Air Traffic Control

0608818

Abstract

This project simulates the movement of aircraft through a region of airspace surrounding an airfield. The task involves modelling the various aircraft, airfield and airspace, as well as their various properties and movements. In addition, it is hoped to model the interactions between various aircraft, such as collision avoidance and convoy-style flight. The project utilises the relational aspects of the DOSTE language in a scenario involving constant movement.

1 Introduction

The scenario of air traffic control has been largely ignored in previous projects using the Definitive Object State Transition Engine (DOSTE), this despite a plethora of investigation into similar situations, such as traffic control and railway lines. Thus, the aim of this project was to demonstrate the range of tasks that DOSTE (individually, or in conjunction with EDEN, the Evaluator for Definitive Notations) is able to fulfil.

2 Improvement Opportunities

2.1 Interpreter

During the course of this project, there was a multitude of challenges encountered. The largest issue encountered was the challenge of determining how the DOSTE language parses and executes DASM code: syntactically correct statements would often cause errors in execution, either with error messages or entirely silently. Due to the line numbers printed being consistently wrong, this was avoidable by repeated execution and testing of the program at every code edit in order to trace erroneous lines.

In addition, an issue was encountered where random numbers fail to be generated. When a parameter in a base class is set to a random variable, objects that inherit that class fail to cause the random variable to be regenerated. This resulted in multiple objects all having the same properties. The solution to this was to set the variable in question (the location of an aircraft) to a fixed value off screen. The x and y positions are controlled by a relation which resets the location to random when an aircraft goes off screen. Due to inherited classes of inherited classes recreating random numbers as expected, this solved the issue.

These two issues demonstrate areas that could be improved with regard to the DOSTE interpreter.

2.2 Language

With regard to the language, the syntax is robust but could benefit from some extensions; functions such as `"^"`, while not strictly necessary, make for much more readable code, and as of the time of writing appear to not exist. In addition, users would benefit from a better definition of how the language is interpreted: due to being based around relationships between objects, the order and frequency at which statements are executed is critical to the accuracy of the program. A flowchart or set of examples illustrating the execution order of, for instance `"is"` statements within `"="` statements could improve understanding of the language.

2.3 tkeden

Finally, the tkeden tool, while fully functional, could be vastly improved by standardisation of the interface: features such as `Ctrl+Enter` running the entered code, and the code box not losing focus when the mouse is moved away would improve the overall user experience and speed up code

development. In addition, the simple addition of a side window to list entities and their properties, in the same tree structure as they are stored, would save a large amount of time that is currently spent repeatedly typing in long queries.

3 Model Structure

As mentioned in section 1, the three basic object types used in this model are: the selected airspace, the airfield at its centre, and the aircraft.

The airspace has been defined as a window, with a default size of 900x760 pixels. By creating the main work pane as a separate entity to the rest of the interface, we are able to avoid errors and complications in transposing coordinates onto the correct screen area. In addition, this method of operation allows for easier detection of objects leaving the viewable area. Creating all other relevant objects as children of this screen completes the partitioning of different window areas.

The airfield is, at the time of writing, an object whose properties are inherited from `@prototypes image`, inheriting its position from the size of the airspace window but keeping a fixed size.

The most complex objects in this model are the aircraft, due to the interactions required with the airspace and airfield. However, while complex, they were an opportunity to use the full strength of the DOSTE language: all the functional code for an aircraft is stored in a single model object, and new aircraft can be created simply by creating a union with the base model. This allows for rapid iterations of testing, as code needs to be changed in only one location, as well as vast reductions in code length and complexity.

As the majority of an aircraft's properties were able to implemented as relations, this use of inheritance to share properties and related functions lead to such a simplification in code that an aircraft is able to be created, with all the movement and landing properties it needs, in only four lines of code:

```
@display children plane = (new union
(@display children planemodel)
visible is {@btncreate bool}
moving is {1}
);
```

The relationships between the various properties of the aircraft object are shown in Figure 2. Each of these dependencies is created using the `"is"` statement, so that they are continuously updated and all facets of the model are kept up to date.

4 Resulting Model

The resultant model fulfilled the majority of the project aims. Contrary to expectations, it was possible to implement every part of the project in DOSTE as, although work-arounds were occasionally required to avoid unimplemented features, it has all the functionality required for even complex projects.

The model highlights graphically the difficulty of keeping aircraft distanced in an area of airspace, as although able to move in all three dimensions, aircraft are not restricted to certain paths, and are unable to stop to avoid collisions. As finished, the model has seven aircraft at any one time in the airspace (this can be easily increased or decreased by the user). With regard to colour coding, red represents objects passing straight through the airspace, while blue represents aircraft that are wanting to land, and green represents aircraft in the process of landing. Aircraft entering the airspace do so with a random heading, and a user set chance of wanting to land (default: `@display percentlanding = 0.3`). The user also has access to the variable `@display maxspeed`, which controls the maximum speed that aircraft travelling directly through the airspace can achieve.

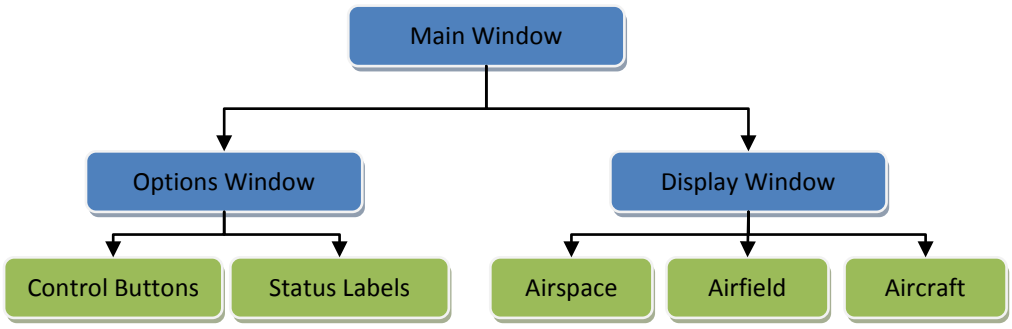

Figure 1 – Inheritance of object properties in the model

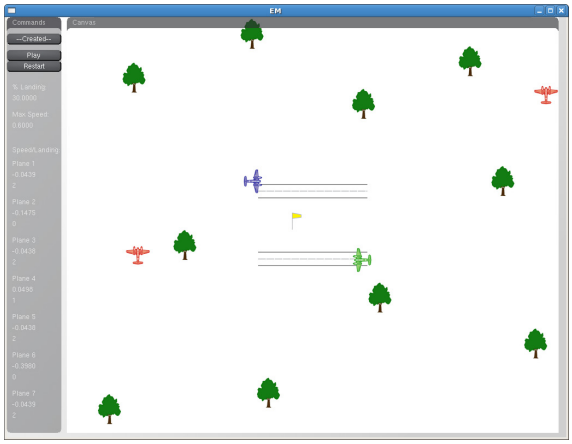

Figure 3 – Example Screenshot

5 – Further Work

While the majority of the features planned have been implemented, there still exists many improvements and extensions that, given time, could be added.

The first and foremost of these is collision avoidance between aircraft; this was attempted, but had to be abandoned due to problems in creating arrays in DOSTE, without which, the number of `if` statements needed would grow exponentially with the number of aircraft on screen. Given the definitions that control aircraft turns towards the airfield, this should be possible to implement quickly.

Figure 4 – Slight pilot error

Again, due to issues with arrays, the airfield's planned status properties had to be delayed. Originally planned were relations such as number of aircraft onsite and assigning aircraft a runway based on the best directional match.

Variables such as maximum speed, percentage landing and number of aircraft will be much more easily editable once sliders and input boxes are fully functional, as they can all be modified while the model is running.

Finally, a major subsection that is still in progress is control and variation of plane altitude. This was unfortunately simply a victim of time pressure.

6 – Conclusion

Given the improvements and additions suggested in section 5, such a model could be useful not only for education, but also for tasks such as getting an estimate of how many aircraft a region of airspace is able to handle. The main advantage of using EM versus a standard procedural language is the use of relationships between items, however the atomicity of updates to values simplifies complex relationships (such as those seen in Figure 2). As shown by this model, empirical modelling, given some maturing, can be an easy to grasp method of modelling real-world interactions, and benefits greatly from being applicable to almost every field. Despite this model having many possible improvements, it demonstrates the strengths of the technology.

Acknowledgements

Craig Rice & James McHugh, for ideas and feedback throughout the project.

References

http://img167.imageshack.us/i/airplaneallinone9tr.gif/
Aircraft image that formed the basis for the various plane icons

http://www.clipartguide.com/_pages/0808-0711-1417-5238.html
Tree image used for background

"Modelling Traffic Flow at traffic light controlled junctions", Unknown Author
http://www2.warwick.ac.uk/fac/sci/dcs/research/em/publications/web-em/01/trafficflow.pdf

"Airport Layout Planning", 0754074
http://www2.warwick.ac.uk/fac/sci/dcs/research/em/publications/web-em/04/airportlayoutplanning.pdf

The role of multimodal communication in cooperation: The cases of air traffic control, *Marie-Christine Bressolle, Bernard Pavard & Marcel Leroux,* "Multimodal Human-Computer Communication"

Method to Balance the Communication Among Multi-agents in Real Time Traffic Synchronization, *Li Weigang, Marcos Dib & Alba de Melo*, "Fuzzy Systems and Knowledge Discovery"

Figure 2 – Dependency between properties of the aircraft model. Property in blue, variables it depends on in red.