Empirical modelling for the bullet trajectory analysis

0953518

Abstract

The analysis of bullet trajectory and the simulation of generating bullet path is a complex physical phenomenon with a series of external force acting on the bullet in the entire flying process from muzzle to the target. That makes the simulation and analysis of the bullet trajectory an extremely difficult project for these Object Oriented programming approaches, since this process of generating bullet path need real time update on a series parameter of the trajectory. Then empirical modelling becomes the perfect method to perform this task by utilising the basic principles of EM which are also the biggest advantage of it over other programming approaches. This paper illustrates some advantages and disadvantages of EM by showing how the bullet trajectory analysis artefact using some remarkable characteristics of EM.

1 Introduction

The primarily focus of this paper is to build the understanding of principles of EM and how that can be practically used in applications. The attempt of putting EM into application is accomplished by developing an EM application of bullet trajectory analysis.

The bullet trajectory analysis involves some physical formula to calculate the impact of external forces onto the bullet. One of the most outstanding merits of EM is that it allows programmer build relationship among the variables, called “dependency”. This is the reason EM is a perfect tool to perform the simulation of some real world scenarios.

This model can provide the whole process of shooting a target, from choosing rifle, aiming, to setting up the speed and direction of wind, lastly, the programme will display the result of shot by showing the bullet deflection graphically.

In the process of building this model,

2 Bullet trajectory analysis

2.1 Effects of weather

There are two main types of external forces acting on the bullet since it flied out from muzzle. The first is gravitational force; the other is aerodynamics [1].
The aerodynamics can be reckoned as a series of forces coming from several aspects, and they are: drag, lift, side forces, Magnus force, spin damping force, pitch damping force, and Magnus cross force [1]. Among these aspects, the most vital one is drag. The summation of the impact from other aspects can be ignored in the calculation of the bullet trajectory.

There is another critical factor which significantly influencing the path of bullet, that is wind. For the simplicity of the mathematical calculation, the overall effect of wind should be deemed as a combination of three components from three axes of 3D coordinate system [1].

Headwind or tailwind will cause the variation of the drag posing onto the bullet. A crosswind will let the direction of the bullet path largely changed from the direction of the extended bore line. As for the vertical wind, it can turn the bullet upward or downward following the direction of the wind. The contribution in the combined wind force from headwind or tailwind is neglectable since it will not cause any of the deflection from the extended bore line. Hence in this paper, the calculation will not include the effect from headwind or tailwind.

2.2 The Ballistic Coefficient

There is a fairly important concept in ballistic trajectory analysis need to be addressed, the BC (Ballistic Coefficient). In an article of William T. McDonald, this term addressed as “The ballistic coefficient of a bullet is a measure of how well it retains velocity as it travels downrange and how well it “bucks” the wind”[2]. Every single type of bullet has different shape and drag index. The BC of the bullet indicates how well the bullet survives the impact of crosswind. The bigger the BC of the bullet is, the smaller the impact bullet will experience from the crosswind.

3 Dependency & Agency

Dependency is the most outstanding feature of this programming approach and is the only vital function a programming language should provide to perform the simulation of some real world scenarios.

There are many existing EM application can proof the fact that EM principles and tools is a perfect tool for modelling [3]. One of the merits the programmers will get from deploying EM is the dependency among observables makes producing GUI far more easier than other traditional programming approaches.

To illustrate how dependencies facilitate the implementation of the dynamic GUI, the following section will explain how that is realised in this model. In this sniper artefact the result of the shot, namely, the deflection of the bullet trajectory displayed in the window shown below:

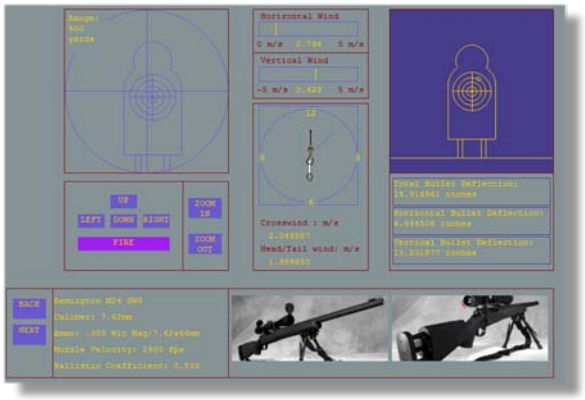

Figure 1: The GUI of this sniper model

When the user finish the setting of speed and direction of both horizontal and vertical wind, and aiming the target, press the fire button, the system will display the deflection. In EM, the current state of an artefact is represented by the series of definition. Since the observables have dependencies, any changing of the observable will directly affect the value of all dependent observables. Once the fire button has been hit, the "string" attribute in the window component in SCOUT file will be changed accordingly and dynamically to the weather details set by the shooter. Other traditional languages will display the variable for once, and if some "listeners" are used, the variable will be dynamically changed only under the condition that the "listeners" are running. If there are hundreds of observables, the CPU resource will be largely wasted on running numerous "listener". Writing these method for observe the change of variables is also a waste of man power.

Agency is a perfect format of data structure for real world simulation ---- this is under my understanding that agent should be reckon as a type of data structure. None of the existing data structure can provide such a high level similarity between the computer model and the connection among objects in the real world any better than agent does in EM environment. Take the model in this paper for instance, all the external forces have been impacting the trajectory of the bullet since it left the barrel. If we set all the forces acted on the bullet as an array, it is a certainty that this will be a disaster for the programmer who using the language without dependency and agency.

4 Execution Strategy

Some features of execution strategy in EDEN are used in this bullet trajectory model.

One of the Execution Strategy is that all observables will be updated before the script execution. All the triggered actions are queued in a waiting list. The actions will be performed after all the values have been updated. This is an extremely strong advantage for a modelling tool. This will save programmers' manual work for scheduling the sequence of action and value updating in a large scale. In this model, the bullet deflection will not be displayed until the rifle parameter, bullet parameter and wind details are updated by the interpreter.

EDEN also has an advantage of scheduling the evaluations. It costs the interpreter more time on arranging the order of evaluation. This is accomplished by utilising a breath-first scheme rather than a depth-first scheme. However, this brings a merit to EDEN. The evaluation will be only executed once in each phase for the observables and triggered actions.

If other OO languages are used for performing the functionality of this model, one thing for sure, that will be a disaster for the programmers. They have no choice but to build a lot of methods and classes for scheduling the sequence of execution for all the methods and variable updating. In the model like this shooting scenario, there will be uncountable variables serve as parameters of bullet path. Arrangement of the updating execution sequence must be an impossible mission.

The action execution management sub-system of EM is elaborately designed. The block diagram below shows the configuration of it. The reason of showing that is to illustrate a feature of EM: the action specification will be invoked whenever the values of observables are changed. What need to be

noticed is that the term “changed” including the overwritten of observables, even with the same value. This certainly is a tremendously useful feature to have. Although this function can be implemented by using java or C++, build two separate functions for both the overwritten with previously value and brand new value. However, the time and resource cost to observe the overwriting actions apparently will not be on the same level with EM.

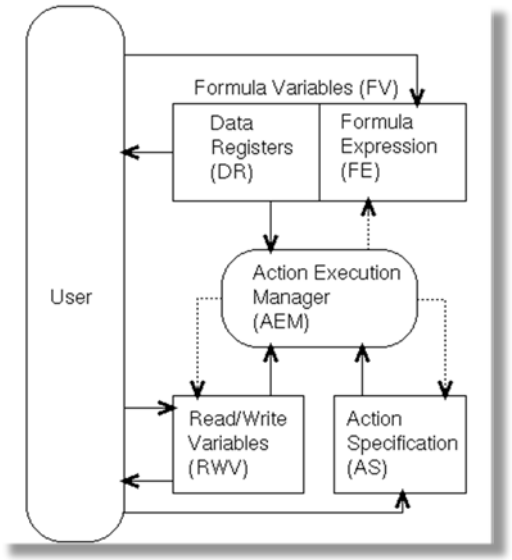

Figure 2: Block Diagram of the Action Execution Management Sub-system

5 Some downsides of EM

In the process of developing this model, some drawbacks showed themselves.

Firstly, in the SCOUT notation, the only available shape of boxes which are used to positioning the Donald viewports is rectangular. Moreover, the boxes can only be solid. If the programmer wants to put some viewports in other customized boxes or let the boxes being transparent in order to show the lower layer boxes up, SCOUT notation will have nothing to do with that. My suggestion is that to open the definition of box and allowing the redefine of the box as a data type.

Secondly, the y coordinate in Donald is opposite with SCOUT. In the definition of box in SCOUT, the y coordinate is counted from the top left of the screen, while the Donald coordinate counted from bottom left. Most importantly, this is not illustrated in the reference of both notations.

Thirdly, based on the fact that EM is not a strong typed language, which means that type of observable is not necessary in the declaration. Actually, there is no such a method served as declaration in EM. This brings the situation like two sides of the coin. In one hand, the users do not have to declare the type of some observables when they have not been determined. In the other hand, the programmers will not have the convenience of knowing exactly properties each data type has.

There is one thing that EM missed, a Graphical tool for designing GUI using Donald and SCOUT. Just like HTML, JSP and other notation used to draw GUI, Donald and SCOUT should have a high integrated platform which could largely save the manual work of coding the buttons and text boxes.

6 Further work

This model could also be improved by using DOSTE. One extremely useful characteristic of DOSTE is that programmer can make the graphically display of observables be an animation. Consequently, people will be able to observe the modelling process more explicitly and the implication behind the model will be easier revealed

Acknowledgements

I would like to thank Meurig Beynon for his valuable advice about the topic selection and the constructive guidance on the designing of this EM model. It is a pleasure working with Dr. Beynon and getting some idea of this remarkable programming language.

References

[1] HEADQUARTERS DEPARTMENT OF THE ARMY Washington, DC. SNIPER TRAINING, *FIELD MANUAL No. 23-10, Chapter 3, section III.* 17 August 1994.

[2] William T. McDonald, Ted C. Almgren. THE BALLISTIC COEFFICIENT. December, 2008

[3] Meurig Beynon, Computer Science, The University of Warwick. *Visualisation using Empirical Modelling principles and tools*, AHRC ICT Methods Network Expert Workshop "From Abstract Data Mapping to 3D Photorealism: Understanding Emerging Intersections in Visualisation Practices and Techniques", June 19th 2007, Birmingham UK.