Textual Authoring for Interactive Narrative

Samuel Lynch[1], Charlie Hargood[1], and Fred Charles[1]

Creative Technology Department, faculty of Science and Technology, Bournemouth University,
Poole, Bournemouth, UK
<i7420737, chargood, fcharles>@bournemouth.ac.uk

Abstract. Narrative design and implementation in interactive softwares has always possessed an intrinsic issue, in that the person or persons with the responsibility of authoring the narrative will most likely lack ability in programming. Because the narrative does eventually become implemented in software, the influence the technical side has on the creative side is unavoidable. The design of the *March22* Engine, its scripting language, and its accompanying authoring tool are all designed to facilitate those of least programming ability, whilst not limiting those of greater ability. By making the scripting language as similar to a screenplay as possible, and showing the flow of narrative via charts, the writer is left with as little programming requirements as possible, whilst still able to produce quality narrative.

Keywords:

1 Introduction

In interactive narrative - specifically game software - the scope of the project might necessitate the utilisation of an author, possibly with experience writing interactive fiction, but not necessarily, especially as it remains difficult to find authors with the required experience. Even one with experience in writing interactive (i.e. narrative that changes flow-based on user interaction) may not possess experience, ability, or skill with programming. Because interactive narrative is almost always software-based, programming is an unavoidable obstacle to quality narrative (as well as a bug-free experience); an obstacle that, in most instances, the author is expected to overcome.

This is an important issue to address, because it creates an entry barrier to the Writer position in interactive software. Creative positions should not possess entry barriers beyond the intrinsic requirement to produce assets in their area (e.g. Autodesk Maya for 3D artists). Ideally, the writer should have zero requirements other than the ability to write, and the requirement of having programming competency- however minimal- is a large one. Not withstanding is the immense task of writing interactive innately; a rule of thumb with writing interactive is tripling the word count at the least, so facilitating the authors is not desirable so much as necessary.

To that end, the March22 Engine was designed and implemented with ease of use in mind. Not just for authors, but for developers and artists alike. With a low entry barrier to usage, the content created can be of a quality more reflective of the developer's respective abilities. And with the framework being built upon the Unity 5 engine- as well as covered by the permissive MIT license- there are no monetary entry barriers either (besides Unity's highly permissive royalty scheme).

2 Background

It is notoriously difficult to tackle the issues related to developing accessible solutions for the creation of interactive narrative applications. This is even more so the case when trying to create authoring systems which bridge the potential gap between the underlying narrative systems generating the compelling narrative experiences for the users. For more than a decade, authoring tools have been designed and devised often for the sole use of authors of these pre-existing interactive narrative systems, such as EmoEmma [2, 1], Thespian [5], Expressionist [4], also applied to the topic of location-based narratives [3].

It has been identified that there exists differing ways and strategies to make a story interactive, and that the creative approaches suggested by these narrative systems differ fundamentally. This further increases the difficulty to find any commonality between the approaches for 'Interactive Storytelling from a creators point of view.

Beyond the requirement for authoring tools to be supporting the underlying technologies utilised to generate interaction and multiple story lines, the creation process in itself remains an unanswered problem even though there has been attempt at leveraging this process [6, 7].

3 Design

The decision to use the Unity engine was the eventual choice, not the first. Originally, the engine was designed and built in C++, but this proved to be far too complex for the scope, as it meant most simple features had to be written from scratch. It provided extremely lightweight software, but could only feasibly run on Windows and Linux. It never came to it, but the ability to extend script functionality was planned to use either ECMAScript or Lua, to prevent the need to recompile the C++ source code.

A second version of the engine was produced, written in Lua via lpp-vita for PS Vita, and Love2D for PC/Mac/Linux. While this version was popular among 'homebrew' PS Vita developers, it was not future-proof and shared a similar problem to the C++ version, and as such was discontinued. It did possess a feature that other versions still lack, and that is precompilation; the scripts would be written in *.rpy* format (a slightly modified implementation of Python for use with the Ren'Py engine) and compiled into Lua as an array of objects, which could then be JIT compiled. This had the benefit of facilitating writers

familiar with Ren'Py (as both this and March22 are Visual Novel engines) but the downside of an extra step for the writers to concern themselves with.

The Unity port was re-written and re-designed from the ground-up to be the best possible implementation. It could become a framework for narrative in all games, as opposed to being limited to an engine for a specific genre, and Unity itself is already an easy-to-use tool with great documentation, so it helps promote the idea of ease of access. In addition, Unity utilises a package system to install modules/frameworks such as March22 with little difficulty, and boasts greater performance and wider platform coverage than alternative solutions such as Ren'Py (which supports Win/Mac/Linux, Android/iOS, and ChromeOS, whereas Unity supports up to twenty-seven unique platforms).

With the idea of facilitating the authors first and foremost, the March22 scripting language was designed to be as close to regular writing as possible. This essentially boils down to the functions being very simple and easy to remember, as well as very few functions being necessary to every script. But to prevent the language being limiting to developers, the 'CustomFunction' module can be utilised to expose more functionality to the script. For example, if one were to use the March22 framework in a Pokmon fan adaptation, a custom function would need to be made to, say, commence a battle at a certain point in the narrative. The script/narrative files were intended to be simple plain-text, so that they could be authored, edited, and "exported" in any software of choice.

However, a number of issues can arise from this, ranging from metadata exported from Microsoft Word or similar corrupting the script, to the script file becoming unmanageable with many lines. The problem of unsupported characters (from various encodings of Unicode) is also an issue. Therefore, it is most ideal to create a centralised tool with which to produce the scripts, while still only exporting/importing a plain-text file, so that 'power-users' can still use their preferred editor at their own leisure. This was the authoring tool.

4 The Authoring Tool

The tool, just as the engine it is made for, is built for ease of access and use. Building it in HTML and JavaScript made the most sense at the time for this goal, as this meant there was no concern for supporting specific platforms or devices (i.e. any device with a web browser), and HTML already possesses all the required frameworks (i.e. text boxes and input). Additionally, it removes the requirement for extra software; only a web browser is required.

For frameworks that did not exist intrinsically, both HTML and JavaScript possess a wide range of open-source projects that provide the required functionality. Frameworks such as *VisJS* for visualising the flow of the narrative, *Ace* for editing the script by hand, and *FileSaver.js* for exporting the script files. The most important of these three being the foremost, as visually representing the flow of an interactive narrative is invaluable to an author, as it helps prevent undesired crossover or indicating where a story becomes too complex.

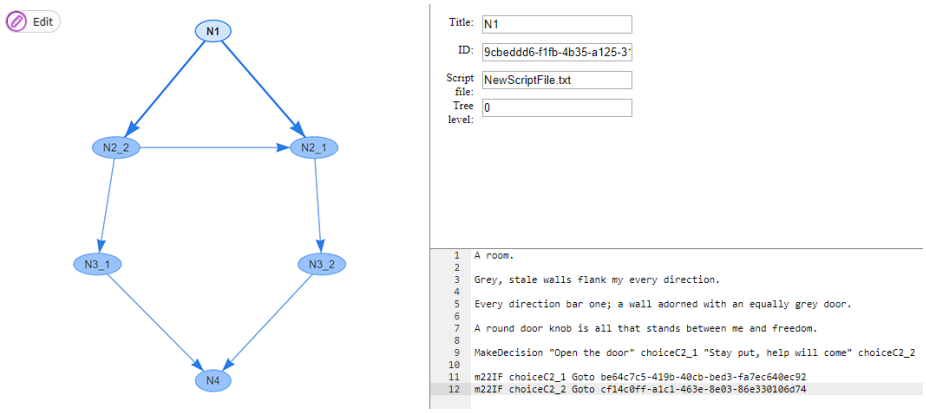

Fig. 1. Authoring tool's GUI (current stable version)

Because the engine is primarily designed for Visual Novel games, with generic interactive narrative being secondary, the authoring tool has to be aware of the necessary components of a Visual Novel script. Components such as character/sprite display, background/scenes, music and sound effects, etc. While the current version of the authoring tool does not display this information to the user yet, the backend tracks all the required resources on a per-script basis, which paves the way for exporting entire projects directly from the authoring tool. Were the authoring tool cloud-based, each project could have its assets contained and organised, so aspects such as reusing existing assets is easier.

Originally, the authoring tool was a simple visualiser; the user added narrative via a text box, including functionality and metadata, and the flow diagram would update to match the input. Additionally, the script compiler was meshed-out in JavaScript, so that the script could be soft-compiled for error-checking/debugging, as well as allowing the import/export of the script files in their raw format. However, maintaining a compiler in both C# and JavaScript was taxing. While Unity does utilise a script language akin to JavaScript called 'UnityScript', it is not entirely the same and is not necessarily inter-operable with web-based JavaScript. Development versions have phased out the compiler completely, in favour of saving in-progress work as JSON, and exporting the script files when needed.

The visualiser was quite effective at showing improper narrative links. Take the above image as an example; the second node or 'chapter' (marked as 1 in the visualiser) skips past many other nodes- including a branch separation from the main flow of the story- and jumps to a much later node. This is obvious from the arrow/link that visibly skips over said branch.

However, the early versions of the tool relied almost entirely on the compiler for the visualisation, meaning that larger scripts produced incoherent and unmanageable charts. It also meant that linking nodes had to be written into script by hand; introducing more programming to the author. Later versions

introduced the ability to link nodes via the visualiser, by dragging arrows to the destination node, and the code to achieve this would be inserted into the narrative automatically.

These later versions also improved on the customisation of the visualiser; previously a node's position was locked and determined by the compiler, which is ideal only if the compiler is perfect. Otherwise, the user must be able to move the nodes freely, so as to rearrange them in a way that is easier to understand whilst writing. Interactive narrative has a tendency to become difficult to visualise, due to the erratic flow of narrative- as is expected with interactive media.

A standard pipeline for writers in interactive could require the game script to be processed by a programmer, after being edited and 'completed'. This is a clear waste of resources that could be minimised by having the game script processed by the writer. Since the writer generally lacks programming ability, this requirement would need to be lessened in order to save resources. The scripting language minimises it but does not eliminate it by making the language as simple to understand as possible. The authoring tool lessens it further by providing an auto-complete to function names, with plans to add the ability to select functions from a list to place into the script.

The language additionally possesses inline functionality; this essentially means that engine functions can be called in the middle of narrative. The 'typewriter effect', where text appears on-screen at a certain speed rather than all at once, permits the use of these inline functions to create better experiences for the player. A minimalistic use of this could be to force pauses in the typewriter effect, creating a unique flow of narrative, while a more daring use might be to play a gunshot sound effect and shake the interface whilst a character is speaking. This functionality is useful and interesting to writers, as it provides them with more storytelling devices, but it also acts as a double-edged sword; it presents another aspect to have to educate the author about in order to fully utilise the engine, and it cannot be easily integrated into the authoring tool.

Currently, the intention is to repackage the authoring tool to be an extension within Unity, while also maintaining the ability to use a web-browser. This is so that the ability to compile/debug script is available to the writer to minimise or eliminate error, without having to re-code the compiler into JavaScript. And this ability can simply be disabled when run from a web-browser, so the author can still write script, but cannot debug. Additionally, the prototypical interface needs to be redesigned to be more approachable. Fortunately, the tool being in HTML- with CSS- facilitates the creation of designs.

5 Conclusions

The system and tools are far from complete; the current development task for the engine itself is reworking how script functions are handled entirely (and overhauling the compiler with it), which in turn means that the authoring tool needs to be adapted to match this. The tool does not possess the compiler, but does need to be aware of available functionality, and the rework affects the aspect

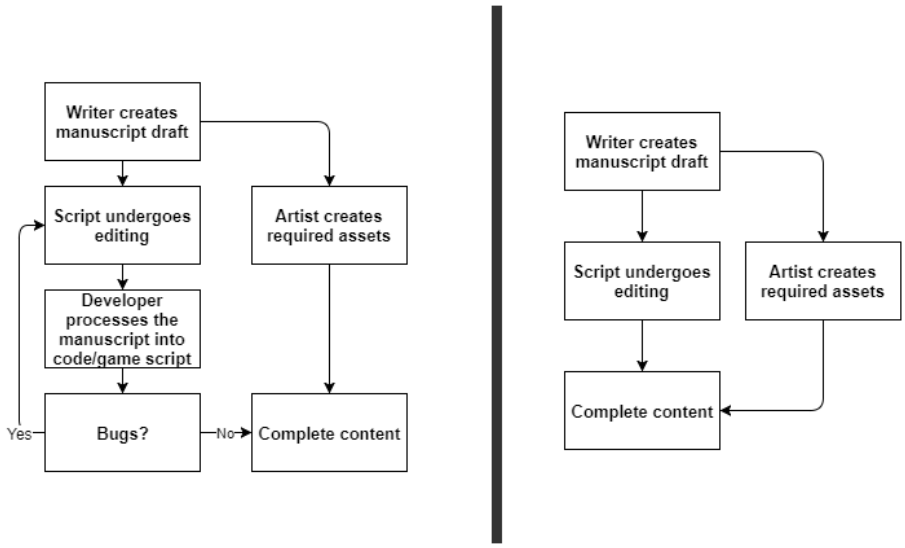

Fig. 2. Standard pipeline versus the March22 pipeline.

of the framework for extending the language (i.e. adding extra functions). Then the problem arises of when there is many functions- custom or otherwise; this is a problem from both a regular and technical standpoint. The engine currently offers twenty-five built-in functions, with less than half of these being exclusively useful to Visual Novel game development, so care must be taken when adding more functionality, lest it overcomplicate the use of the framework.

Programming a tool that essentially exposes the programming-uninitiated to programming presents an especially trialling task, since it can be difficult to ascertain whether an aspect of either the tool or the engine is 'too much' for them to understand. Hiding as much technical (i.e. unnecessary to most persons) information from the user, and providing a friendly interface for them, is as much as can be done without direct testing and feedback. The same is true for the script language itself; while feedback from people have sampled the tool has been positive, the feedback from those who genuinely utilise the tool for a project is more desirable.

The back-end of the engine has objectively benefited from having been rewritten in several languages. Certain aspects such as the compiler has been optimised with every rewrite, making it more efficient with larger scripts that contain many multimedia elements. Some parts of the back-end have been deprecated entirely, such as Unicode parsing, since Unity/C# provides a far more optimised system for this.

The key intention of the March22 framework and its scripting language was to provide accessibility to the inexperienced, and power to the experienced. The most basic scripts that March22 can run could be just five sentences; no func-

tions, no metadata required. While the most complex scripts are akin to Ren'Py script, with multiple nodes interconnected, and could be in a quadruple-digit line count. This intention is currently realised, but care must be taken so as to not overcomplicate the operation of the framework. Documentation on how to use the engine is being created alongside the documentation of the source code, with tutorials from getting started with the engine to publishing a title using it.

References

1. Marc Cavazza, David Pizzi, Fred Charles, Thurid Vogt, and Elisabeth André. Emotional input for character-based interactive storytelling. In *Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems-Volume 1*, pages 313–320. International Foundation for Autonomous Agents and Multiagent Systems, 2009.
2. Fred Charles, David Pizzi, Marc Cavazza, Thurid Vogt, and Elisabeth André. Emoemma: Emotional speech input for interactive storytelling (demo paper). In *The Eighth International Conference on Autonomous Agents and Multiagent Systems*.
3. David E Millard and Charlie Hargood. Location location location: Experiences of authoring an interactive location-based narrative. In *Interactive Storytelling: 9th International Conference on Interactive Digital Storytelling, ICIDS 2016, Los Angeles, CA, USA, November 15–18, 2016, Proceedings 9*, pages 419–422. Springer, 2016.
4. James Ryan, Ethan Seither, Michael Mateas, and Noah Wardrip-Fruin. Expressionist: An authoring tool for in-game text generation. In *Interactive Storytelling: 9th International Conference on Interactive Digital Storytelling, ICIDS 2016, Los Angeles, CA, USA, November 15–18, 2016, Proceedings 9*, pages 221–233. Springer, 2016.
5. Mei Si, Stacy C Marsella, and David V Pynadath. Thespian: An architecture for interactive pedagogical drama. In *AIED*, pages 595–602, 2005.
6. Ulrike Spierling, Sebastian A Weiß, and Wolfgang Müller. Towards accessible authoring tools for interactive storytelling. In *International Conference on Technologies for Interactive Digital Storytelling and Entertainment*, pages 169–180. Springer, 2006.
7. Nicolas Szilas and Steven Wingate. Exploring new approaches to narrative modeling and authoring. In *Interactive Storytelling: 9th International Conference on Interactive Digital Storytelling, ICIDS 2016, Los Angeles, CA, USA, November 15–18, 2016, Proceedings*, volume 10045, page 464. Springer, 2016.