THE DEVELOPMENT OF A DOMAIN SPECIFIC MARKUP LANGUAGE FOR INTERACTIVE STORYTELLING

Denis PRODAN[1] , Rodica PRODAN[1], Sandu GAZEA[1], Daniel BUCĂTARU[1], Nicolae CHIPERI[1]

[1]*Department of Software Engineering and Automation, Group FAF-211, Faculty of Computers, Informatics and Microelectronics, Technical University of Moldova, Chisinau, Republic of Moldova*

*Corresponding author: Denis Prodan, denis.prodan@isa.utm.md

Scientific coordinator: Braga Vasili, University lecturer, Technical University of Moldova

Abstract. The problem being dealt with is the need for a more intuitive and user-friendly DSL for creating interactive stories with branching paths and multiple outcomes. The language should incorporate declarative and imperative syntax, objects, variables, and collections, and require user input in the form of choices and interactions. User-generated content and various types of output can enhance the storytelling experience. The goal is to create a program that allows for dynamic, personalized stories that engage the reader and encourage active participation.

Keywords*: DSL, storytelling, outcomes, branches, grammar, parse tree.*

Introduction

Interactive storytelling is a rapidly growing field that involves creating stories where the reader or player performs an active role in shaping the narrative. While there are numerous programming languages available for game and simulation development, creating interactive stories with different pathways and multiple outcomes can be a complex and time-consuming task. This is where a domain-specific language (DSL) can prove to be incredibly beneficial. A domain-specific language (DSL) is a programming language that is designed for a specific domain or task, rather than being a general-purpose language. A DSL that is specifically designed for interactive storytelling can provide domain experts, such as writers, game designers, and virtual reality enthusiasts, with a more intuitive and user-friendly way to create compelling stories with branching paths and dynamic outcomes. By using a DSL, domain experts can focus more on the creative aspects of storytelling and less on the technical details of programming, thereby enabling them to bring their ideas to life more efficiently and effectively. Furthermore, a DSL has more benefits, like: productivity, quality, validation and verification, data longevity, platform independence, and domain involvement [1].

Domain Analysis

The problem being addressed is the creation of a DSL that provides a more intuitive and concise way to create interactive stories. While there are numerous programming languages available for developing games and simulations, these languages don’t always make it simple to create interactive stories with different pathways and multiple outcomes. There were implemented several tools for operating with narrative text, for example Inky editor [2].

These types of languages are used by people who like to engage in action completely, so the aim is to please different groups of people, including: young adults, gamers and virtual reality enthusiasts. It should allow users to easily create branching story paths, define character actions, dialogue, and create different outcomes based on user input. This language should also be easy to learn, so that even people who have no experience with programming can use it to create compelling interactive stories.

Language Overview

Interactive storytelling is a type of storytelling in which the reader or player actively shapes the narrative. There are several fundamental structures that must be understood before developing a program or domain-specific language (DSL) for interactive storytelling. The imperative model specifies the sequence of events in the story as well as the conditions that must be met for certain events to occur. The declarative model establishes the story's characters, settings, and objects, as well as their relationships and interactions. The functional model may also be used to process and manipulate data to determine the next steps in the narrative. Objects, variables, and collections are the basic data structures in this type of DSL. Objects in the story represent characters, settings, and objects, as well as their attributes and properties. Variables store and manipulate data, such as the reader's choices and the narrative's current location. Collections group together related data, such as a list of characters or potential story paths.

The user would typically use a combination of declarative and imperative syntax to control the flow of the narrative, including branching structures to evaluate the reader's choices and actions and looping structures to repeat a set of actions multiple times. This type of program or DSL requires user input in the form of choices, interactions, and other types of user input from the reader or player. The program would use this input to control the narrative's flow and manipulate data to update the story's state based on the reader's choices. Reader choices, interactions, and inventory management are some of the specific types of input that a program in this DSL may require.

User-generated content is another important type of input that a program in our DSL for interactive storytelling may require. User-generated dialogue, character names, locations, and even entire story paths are examples of this. User-generated content can add a high level of customization and personalization to a story, making readers feel more connected and engaged with it. A program the DSL may include various types of output in addition to reader input to improve the reader's experience. Output could include, for example, dialogue, images, sound effects, and music. These elements can aid in the creation of a more immersive and engaging storytelling experience, drawing the reader deeper into the story.

Finally, a program or DSL for intuitively narrating would have to incorporate a wide run of components and usefulness to form locks in and intelligently accounts that can adjust to the reader's choices and activities. By combining revelatory and basic language structure, essential information structures, control structures, and input/output usefulness, such a program might permit journalists and engineers to make energetic, personalized stories that capture the creative ability and motivate perusers to be dynamic members within the story.

Grammar

VN = {<program>, <var>, <knot>, <ID>, <content>, <text>, <goto>, <print>, <img>, <choice>, <var_op>, <expr>, <int>, <str>, <value>, <var_name>, <knot_name>, <img_name>, <option_text>, <EQ>, <LPAREN>, <RPAREN>, <LCURLY>, <RCURLY>, <GH>, <EXLAM>, <IMG_NAME>, <INT>, <ID>, <WS>, <EOF>}

VT = {'EOF', '(', ')', '*', '=', '<', '>', '[', ']', '{', '}', '!', '+', '-', '/', '(', ')(', ')', '<', '>', '=', '[', ']', '!', a'{', '}', ',', '.png', '.jpg', ‘0’..’9’, ‘a..z’, ‘A’..’Z’}

Table 1.

Meta Notation

Notation	Meaning
<foo>	means foo is a nonterminal
foo	foo in bold means foo is a terminal
x*	zero or more occurrences of x
x+	zero or more occurrences of x
|	separates alternatives
→	derives

<program>→ (<var>)* <knot>+ EOF

<knot> → <ID> '{' <content>* '}'
<var> → <var_name> **'='** <value>
<value>→ <int>|<str>
<int>→<INT>
<str>→'"'(<ID>|<INT>)*'"'
<content>→ <text>
| <goto>
| <print>
| <img>
| <choice>
| <var_op>
<var_op> : **'['** <var_name> **'='** <expr> **']'**
<expr>: <expr> (**'*'**|'/') <expr>
| <expr> ('+'|'-') <expr>
| <int>
| **'('** <expr> **')'**
| <str>
| <var_name>
<goto> → **'('** <knot_name> **')'**
<print> → **'('**'**('** <var_name> **')'**'**)'**
<img> →**'('**'**!'** <img_name> **')'**
<choice>→ **'('** **'('** <pair>* **')'**'**)'**
<pair>→**'!'**<option_text> <goto>
<option_text>→ (<ID>|<INT>)*
<knot_name>→ <ID>
<var_name>→<ID>|<INT>
<img_name>→<IMG_NAME>
<text>→ <ID>|<INT>;
<EQ> → **'='**
<LPAREN> → **'('**
<RPAREN> → **')'**
<LCURLY> → **'{'**
<RCURLY> → **'}'**
<GH>→'"'
<EXLAM>→ **'!'**

<IMG_NAME>→ <ID> **'.png'**| <ID> **'.jpg'**
<INT> → **[0-9]+**
<ID>→ **[a-zA-Z_][a-zA-Z_0-9]***
<WS>→ **[\t\n\r\f]+ -> skip**

Parsing Tree

A parsing tree or concrete syntax tree is an ordered, rooted tree that represents the syntactic structure of a string according to some context-free grammar. The term parse tree itself is used primarily in computational linguistics. In theoretical syntax, the term syntax tree is more common [3]. For the following code snippet, the corresponding parse tree was generated (Fig. 1):

```
myvar="hello world"
second=12
f {
        ce faci132
        (y2)
        ((j))
        (!car.jpg)
        ((!what are you doing (y2)
        !here (y4)))
   }
y{
hi io hi
}
```

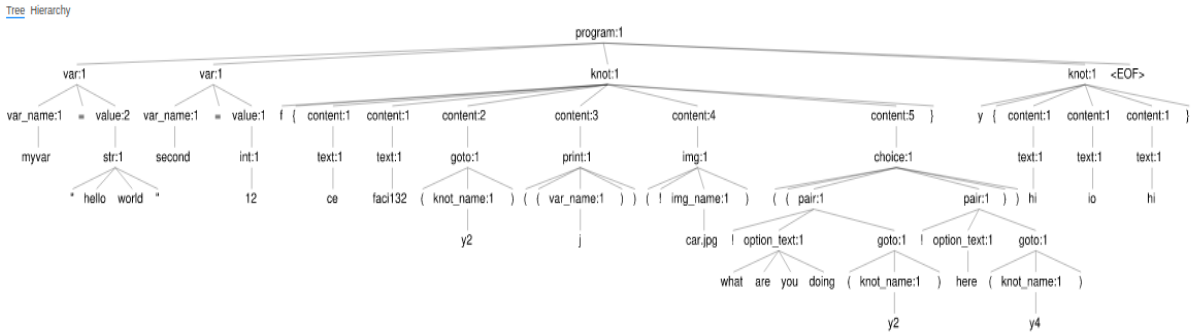

Figure 1. Parse Tree

Conclusions

Domain Specific Language can be useful for different tasks, purposes and people. The difference between Interactive storytelling DSLs and other programming languages is that they must cope with outcomes, branches and characters. These notions are naturally understood by humans, but can be complicated to be transposed into code. This article introduces a new DSL concept, which develops the idea of a more friendly markup language for interactive storytelling.

References

1. VOELTER, M. DSL Engineering: Designing, Implementing and Using Domain-Specific Languages. Germany, 2013.
2. Writing web-based interactive fiction with ink, [online]. [accessed 05.03.2023]. Available: https://www.inklestudios.com/ink/web-tutorial/.
3. Compilers: Principles, Techniques, and Tools [online]. (accessed 05.03.2023]. Available: *https://en.wikipedia.org/wiki/Compilers:_Principles,_Techniques,_and_Tools#cite_note-1*