum);
  for (j=NTAB+7;j>=0;j--)
  {
    k = (*idum)/IQ1;
    *idum = IA1 *(*idum-k*IQ1) - IR1*k;
    if (*idum < 0)
    {
      *idum += IM1;
    }
    if (j < NTAB)
    {
      iv[j] = *idum;
    }
  }
  iy = iv[0];
}
k = (*idum)/IQ1;
*idum = IA1*(*idum-k*IQ1) - IR1*k;
if (*idum < 0)
{
  *idum += IM1;
}
k = (idum2)/IQ2;
idum2 = IA2*(idum2-k*IQ2) - IR2*k;
if (idum2 < 0)
{
  idum2 += IM2;
}
j = iy/NDIV;
iy=iv[j] - idum2;
iv[j] = *idum;
if (iy < 1)
{
  iy += IMM1;
}
```

```
  if ((temp=AM*iy) > RNMX)
  {
    return RNMX;
  }else
  {
    return temp;
  }
}
```

Annotated Bibliography

[1] Apache Software Foundation, "Apache License, Version 2.0," http://www.apache.org/licenses/LICENSE-2.0, 2004.

    This is my annotation. I should add in a description here.

[2] ——, "Apache POI - the Java API for Microsoft Documents," http://poi.apache.org, 2014.

    This is my annotation. I should add in a description here.

[3] H. M. Dee and D. C. Hogg, "Navigational strategies in behaviour modelling," *Artificial Intelligence*, vol. 173(2), pp. 329–342, 2009.

    This is my annotation. I should add in a description here.

[4] S. Duckworth, "A picture of a kitten at Hellifield Peel," http://www.geograph.org.uk/photo/640959, 2007, copyright Sylvia Duckworth and licensed for reuse under a Creative Commons Attribution-Share Alike 2.0 Generic Licence. Accessed August 2011.

    This is my annotation. I should add in a description here.

[5] M. Neal, J. Feyereisl, R. Rascunà, and X. Wang, "Don't touch me, I'm fine: Robot autonomy using an artificial innate immune system," in *Proceedings of the 5th International Conference on Artificial Immune Systems*. Springer, 2006, pp. 349–361.

    This paper...

[6] W. Press *et al.*, *Numerical recipes in C*. Cambridge University Press Cambridge, 1992, pp. 349–361.

    This is my annotation. I can add in comments that are in **bold** and *italics and then other content.*

[7] Various, "Fail blog," http://www.failblog.org/, Aug. 2011, accessed August 2011.

    This is my annotation. I should add in a description here.