SCRIPTING AUTOMATION FOR A GRAPHICAL SCENARIO AUTHORING SYSTEM

David A Heitbrink
National Advanced Driving Simulator, University of Iowa
Iowa City, Iowa

ABSTRACT

Scenario authoring at the National Advanced Driving Simulator (NADS) uses an interactive graphical tool that permits non-technical people to create scenarios without requiring any programming experience. This tool, the Interactive Scenario Authoring Tool (ISAT), has been enhanced by the addition of a scripting system for automation and facilitating authoring tasks. ISAT scripting (ISC) goals are: 1.Reduce the time to add new features to ISAT through automation, 2. Enable experienced programmers to build standardized scenario event components that other users can incorporate into other scenarios, and 3. Automate highly repetitive scenarios that contain similar events. Authoring highly reparative scenarios using a purely graphical interface can be a time consuming, monotonous, and potentially error prone process.

Inspired by LOGO, the ISC language is designed to automate placement and manipulation of scenario objects (including visible scenario objects, and control objects) with simple direction commands that navigate a road network. An ISC script can ask the user questions, such as asking how far ahead from a point in the environment to place or select an object. Once an ISC script is created, the user can drop the script into the scenario and the script will start executing at the point where the users drop it into the scenario. Overall ISC has been successful providing automation for a series of manual tasks, reducing repetitive task performed by scenario authors, reducing scenario creation time, and decreasing authoring errors.

INTRODUCTION

Scenario authoring at the National Advanced Driving Simulator (NADS) uses an interactive graphical, the Interactive Scenario Authoring Tool or ISAT, that permits non-computer professionals to create scenarios with minimal programming expertise. ISAT requires no programing on the part of the scenario author.

Unfortunately, one side effect of this fully graphic development environment is that highly repetitive scenarios can become more difficult to author. For long series of highly repetitive events like object detection, the scenario is likely to require a series of triggers with slight changes between each trigger, and slightly different object placement. The user would most likely navigate through multiple dialogs and change values in multiple places, and for an object detection/recognition scenarios frequently involve more than 100 objects. Without automation, scenario development can become error–prone.

Given these short comings, ISAT has been enhanced by the addition of a scripting system for automation and facilitating authoring tasks. ISAT scripting (ISC) goals are: 1.Reduce the time to add new features to ISAT through automation, 2. Enable experienced programmers to build standardized scenario event components that other users can incorporate into other scenarios, and 3. Automate highly repetitive scenarios that contain similar events. Authoring highly repetitive scenarios using a purely graphical interface can be a time-consuming, monotonous, and potentially error- prone process.

ISC takes its inspiration from the programing language LOGO [1], a simple programing language that was designed for children. The ISC language automates the placement and manipulation of scenario objects (including visible scenario objects, and control objects) with simple direction commands that navigate a road network. An ISC script can ask the user questions such as: “How far ahead from a point in the environment should an object be placed?”. Once an ISC script is created, the user can drop the script into the scenario and the script will start executing at the point where the users drop it into the scenario. Overall ISC has been successful providing automation for a series of manual tasks, reducing repetitive tasks performed by scenario authors, reducing scenario creation time and decreasing authoring errors.

BACKGROUND

ISAT or Interactive Scenario Authoring Tool is a fully interactive tool scenario authoring tool, which allows graphical authoring of driving simulator scenarios as well as rehearsal and review of simulator runs. ISAT has been designed to use graphical menus, visual feedback, and guided

Presented at the **IMAGE 2011** Conference
Scottsdale, Arizona – June 2011

input dialog boxes to increase usability. ISAT is designed to be a fully graphical editor that does not require any level of programing by the scenario author at any time.

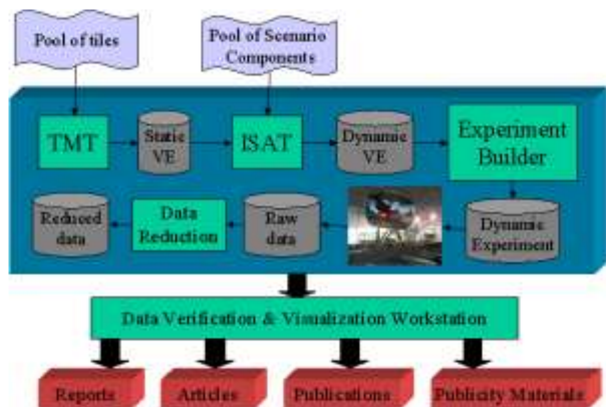

Figure 1. Overview of scenario development integrated into the experimental design.

ISAT creates a text based scenario file, and relies on a pre-built logical database. These logical databases are built using the Tile Mosaic Tool, which stitches together pre-built segments of the virtual world and outputs the logical data base, among other things. This logical database contains the logical road network system. This progression can be seen in Figure 1. The logical data base consists of "roads" and "intersections"; a road is a road segment and made up of a number of lanes, each lane then has a direction of travel. As shown in Figure 2, intersections are used to connect one road to another road, each intersection has number corridors that connect one lane in road to one lane from another road, as can be seen in Figure 2.

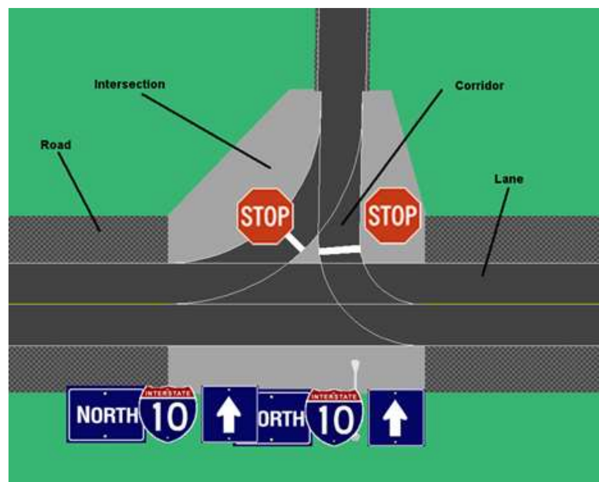

Figure 2. An example of a logical road network.

The role of ISAT is to create the scenario. The scenario consist of a collection of settings to static objects in the logical database such as speed limit signs, control units such as "road pad triggers", and dynamic elements such as AI controlled cars. ISAT also has an extensive feature set for reviewing collected data. Most of the control in scenario is provided by the way of "Triggers". A trigger resembles a logical if-then statement. Each trigger has a predicate condition and set of actions. The predicate is the logical "if" part; the actions make up the "then" part. If the predicate is true, then the trigger performs the "actions". One of the most basic triggers is the road pad trigger (Figure 3). When the target element such as a car is over the road pad, then the trigger fires its actions, the appearance of a vehicle on the adjacent road.

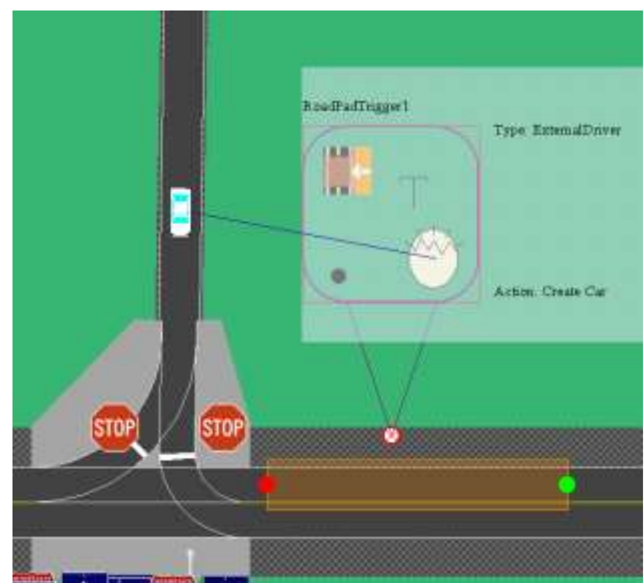

Figure 3. An example of a road pad trigger

Numerous scenario elements have "road pads", a road pad is describes a "path". A path is a set of road segments and corridors that describe a route through the road network.

ISAT SCRIPTING LANGUAGE

LOGO is a language that was designed in Cambridge MA, in the last 1960's by Daniel G. Bobrow, Wally Feurzeig, Seymour Papert, and Cynthia Solomon, as a simple programing language that children could use. ISC borrows the concept of LOGO's turtle to navigate through the road network to place objects. The turtle, is a graphic system that instructed a "turtle" (originally this was an actual robot with a pen) to raise/lower its pen to paper and to move/and or rotate. All direction was relative to the locations of the turtle. This provided a simple system that children could grasp, and

use to create relatively complex drawling.

ISC borrows from the concept of the turtle with the "Position" variable. The position variable describes a location in the virtual road network. The "anchor" is constant that represents where the user dropped the script down. From this "anchor" point the author can navigate through the road network using a simple set of commands, and to use these position variables to place scenario elements.

The Position variable type represents a location on the virtual road network. The "Anchor" variable is the initial location where the user drops the script into the scenario. The position variable consist of three parts, it has a "path" that can be used as a roadpad, it has a road position (has road/intersection name, a lane/corridor and an offset, and a XYZ position. After each operation that changes the position, the XYZ position is remapped from the logic database. The path is a set of directions that specify a start and end location in the logical road network. As the position is moved locations are added to the end of the path. The position variable has the following operations:

GoForward

Go forward a distance (can be negative for traveling backwards)

TurnLeft, TurnRight, GoStraight

These functions advance the position through the next intersection either making a left turn right turn or going straight

HalfTurnLeft, HalfTurnRight

These functions advance to the midpoint through the next turn from the current position

PathOn, PathOff

Like the pen up and pen down function in LOGO these functions turn off and turn on the "path". A path is a representation of a route through the road network. When PathOff is set, as the position is moved, its location is not added onto the path, when PathOn is set, as the position is moved, its changes are added. When PathOn is set it resets the Path. The path is set off by default.

SetOffset

This function set an offset from the center of the lane. Negative values to the left of the center line, positive to the right.

ChangeLaneRight, ChangeLaneLeft

These functions change the current lane the RoadPos is in either to the right or to the left of the current position, without changing direction of travel.

Block Type

The Block variable represents a "block" of text. The %%% represents the start and end of the block of text, the "Block" variable is used to load a scenario element. Once a block is assigned to a scenario element variable type, the scenario element parses the block and treats as if it was loaded from a scenario file. Various text replace operation can be done on the Block, such a text replace, and this can be used to create unique names for every scenario element created from the "block". The block has a header, and an end statement. Each line between the beginning and end will be parsed as a "key" then a value. Each of these lines must contain at least two separate values. The contents of a block variable can be copied directly from the scenario file. The following is an example of a "Block" declaration, where the contents of a road pad trigger have been copied from a scenario (all the text between the "%%%" on the first and last line are directly from the scenario file):

```
Block RoadPadTrigBlock %%%
HCSM RoadPadTrigger
  Position -4.5440845E+004 -
6.3007273E+004 3.0000000E+001
  DrawPosition -4.5440845E+004 -
6.2932273E+004 3.0000000E+001
  ByTypeSet "ExternalDriver"
  FireDelFrames 0
  Lifetime 0.0000000E+000
  Name "RoadPadTriggerXXX"
  OneShot 1
  SeqAct 0
  ExtInfo "59.904864:102.310000"
  Path "R:r3c_-48840_-
67320:1[108.06:5.75]"
&&&&End&&&&
%%%
```

Scenario Elements

ISC also has a large number of scenario element variables (ADO, dDDO, DDO, Static, TimeTrigger, ExpressionTrigger, RoadPadTrigger, TTATrigger) that represent scenario elements. All of these elements support a SetBlock operation that takes in a block variable and parses it, creating the scenario element. The position and roadpad/path of the object can be set through set position and set road pad. These functions both take a position variable as a parameter. Also, the scenario elements each

support a “Clone” function. The Clone function creates a new instance of the object with all the setting of the original object. Once the object is cloned it is assumed that any modifications of the object are complete and the cloned instance is no longer modifiable. In the bellow example we can create a simple line of 20 barrels:

```
Static Barrel
Position posRp
Barrel.SetBlock(BarrelBlock)
posRp = Anchor
Barrel.RoadPos = posRp
Value Dist
Value Offset
Repeat 19
      Barrel.Clone
      Dist = 20 + Rand() * 5
      Offset = 6 + Rand()
      posRp.GoForward(Dist)
      Barrel.RoadPos = posRp
End
```

Note that this loop is repeated only 19 times, as the original instance of the barrel is never destroyed and will be last in a line of 20 barrels. Each scenario element variable supports a SetKey operation that sets key in the variables parse block. For example, the following line can be used to set the “Lifetime” setting for a variable:

```
MyCar.SetKey(“Lifetime”,
1.0000000E+0010)
```

The above line would instruct “MyCar” to remain active for 100 seconds after it was created. The SetKey function allows the programmer to change properties of objects without any kind of language support for the property.

Action Type

Triggers function as logical if then statements. The actions function as the then part. The action type represents these actions. The actions type allows for the manipulation of triggers actions, such as setting there type, and doing things such attaching ADOs to a create actions.

Value Type

The Value type represents either a number value or a text string depending on the context of use. The value variable supports mathematical operations, and a few functions such as Rand (random number generator). For instance in the above example where 20 barrels were created, the variable Dist is set to a random value between 20 and 25, and Offset between 6 to 7. This allows us to create a line of barrels that looks like it was created by a construction crew.

Ask and Select

ISC also support a number of user interaction functions. An ISC can instruct the user to select a scenario element, or a location. The Select statement, if given a message, will display a text box and wait for the user to select an object. The following example is for a script that is embedded into ISAT, and is executed when the user selects a context menu item “Place Object At”:

```
.Internal
.Name Push
ScenElem myElem
Position pos
pos = Anchor
Value val
Ask(val,\"How Far Forward do you want
to set the other Object\")
pos.GoForward(val, “Please Select the
Other Object Now”)
Select(myElem)
myElem.RoadPos = pos
.End
```

The above will select a scenario object and move it to the exact distance from the object the context menu is selected from.

RESULTS

ISC scripting has been used in various ways. ISC scripts can be embedded into ISAT context menus. When the user right clicks on a scenario element and selects a menu item, a script can be executed. This reduces programming labor to implement said feature. For example; the “Make Next Right Turn” menu option executes a simple script that modifies the vehicles path to make the next right hand turn.

At NADS, ISC has helped reduce the time to develop certain scenarios, and has enabled us to better utilize student labor, as a script for an event that is repeated frequently in an event can be placed repeatedly will little prior instruction. Also, ISC has given us an increased level of flexibility. At NADS, we have had instances where entire sequences of events where created with a single ISC script. When a sudden change to the fundamental way the event was timed was required, a small change to the script could be made, and the scenarios could quickly be recreated.

Another use of ISC was to help facilitate the process of scenario standardization. Omar Ahmad [1] in 2005

described a process for creating scenario an event definition based on anchor points and locating scenario elements relative to said anchor points. ISC is partially an attempt at starting to facilitate said process. ISC can be used to describe a higher level implementation of an event, where the locations of scenario elements are relative to given points. The refinement of ISC is a continuing effort.

Future enhancements include the validation of the fitness of a location to match a script. For example a script requires the intersection of two 4 lane roads, “validate” statements could be added to the script specifying said requirement. If the location the user has selected fails to meet this requirement a messages could be displayed instructing the user why they cannot use a particular script for a certain area. Another future enhancement is the interaction with traffic lights, as more additional traditional language features (case statements, more loop types, etc.).

CONCLUSIONS

Overall ISC has both saved time and reduced errors in scenario authoring process. The reduced complexity of inserting premade events into scenarios has enabled NADS to use staff with less experience to reliably create more complicated scenarios. Although the ISC scripting language is still a work in progress, it has already made contributions to the scenario authoring process at NADS. Because ISC adds automation to the authoring process, and does not change the underlying scenario file format, it has allowed NADS create repetitive scenarios without having to modify our underlying scenario control system.

AUTHOR BIOGRAPHIES

Optional; may be copied from paper proposal form.

REFERENCES

[1] *The LOGO fondation, What is LOGO* retrieved May 4, 2011, http://el.media.mit.edu/logo-foundation/logo/index.html

[2] A.N. Omar Ahmad, 2005, “Issues Related To The Commonality And Comparability Of Driving Simulation Scenarios", *Proceedings of the IMAGE 2005 Conference*, The IMAGE Society