Emergency Egress Simulation: Investigating an Empirical Modelling Approach

0101187

Abstract

This paper examines the benefits of using an Empirical Modelling approach to simulating the movement of a collection of people out of an area during an emergency, using an original model developed with the `tkeden` tool. It is found that such a model can benefit from the use of EM, since the need to allow the user to experiment is met by a coherent, open environment which can cope well with change because of the use of dependency. Shortcomings are found in EM's dependency notation, and language extensions are given to address these issues.

1 Introduction

This paper will examine and evaluate the benefits of using an Empirical Modelling (referred to as "EM") approach to simulating the movement of people out of an area during an emergency. The paper is based on an original model called the Emergency Egress Simulation (EES), developed by the author using the `tkeden` tool.

In the first and second sections respectively, the positive and negative experiences of creating the model using EM are discussed, where possible making comparisons with alternative solutions for creating such a simulation. Finally, some suggestions are made as to how EM tools such as `tkeden` may be changed to improve their suitability for creating such models.

1.1 Background: The EES model

In this model, one can observe a set of people attempting to exit from a room. The room contains walls (which a person cannot move through or see over) and obstacles (which can be seen over but not moved through). The user of the model may add, move and remove walls, obstacles and people. The model gives statistics, such as the total time it took for every person to exit, and the number of people that went through each exit. A replay function allows the movement of people to be examined step by step.

It is helpful at this point to try to define and categorise the EES model so that it may be understood how the experiences given here may apply to other the creation of other models. It is:

- *Multi-agent*: with a population that is reasonably homogeneous.[1]
- *Data-intensive*: much data must be handled, modelling the position of walls, people, obstructions and exits.
- *Size-variable*: the size of the population, room area, and many other factors may increase or decrease.
- *Modelling complex behaviour*: Even to achieve simple overall behaviour, each person must be constructed using complicated individual behaviours.

2. EES and EM

2.1 Benefits of development in EM

2.1.1 Dependency models state

A core aspect of EM is the use of definitive notations, and this has several advantages when developing a model such as the EES. Since dependency models relationships between observables, its use within EM often removes the need for certain procedures in the EES– those that are there to maintain state. In traditional procedural programming, relationships between two variables must be maintained using procedural methods. However, when using procedures, the programmer must ensure that all of the necessary instructions are

[1] However individuals within the population can have differing properties, such as their speed of movement.

executed, in the correct order, to update the state of each variable. Dependency is a more coherent way of maintaining state, since observables have a well defined relationship to each other, and that relationship is automatically maintained by the system.

A good example of this is the way in which the two approaches are used to create graphical interfaces. In the EES model, the DoNaLD notation (Beynon et al, 1986) is used to create an interface driven almost entirely by dependency. (Some of the re-sizing of the interface must still be done by procedures, but this is because it requires the addition and removal of definitions).

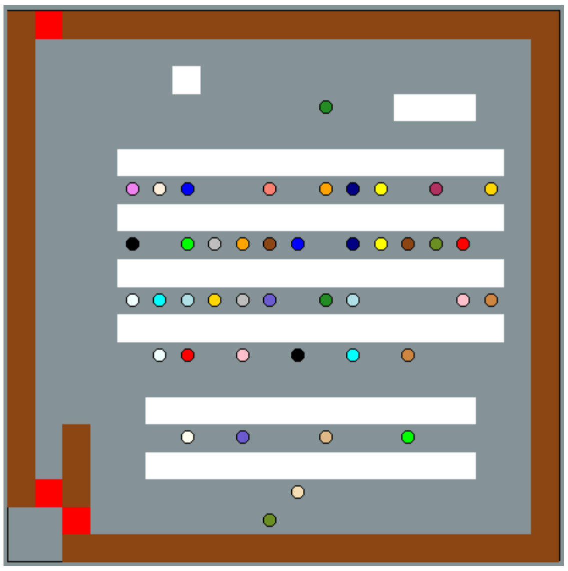
Figure 1: The animation section of the EES GUI

For instance, each person is drawn as a circle – where their position, size and colour is determined by a DoNaLD dependency similar to:

```
person1 = circle( origin + {person1_x! *
  gridwidth!,
  person1_y! * gridheight!}, personSize!)
```

This results in an interface which maintains itself almost automatically. If a person's position (represented by the EDEN observables `person1_x` and `person1_y`) changes, then the animation will update itself.

Contrast this with programming in C, for example, where one must create procedures that draw and remove the circle from the screen, and then whenever the person's attributes are changed the programmer must remember to call those procedures. When one considers that there is a circle for every person in the simulation, and a square for every grid position, a typical interface with a 20 cell wide and high grid may require around 450 shapes – a substantial number of elements to control.

On face value, the advantage of using dependency is that the modeller is relieved of the burden of updating the interface, lightening their workload and so quickening model development. However on a deeper level it also means that drawing interfaces is further abstracted away from the modeller, and that the interface represents a continuous view of state (as discussed above).

2.1.2 EES suits the empirical approach

Empirical modelling promotes experimentation as the basis for improving a model. Therefore the method of modelling is one of trial, review and improvement. To this end, it is often said that models made in EM can never be considered finished – because of there is always more experience which has not yet been accurately modelled.

EES is good example of model that is never finished: The amount of research devoted to Artificial Intelligence shows that modelling human behaviour is clearly very difficult. Indeed, AI is a topic that is particularly suited to EM as it is rather intangible, and so obtaining correct behaviour is achieved through trial and error. Also, in AI, one is generally only concerned with achieving the correct external behaviour, and not with the internal functionality which produces that behaviour. In developing the model, the behaviour of the people had to be continually adjusted and improved, gradually adding new features to cope with problems that new room designs exposed: To begin with, the person simply walked straight at the exit, but then it was given the ability to walk around obstructions if it had no-where else to go. To add realisim, a procedure was added to test if an exit could be seen. Finally the person was designed to prefer not going where it had been before.

The model is also suited to EM because it is by nature experimental: its goal is to allow a user to adjust the properties of the simulation to investigate the effects on the observables, such as total exit time, and then experiment with room designs to improve their results.

2.1.3 Openness

Openness is the critical quality in allowing the modeller or user to examine and experiment with the model. A simple example of this within the model is that a user can easily adjust, increase or decrease the placement of items within the environment, such as walls and people.

An example of more extreme modification is that such changes can be made whilst a simulation is running. For instance, one could simulate a person breaking their leg by slowing their speed drastically. A procedurally developed counterpart may well not cope with such a change.

2.2 Issues with development in EM

A problem that arose in the creation of this model was that often observables needed to be dependant on every element in a list. For instance, the `num_used_exit_1` observable must depend on each item in `person_positions` to calculate whether each person used exit 1:

```
num_used_exit_1 is
 (person_positions[1][5]==1) +
 (person_positions[2][5]==1) + ...
```

However the number of items in the `person_positions` list is subject to change as people are added and removed from the model, which poses problems: if more people are added than is included in the `num_used_exit_1` definition then the result will be incorrect; even worse, if people are removed, then the definition will contain references to items that are no longer in the `person_positions` list, and an error is generated. Techniques employed in the model as work-arounds to this problem are:

1. Adding triggered actions using `execute()` to regenerate the dependencies – this is inelegant, and also creates extra work for the modeller.
2. Lengthening the dependency to include terms for as many people as may be needed. Extra conditions must then be included every term to prevent an error:

```
num_used_exit_1 is
 (number_of_people >= 1 ?
 person_positions[1][5]==1 :0) +
 (number_of_people >= 2 ?
 person_positions[2][5]==1 :0) +...
```

2.3 Recommendations

Ward (2004) recognises a number of issues relating to the declaration of dependencies within lists, and addresses them by introducing a new notation. It is suggested that, in a similar way, the solution to the issues raised may lie in introducing a new notation which improves the set of operators that can be used in a definition. When attempting to create the definitions to control the visible attributes of a grid cell, each one had to be assigned to a definition similar to:

```
A_room1_grid_cellh1v1 is
 occupancelist[1][1]==-1 ? attr_wall_cell
 : (occupancelist[1][1]==-2 ?
 attr_exit_cell : (occupancelist[1][1]==-
 3 ? attr_obs_cell :
 attr_unoccupied_cell));
```

As it is extended, such an expression can become unwieldy. To alleviate the need to use nested conditional operators, it is suggested that a set of operators could perform a similar function to the '`switch`' keyword of many procedural languages. So the definition could be replaced with a syntax such as:

```
A_room1_grid_cellh1v1 is
 occupancelist[1][1]?? -1 ^
 attr_wall_cell: -2 ^ attr_exit_cell: -3
 ^ attr_obs_cell: attr_unoccupied_cell;
```

To address the problems expressed in 2.2, another possible extension to the language is to add syntax which will iterate over every item in a list in a given way. It is possible that the definition from that section could be replaced with:

```
num_used_exit_1 is
 +{person_positions[$][5]==1};
```

Where the braces enclose that action to be performed on each item, the `$` is the value that is iterated for the length of the list, and the symbol before the braces denotes the action with which to combine the results[2]. This syntax given is only to demonstrate the concepts, and would require refinement. A new notation could be written which would translate these new operators into functions in standard EDEN.

3. Summary

The EES model benefits well from the use of EM. Such models, where the goal is to allow the user to experiment, suit EM well. The use of dependency allows such models to cope well with unforeseen changes. EM encourages openness within the model, which is vital for experimentation. However, in some circumstances, EM does not cope well with automatic scaling of the model, and extensions to the EDEN dependency notation are suggested to deal with these issues.

References

W.M. Beynon, D. Angier, T. Bissell and S. Hunt. *DoNaLD: A Line-drawing System Based on Definitive Principles,* 1986

A. T. Ward. *Interaction with Meaningful State: Implementing Dependency on Digital Computers,* 2004

[2]This could include `+`, `-`, `*` and `>`,`<` for least and greatest