An Early Evaluation of Js-Eden through the Modelling of Eightball

0823443

Abstract

Model development through the use of observables, dependency and agency has been well supported by the tkeden tool for a number of years (Beynon, 2010). This paper is focussed on a new browser based environment, called Js-Eden, in order to establish its qualities including its support for empirical modelling methodologies. The method of investigating these qualities was to carry out a model development exercise from scratch, which is intended to simulate a break in a game of eightball. This paper includes a discussion on the development of the model, extensions made to Js-Eden over the course of the project and some of the issues in the tool that remain to be resolved.

1 Introduction

Empirical Modelling (EM) is about using observables, dependencies and agents to generate artefacts that support human thinking. It is concerned with dealing with the way in which understanding can emerge through experimentation (Beynon, 2006).

Js-Eden was a tool initially developed by Timothy Monks for his MSc dissertation (Monks, 2011). It was implemented using the scripting language, JavaScript. The new EM tool allows interactive models to be developed, updated and displayed in the browser, which makes it very accessible for users. However, as the tool is still new, there are not yet many models available for it. This also means that Js-Eden's functionality has not yet been fully tested.

2 Modelling Eightball

For my assignment I decided to take some of the ideas presented in (Yung, 1996) and develop a model using Js-Eden. I used the tool to model an initial break in a game of eightball.

This game consists of seven red, seven yellow, one black and one white ball. A cue stick is used to hit the white (cue) ball into the other balls on the table with the intention of scattering and potentially pocketing them. The overall layout of the eightball table and the initial set up is shown in Figure 1, with the Baulk Line visible in white along the lower quarter of the table.

My model sets the table in accordance with the rules laid out by the World Eightball Pool Federation (WEPF) (Federation, 1996). Although game rules allow the cue ball to be placed anywhere behind the Baulk line, my model places the ball at the centre of this line by default.

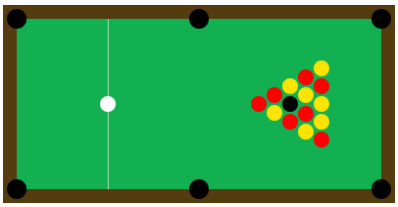

Figure 1: Table layout for the start of a game of Eightball

In my model, one end of the cue stick is always pointing directly towards the middle of the cue ball. The other end can be placed by clicking the mouse anywhere on the canvas element. The direction and length of the cue will dictate how much force goes into hitting the cue ball, and hence what velocity it is given. In accordance with the WEPF rules (Federation, 1996), the cue ball can be hit in any direction.

The aim of the model was to simulate an initial break shot. When the user has placed the cue, pressing the 'Start' button will begin the break simulation. The cue ball will move around the table and collide with cushions and other balls. In the latter case, some of its velocity will be transferred, sending other balls moving round the table. Balls that fall into the pockets will be removed from the table and the simulation will continue until all balls have come to a stop. As collisions have been modelled as being perfectly elastic, friction was used in the model to cause the balls to lose velocity over time.

As the focus of this paper is not to discuss my model in large detail, a second document will be provided to describe how it works.

2.1 Model Limitations

One of the issues I faced with my model was how balls in the triangle would detect collisions occurring between each other. This was due to the initial spacing between the balls, which although looked more realistic, meant they were actually overlapping slightly. This caused issues, as the model would try to respond and resolve the ball collisions. My solution was simple and was to increase the spacing. Although this decreases the accuracy of the model (balls should not be spaced out in real life), it meant that fewer calculations were required compared to getting the model to ignore collisions before the cue ball breaks the triangle. I felt this trade off between accuracy and performance was justified. Performance is discussed in more detail in the next section.

An issue that occurs with all models involving collision detection is how a moving object may move too far in an update to its position and end up overlapping another object. Though a collision is found, the resolution may not be great enough to get rid of the overlap in the next update, causing another collision to be detected. On top of this, resolving a collision when overlap is occurring means the resolution can be slightly off, leading to unrealistic velocities after impact. My solution to these issues was to limit the x and y velocities of the balls to a range of -200 to 200 and then divide them by one-hundred before using them to update positions. This meant that balls moved sufficiently small amounts to help prevent overlap situations, but also means the balls move quite slowly. Unfortunately, the model does not cope well if the user re-locates balls in such a way that they overlap and they will not bounce away from each other.

The model is also limited in that it requires the user to place the cue and press the 'Start' button before the model can be run. Only when these conditions are met will the `moveBalls()` procedure be called. Alternatively, the user must enter the following:

```
initialHitTaken = true;
moveBalls(poolBalls);
```

Optionally, the user can set the velocity of one or more balls using (for example, for the cue ball):

```
poolBalls[1].velocityX = 10;
poolBalls[1].velocityY = 20;
```

2.2 Model Improvements

As seen, a number of issues still exist within my model. One of these is the model's reliance upon certain observables to be set before the 'break' can be made. One possible way of getting round this is to instead use triggered actions that are fired whenever a ball is updated, which will check whether that update has caused it to collide with anything and update its position accordingly. However, this may be costly performance wise. A second option could be to try and use dependencies to determine whether two balls are colliding, but this may overcomplicate the model.

Other improvements could be to figure out how best to deal with overlapping balls, whether caused by a user moving balls so that the overlap, or by balls moving too fast and collisions not being detected quick enough. One solution may be to predict when collisions will occur and use

this calculated moment in time to reverse balls (and hence move them backwards) to just before the collision. The collision resolution can then be calculate from these locations.

3 Experiences with Js-Eden

In this section of the report I will discuss the advantages and disadvantages of Js-Eden over other EM tools and how I contributed to improving the tool.

3.1 Improvements over previous tools

One major advantage of Js-Eden over other tools such as tkeden is its ability to allow users to add new functionality to the tool. This is because the tool's source code is all currently being hosted as a repository on GitHub. From here, users can easily access an up-to-date version of the tool and make a local copy on their computer. If they wish, they can carry out development on Js-Eden to improve it or fix any bugs. If these changes are useful, they can then be merged back into the main repository on GitHub (via a pull request). The fact that the tool is open source provides great benefits, as more people are able to help with its development and hence speed up the process of improving the tool. It also greatly increases the presence of EM on the internet, which may help make people aware of the research area.

Another advantage is how users can add features such as new libraries and new Object types to the tool. These are then immediately available to interact with. This contrasts a lot with tkeden, which requires that any new functionality must be added and the tool re-compiled before the features can be used. This quality of Js-Eden seems more in the spirit of EM, which values the ability to update models without reloading them.

An undocumented extension to the EDEN notation that Js-Eden offers is the ability to use object types. Objects allow users to easily make multiple versions of the same type. On top of this, the different attributes of an object can be assigned names to make them easier to remember and use. For example, my model makes use of a 'Ball' object, which stores a ball's x and y location and velocities. These attributes can be accessed using, for example, '`ballA.x`', as opposed to '`ballB[2]`', where '`ballB`' is represented as a list as used in tkeden. The addition of objects in this new tool means that users can implement their own objects for their models and use them straight away, without having to compile them as a new feature of the tool.

3.2 Issues with Js-Eden

The main issue that arose when the eightball model was under development was the performance of Js-Eden. When using EDEN code to describe intensive computations, the performance of my model suffered greatly. This was partially due to how the model was implemented, which meant that the distance between every two balls and every ball and every pocket had to be calculated to determine whether they were colliding.

As there was little that could be done to change the performance of Js-Eden within the Web EM 8 time scale, I decided to alter my initial plans for my submission. Instead, I created three models, each with a different number of balls. The initial model contained only two balls - the cue ball and the foremost red ball. This version of the eightball model, though very unrepresentative of the actual game, shows the performance that was achievable when only a very small number of balls had to be dealt with.

The second model contained all sixteen balls but the performance of the model was massively decreased. It was so slow that it made it hard to follow what balls were doing, as the time between updates in ball locations was great enough that it was easy to forget exactly what direction balls were meant to be moving in. Therefore, despite being an accurate model for representing an initial break in a game of eightball, its performance meant that monitoring how observables changed over time was hard.

The last version of the model reduced the number balls back to a total of seven and included only the first three rows of the triangle. The performance was better than that of the second model, but still suffered when compared to the first. Ball movement speed was at least a half of that found in the first model but the performance in this model did mean that there was a good balance between the accuracy and usability of the model. The performance in this model made it easier to observe how the balls moved around the environment and interacted with each other. I therefore decided that, although all three versions of the model will be submitted, this version would be my main model.

One other issue that I came across was to do with reference types. The issue was brought up in (Monks, 2011). In EDEN, the value of an observable changes if an agent has changed it or if it was dependent on another observable, that has also been updated. However, due to how Js-Eden is implemented, this latter update can sometimes fail. An example of the bug is illustrated in the code below:

```
ballA = Ball(25, 34);
ballList = [ballA];
ballList[1].x = 25;
```

Although in this example `ballList[1]` was updated, the implementation issue in Js-Eden meant that `ballA` was also updated, which it shouldn't have been. Unfortunately it was not possible for me to look into fixing this issue during the course of the project. Please note that this example code uses a simplified version of the Ball object and considers only the x and y location values.

3.3 Extensions to Js-Eden

An issue that I came across very early on in the assignment was how drawing a Circle object would cause any Text objects drawn after the circle to become distorted. This was due to the value for stroke being global. All but the circle function required a stroke value of 1, but circle increased the value and it didn't get reset anywhere. My first fix was therefore to set the stroke width in all of the objects that could be drawn, so they did not have to rely on the global setting not changing.

A second improvement I made to the tool was to add functions to each of the Objects, which improved how they were represented in the 'Observables' panel of the site. Previously the 'value' for the objects was being displayed as '`[Object object]`', something that was not very useful for users, as the field values for that object were not visible. The functions I added meant that each object's field values would be represented as a list, allowing users to distinguish between similar objects and easily view the different field values at a glance. This functionality was made possible by wrapping a function around JQuery's '`toString()`' method. With this addition, observables went from being displayed as, for example:

```
rect = [Object, object]
```

To

```
rect = Rectangle(x1, y1, x2, y2, colour)
```

With the appropriate values substituted in.

Although maths functions did already exist within the Js-Eden libraries, there were some that were missing. These included an atan2 function and a function that returned the value of Pi. These functions were needed in a version of my model but I no longer use them. However, they may be useful for other people. The Js-Eden functions wrap around JQuery's atan2 method and its inbuilt value for Pi and return the calculated values accordingly. For example, Pi:

```
func PI {
  return ${{ Math.PI; }}$;
}
piValue = PI();
```

Arguably the biggest feature I added to Js-Eden was the introduction of four new observables

that would both track the mouse's x and y co-ordinates as it moved over the canvas and also the x and y co-ordinates of the last place the mouse was clicked on the canvas. In both cases, the top left corner of the canvas was represented as the point (0, 0), and the bottom right was equal to the width and height of the canvas. They made use of the '`mouseMoved`' and '`mouseClicked`' event handlers in JQuery.

4 Conclusions

Throughout the duration of this assignment I have used Js-Eden extensively. As this report shows, the tool is still new and lacks features that users may find useful when developing models. I have discussed some of the ways the tool can be improved and some of these may make good projects for students taking part in the next Web EM assignment.

Perhaps the most important issue raised by my project was the performance of the tool, which hindered my ability to create a realistic model. Profiling my model to determine exactly which parts of it hinder performance could be a useful exercise for the future.

The problems arising with shared references to list and object data should also be solved to ensure that Js-Eden remains consistent with the conceptual framework established by tkeden.

However, despite its issues, Js-Eden has proved that it provides many of the features available in the older tools. On top of this, the increased accessibility and open source nature of the tool means that it will likely play a big part in encouraging people to embrace the teachings of EM and may be a stepping stone in EM's future development.

5 Acknowledgements

I would like to thank Dr Meurig Beynon for his support and interest in my project. I would also like to thank Nick Pope for being so kind as to accept my improvements to Js-Eden and merging them with the main release of the tool. Finally, I would like to give a special thank you to Tim Monks, who provided continuing encouragement and patiently helped me understand some of the more difficult concepts of EM and Js-Eden when I got a bit confused.

6 References

Beynon, M., 2006. *Principles and Tools*. [Online] Available at: http://www2.warwick.ac.uk/fac/sci/dcs/research/em/intro/principles/ [Accessed 21 December 2011].

Beynon, M., 2006. *What is Empirical Modelling?* [Online] Available at: http://www2.warwick.ac.uk/fac/sci/dcs/research/em/intro/whatisem/ [Accessed 21 December 2011].

Beynon, M., 2010. *EDEN - the Engine for DEfinitive Notations*. [Online] Available at: http://www2.warwick.ac.uk/fac/sci/dcs/research/em/software/eden/ [Accessed 20 January 2012].

Federation, W.E.P., 1996. *World Eight Ball Pool Playing Rules*. [Online] Available at: http://www.wepf.org/playrules.php?option=1 [Accessed 22 December 2011].

Monks, T., 2011. *A Definitive System For The Browser*. [Online] Available at: http://www2.warwick.ac.uk/fac/sci/dcs/research/em/publications/mscprojects/timmonks/trmonks_dissertation_report.pdf [Accessed 20 January 2012].

Yung, S., 1996. *Billiards*. [Online] Available at: http://empublic.dcs.warwick.ac.uk/projects/billiardsYung1996/ [Accessed 15 December 2011].