Advanced HUD Indicator (v2)

John [Moggie100] Vidler

April 24, 2009

1 Overview

The Advanced HUD Indicator (Adv. HUD) was designed to allow users to draw directly onto the screen and aid in the creation of dynamic graphic displays primerily for vehicles.

While version 1 acheived this to a small degree, with the precreated graphical elements allowing developers to draw certain types of graphics with ease, freedom was limited to targetting displays and some minor system's status displays without requiring far more scripted entites (SENTs) than is convenient.

Since the development of the Expression Gate and the further development of Egate2 and the in-game editor by Syranide, it has been proven that high-speed parsing of human readable script is possible in Garrys Mod (GMod) Lua, and as such, it would a far better development to use a similar scripting approach to the HUD.

2 Language Selection

Unfortunately, the languages with already implemented parsers in Lua are unusable for the task of drawing on-screen elements without extreme modification or complex in-game scripting. Therefore it is nessicary to use a more appropriate language for *markup* of HUD elements.

I highlight *markup* in the previous sentence as important as this problem has already largely been solved. HTML has been shown to be popular and easy to understand as a language for defining elements on a screen. However, some of the concepts of web scripting languages do not transfer as well as others for HUD usage, text-flow for one is less useful in a HUD scripting language as developers may want to have text overlaying images, but will want text to be kept inside block areas.

A simpler way to create a managable script would be to use a less complex and established markup form. Enter XML.

2.1 XML - The Extensible Markup Language

XML is *ideal* for this class of problem, as it easily allows for the definition of nested blocks while allowing functionality to be extended without breaking the overall language definition, giving great freedom in the way of variables and tag names.

This freedom is the main reason for choosing XML as the language base for my markup language, however other advantages are also present, such as ease of use, as most developers have used XML in some form at one point or another, allowing them to quite easily slip in to using the markup here.

3 HML - HUD Markup Language

Obviously, XML in its raw form is not ready to be used as a full HUD scripting language, so additional tags and variables need to be defined.

However, before this can be concreted the final form of the markup must be defined, otherwise it could lead to issues later on when non-standard parameters break the schema becuase they were not considered at the start.

Consider the simple case of a small blue semitransparent box being centered in the screen;

```
<rectangle x=CENTER-10 y=CENTER-10 width=20 height=20 color=(0,0,255,100) />
```

In this simple case we have already used singletons, global variables, evaluations and vectors, giving a good point to start off the creation of a symbol set.

Tokenizing the script is relitavely simple, producing an output of...

Algorithm 3.1: L(B)

$$RACE, NAME, PARAMNAME, EQ, GLOBAL, NEG, LITERAL, PARAMNAME, EQ, GLOBAL, N$$

...which shows one form of EVALUATE, three forms of PARAMETER, three forms of VALUE one form of VECTOR, and one form of TAG... All of which is described in the HML BNF section.

4 HML BNF

This section details HML in BNF. Some of the symbols (notibly 'GLOBAL') are context sensitive, and as such cannot easily be defined in BNF, therefore, for these symbols, I have expressed their function in ”” blocks.

4.1 Special Symbols

These symbols are not really part of any definitions, but are important as they help define particular input strings more easily.

NULL = ””
HASH = ”#”

4.2 Separators

Seperators break up the input into nice processable chunks.

COMMA = ”,”
SPACE = ” ”

4.3 Math Symbols

Mathematical symbols, allowing calculations to be performed in inline expressions within tags.

Useful for positioning code, as statements such as *x=CENTER-10* give functionality that would not otherwise be possible, locking UIs into the top left hand corner of all users displays.

Additionally, the expressions can be used to scale the graphical symbols to each user's screen resolution.

EQ = ”=”
NEG = ”-”
PLUS = ”+”
TIMES = ”*”
DIVIDE = ”/”

4.4 Block Construction

Start and end marker tokens for tags to construct blocks.

LBRACE = ”¡”
RBRACE = ”¿”
SLBRACE = ”¡/”
SRBRACE = ”/¿”
LBRACKET = ”(”
RBRACKET = ”)”

4.5 Identifiers

LABEL = CHARSEQUENCE

There is an additional *GLOBAL* however, this is context specific, but basicly a list of known character sequences to check for and replace with a variable. useful for obvious constants like ”CENTER”, ”WIDTH”, ”HEIGHT” and so forth.

4.6 Special Variable Types

VECTOR = LBRACKET VALUE COMMA VALUE COMMA VALUE RBRACKET
COLOR = LBRACKET VALUE COMMA VALUE COMMA VALUE RBRACKET — LBRACKET VALUE COMMA VALUE COMMA VALUE COMMA VALUE RBRACKET
INPUT = HASH CHARSEQUENCE

4.7 Parameters

PARAMLIST = NULL — PARAM — PARAM PARAMLIST
PARAM = LABEL EQ EXPRESSION
VALUE = LITERAL — VARIABLE — VECTOR
CHARSEQUENCE = CHAR — CHAR CHARSEQUENCE
STRING = QUOTE CHARSEQUENCE QUOTE
LITERAL = NUMERIC — STRING

4.8 Tags

OPENTAG = LBRACE LABEL PARAMLIST RBRACE
CLOSETAG = SLBRACE LABEL RBRACE
SINGLETAG = LBRACE LABEL PARAMLIST SRBRACE
BLOCKLIST = BLOCK — BLOCK BLOCKLIST
BLOCK = OPENTAG CLOSETAG — OPENTAG BLOCK

4.9 Expressions

It should be noted here that expressions should follow BODMAS convention such that they are easily understandable by most people.

EXPRESSION = VALUE — VALUE OPERATOR VALUE — EXPRESSION OPERATOR EXPRESSION — LBRACKET EXPRESSION RBRACKET
OPERATOR = NEG — PLUS — TIMES — DIVIDE

4.10 Example First Pass Tokenization

The following input string;

```
<rectangle x=CENTER-10-#XOFFSET y=CENTER-10-#YOFFSET width=20 height=20 color=(0,0,255,100
```

Becomes;

```
[LBRACE] [LABEL]
[LABEL] [EQ] [GLOBAL] [NEG] [LITERAL] [NEG] [HASH] [CHARSEQUENCE]
[LABEL] [EQ] [GLOBAL] [NEG] [LITERAL] [NEG] [HASH] [CHARSEQUENCE]
[LABEL] [EQ] [LITERAL]
[LABEL] [EQ] [LITERAL]
[LABEL] [EQ]
[LBRACKET]
[LITERAL] [COMMA]
[LITERAL] [COMMA]
[LITERAL] [COMMA]
[LITERAL]
[RBRACKET] [SLBRACE]
```

(Separated by sub-block, where space == newline)