Contemporary Issues in Interactive Storytelling Authoring Systems

Daniel Green, Charlie Hargood, and Fred Charles

Bournemouth University, UK
{dgreen, chargood, fcharles}@bournemouth.ac.uk

Abstract. Authoring tools for interactive narrative abstract underlying data models to allow authors to write creative works. Understanding how our program and interface design decisions alter the User Experience design could lead to more robust authoring experiences. We contribute a taxonomy of authoring tools with identified program and User Experience observations with discussion into their impact on the authoring experience as well as reflection on two detailed experiments. We then present our own authoring tool, Novella, and discuss how it has implemented the lessons learned from the analysis and how it approaches solving the identified challenges.

Keywords: Interactive Narrative · Authoring Tools · User Experience.

1 Introduction

The design of interactive narrative authoring tools is an emergent process from the underlying narrative data model that the tool supports. While this makes sense from a structural point of view it is possible that we are ignoring the User Experience (UX) impact of a variety of design decisions and interface paradigms. Without well-designed UX, accessibility is reduced and systems become restricted to users with the appropriate technical know-how, which can result in a frustrating user experience and can contribute to a reduced rate of adoption by interactive fiction communities. It is also possible that there may be repeated UI trends in the design of these authoring tools that may have an unforeseen impact on the resulting stories.

In this paper, we survey the current state of the art and present a categorized taxonomy of authoring tools for interactive fiction, a listing of what we believe to be the prominent challenges facing the interactive fiction authoring tools community accompanied by a more in-depth considering of two commonly used tools: Twine [11] and inklewriter [7]. We also present progress on our own authoring tool, Novella, which implements our previous model [18] and attempts to target some of these challenges. Our analysis focuses on authoring tools for structural and choice-based narratives rather than simulated or generative ones.

2 State of the Art

In our survey of the state of the art, we have considered 29 authoring tools. 14 were sourced from academically published research. 4 are developed and sold as commercial products. The remaining 11 come from other non-commercial, non-academic sources, such as open-source or otherwise free projects. We have found this distinction to be important, as the purpose of these tools differs based on their origin; commercial products are created with a different end-goal than academic systems, for instance, which can impact the focus and quality.

Delivery Methods & Interface Paradigms

These tools can be broadly categorized by their delivery methods (**Standalone**, **Web-based**, **Integrated**) and high-level interface design paradigms (**Form**, **Graph**, or **Text**-based).

Standalone tools offer a dedicated application which is independent from additional software other than the host platform. A drawback of such approaches is the difficulty of cross-platform support, although this is becoming less problematic. The Emo-Emma [15] authoring tool, for instance, provided a Windows binary, but does not support any other systems. Some standalone applications are able to export to generic or specific formats, which increases the usability of the tool. articy:draft [4], for example, can export to Unity [12] as well as providing an API for integration into arbitrary systems.

Web-based tools provide a browser-based solution which comes with the advantage of being mostly platform independent and easily accessible. Editors such as StoryPlaces and inklewriter present web interfaces for authoring, testing, and publishing of interactive fiction. Twine, which is based on web technologies, provides both a browser interface and a wrapped standalone multi-platform desktop application, which increases its accessibility and availability.

Integrated tools are built directly into host software such as game engines, having the advantage of being able to communicate directly with and be tailored to a given system. Fungus [5], for instance, is built as a Unity plugin, requiring no other software and being able to integrate directly with Unity's systems.

Table 1 presents an incomplete but representative taxonomy of authoring tools for interactive narrative classified by their method(s) of delivery.

Authoring tools can also be broadly grouped by the paradigms used in the interface with abstraction of data. It is to be noted that these paradigms are not mutually exclusive, and many tools make use of one or more in their design.

Form-based interface design is the most common interface paradigm by a large margin. These are atomic user interface controls that are often mapped closely or directly to a data model. For instance, a character editing interface using a form-based approach may contain text fields directly mapped the character's attributes. These kinds of fields are necessary in essentially any tool that requires data input. Their danger lies within misuse, as creating an interface

Standalone	**Integrated**	**Web**
ACADEMIC		
ASAPS [24], DraMachina [16], Emo-Emma [15], FearNot! [25], GAIA [23], NM2 [31], PaSSAGE [30], Scenejo [17], StoryTec [20], SVC Editor [32], Virtual Human [21]	GHOST [19], Story World Builder [28]	StoryPlaces [22]
COMMERCIAL		
articy:draft 3 [4], HyperCard [1], Storyspace [2] Tinderbox [14]	None	None
OTHER		
Inform [6], Quest [8], Ren'Py [9], Squiffy [10], TADS [29], TextureWriter [26], Twine [11]	Fungus [5]	Genarrator [3], inklewriter [7], Playfic [13], Quest [8], Squiffy [10], Twine [11]

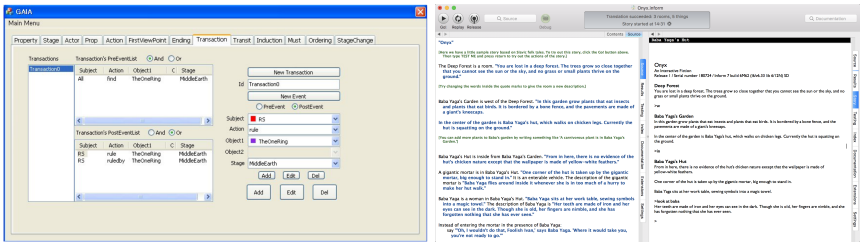
Table 1: Authoring tools sorted by their method(s) of delivery.

purely of atomic controls can be difficult to maintain good UX, and can have little tangible benefit other than basic accessibility when it comes to abstraction of an underlying model for the layperson. The GAIA authoring tool [23] displayed in Fig. 1a demonstrates heavy use of a form-based editing approach.

Graph-based editors abstract the structure of data into visual graphs, the most common being a node-line graph. They may be for pure visualization purposes, such as in inklewriter, or as a core part of the actual editing experience, as in Twine. This abstraction of connected data into a more visual form provides a tangible benefit that is not possible with simpler form-based approaches. Fig. 3b shows the Twine editor's graph view, which provides an overview of the story data and its connectedness in a node-line style editable interactive graph.

Text-based systems feature an augmented plain text editor as their core interface paradigm, providing additional features such as autocomplete, syntax highlighting, and other markup. This approach is commonly found partnered with a domain-specific, or other form of scripting language. For instance, the GUI application of Inform [6] is centered on a plain text editor with integration into the Inform language such as markup and simple debugging support. The Inform interface in Fig. 1b shows an augmented text-based implementation with full syntax highlighting based on the Inform language.

It is important to consider the way in which accessibility can be affected when choosing a delivery method. Standalone distributions, for instance, empower the possibility of native performance and can offer a homogenous experience within the host platform, but do so at the expense of difficulties encountered with cross-



(a) (b)

Fig. 1: GAIA (a) and Inform (b) using form-based and text-based designs.

platform native development. Supporting only a single platform can reduce accessibility of a system. Web-based systems, on the other hand, have increased accessibility due to their platform independence, although are faced with cross-browser complications and lesser performance. As technologies advance, however, cross-platform development difficulties and performance gaps, among other impediments, are shrinking, resulting in more freedom of choice with regards to delivery methods. Integrated solutions can tie closely into existing systems, but in doing so restrict themselves in scope.

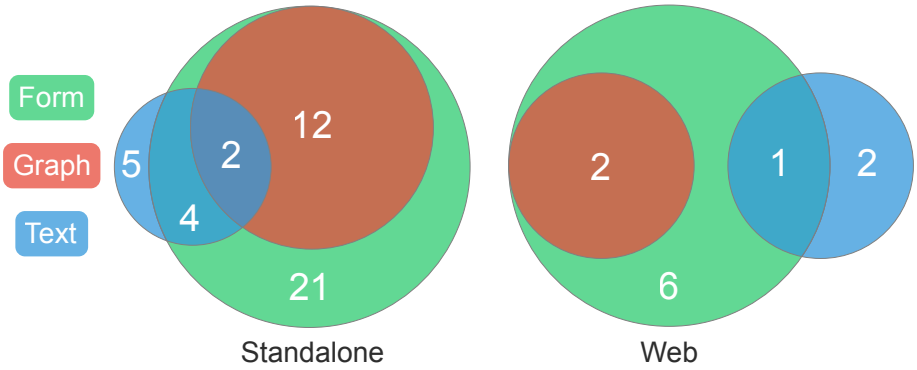


Fig. 2: Distribution of interface paradigms for Standalone and Web delivery methods. Integrated is not included as it has only 3 form and graphs, which both overlap.

Fig. 2 shows Venn diagrams of the interface paradigms for standalone and web-based authoring tools. Each paradigm presents its own advantages and disadvantages to the authoring process. Using a given paradigm does not necessarily mean that the UX will be a certain way, as this is largely dependent upon the implementation. We can, however, conclude general observations of trends

regarding accessibility when these paradigms are used. Based on our analysis, we found that text-based interfaces provide the most power to the user due to their exposing of a narrative grammar for authoring, but do so at the expense of usability. Authors must first learn the grammar and have little to no assistance beyond highlighting and documentation, which can make the authoring process tedious, especially for new or non-technical authors. On the other hand, we found that graph-based systems were the most accessible due to their prowess at data visualization, which is an ideal match for complex intertwining stories that are difficult to represent visually in text-based systems. This does come at the cost of losing some control that is provided by text-based systems due to the abstraction provided by graphs. Form-based approaches are variable, reliant on the visual presentation of the atomic controls. If well done, they can provide a good abstraction of the story data with reasonable levels of control, albeit less than a text-based approach. Caution must be taken when using form-based approaches to present meaningful data in a meaningful way, and to not overload authors with too much at once. This is evident by the number of systems, not only in interactive narrative, that appear to be designed by technical users for other technical or power users, rather than what the author really *needs* to see.

We summarize by concluding that these three paradigms (**Form**, **Graph**, or **Text**) play different roles in the design of interactive fiction authoring tools but that each has an impact on the UX - principally on *Accessibility* (the ease of use to construct a story), *Author Power* (the degree to which the author is able to express more detailed systems), and *Content Fidelity* (the accuracy with which the paradigm presents narrative structure or content specifics). Our survey already highlights potential trends in these design paradigms - **Graph** interfaces demonstrating a lower degree of *Author Power* than **Text** interfaces, but higher *Accessibility* - or **Graph** interfaces demonstrating good *Content Fidelity* at the structural level but potentially less at the specific content level than **Form** interfaces. A comprehensive review of the exact nature of the impact in these paradigms must form part of the future work in this area.

Availability

When evaluating the availability of an authoring tool, we looked at its presence online through official websites and third-party mentions (such as Redcap[1]), as well as the accessibility of both binaries and source code.

The availability and longevity of a tool directly affects its rate of adoption and its ability to be additionally developed or used otherwise for further research. Systems that have long become dormant, for instance, can become devalued relative to their initial contribution, and are unlikely to be adopted in the long term by interactive fiction authoring communities. Inform, for example, has been used and developed since 1993, but due to its online presence and community-driven development, remains a strong contender even today.

[1] `http://redcap.interactive-storytelling.de`

Online presence is an area that academic authoring tools struggle to maintain. Of the sampled tools, only six ever had a dedicated website, of which only four are functional today and are seldom updated. Temporary websites often result in unreachable links and are not a suitable substitute for a dedicated page where tools can be publicized. To illustrate, an article of Emo-Emma[2] provides links to academic papers and binaries, but all links reside on a university staff page that no longer exists, making it difficult to source the original contribution.

Another related area that academic works struggle with is the distribution of binaries and source code. Some projects are understandably protected intellectually, but those that are not should attempt to share their work in order to best maximize its chance of adoption and provide an opportunity for further research to be conducted. Of the fourteen sampled tools, only five offered binaries at some point, with only *one* remaining available (StoryPlaces [22] which offers both the application and source code). Sometimes software is available upon request from the authors. StoryTec [20], for instance, used to provide a software request form, but has since removed it in 2015. The authors followed protocol outlined by ASAPS [24] website for requesting the software, but were unsuccessful in obtaining it. This highlights the need for care to be taken to ensure that if software is intended for public use, that it is made available, and remains easily available, otherwise any traction gained could be rapidly negated. The lack of availability of academic software also has serious research implications for this community, as it prevents the reproduction of any experimental results and hinders their study so that the community might incrementally improve and iterate on their work, or form a greater understanding of this area. While some conclusions can be drawn from documenting articles this is not a replacement for the software itself, and software availability and sustainability can be seen as a notable challenge for this community to overcome.

User Experience

In order to better understand the UX of select authoring tools, we participated in a reflective autoethnography by taking a story segment from Life is Strange[3] and recreating it using inklewriter and Twine. The chosen story segment[4] consisted of multiple choice discourse with some looping pathways and certain options becoming available only once set conditions were met (i.e. a given pathway node was experienced). We firstly familiarized ourselves with the tools by creating sample stories containing all of the features of our chosen story segment (i.e. how to branch, loop, present multiple choices, and lock choices until conditions are met). Each experiment was timed and narrated, and both audio and video were captured for further reflection on the authoring experience. Following the completion of the authoring process, UX laws and heuristics were applied,

[2] `http://redcap.interactive-storytelling.de/authoring-tools/emo-emma/`

[3] Life Is Strange, Dontnod Entertainment, 2015

[4] Section titled **Conversation with Juliet** from the game's script: `http://life-is-strange.wikia.com/wiki/Episode_1:_Chrysalis_-_Script`.

and any sources of usability frustration were noted. Each story was also tested to ensure accuracy, which is included in the recorded timing. To evaluate the UX of authoring systems, we applied a selection of heuristics as described by Nielsen [27], as well as the listing of the Laws of UX[5].

inklewriter is a web-based authoring tool of interactive narrative shown in Fig. 3a. In the central pane, texts are displayed as detached segments, each of which can be edited inline. There is also a content browser for texts, and a static node-link graph view for better visualizing the flow of the story.

The *Doherty Threshold* states that system feedback should be $\leq$400ms else user attention and productivity can suffer. However, the text segments in inklewriter animate before being presented, which takes around one second. While this may seem insignificant, if working on a large project that requires lots of navigation, the extra time can add up and reduce the overall authoring experience.

The *Law of Similarly* advises elements of differing functionality to be visually dissimilar, and those that are similar to be treated as related or as a group regardless of physical separation. In inklewriter, segments are accompanied by informative red italic text describing things such as the number of incoming or outgoing links. However, errors are also presented in the same visual style. Differentiating between information and errors can reduce potential confusion for users and help them focus on authoring rather than understanding the system.

Hick's Law suggests reducing complexity where possible, as the time it takes to make decisions or take actions is altered by the number and complexity of options available. In order to delete an unwanted segment, it must be first unlinked from surrounding segments, and then becomes detached. Then it must be located in the content explorer and manually removed. While this is likely by design in order to retain discarded texts, the complexity involved for the desired removal of such texts is high and could be reduced significantly. Keeping complexity low can aid UX and increase productivity.

Two of Nielsen's heuristics state that shortcuts should be included in order to satisfy experienced users, and that actions made should be easily reversible, or warned if they are not. inklewriter does not provide notable shortcuts and does not has the ability to undo changes beyond in text fields.

The total time spent authoring the story was 17m20s. This is reasonable given the size of the story, but could have been improved, even if only a little, by the interface being less delayed.

Twine is a web and desktop authoring tool displayed in Fig. 3b. It uses a connected node graph for visualization and editing of the narrative texts. Content is edited through modal popups that fill the screen.

Jakob's Law specifies consistency in an array of manners, one of which is standardized and recognizable icons. While Twine's icons are largely consistent

[5] `https://lawsofux.com` by Jon Yablonski. Many of the laws are grounded in research, and the few that are not are widely accepted heuristics.

with those commonly used, its icon for 'find and replace' represents a list rather than something that would obviously indicate searching. Where possible, we should reuse common or recognizable adaptions of icons to reduce users having to guess functionality and to take advantage of already learned connections.

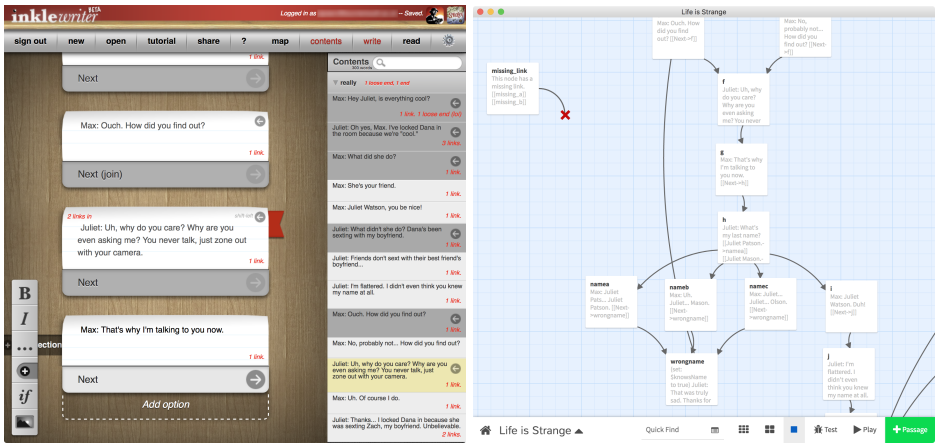
Miller's Law focuses on reducing cognitive load. Twine makes no major violations as such, but entering links, which is done manually, could be improved by providing autocomplete of existing nodes in order to reduce the number of items a user has to memorize at once and help mitigate errors.

The *Zeigarnik Effect* is about informing users of task progress. When a node in Twine defines a connection to another node that doesn't yet exist, it is replaced with a bold red **X**, indicating the incomplete state of the task. This could be improved, as regardless of the number of missing links, only one **X** is displayed. Showing multiple corresponding with the number of missing links would better implement this rule. Additionally, *Parkinson's Law* - the saving of time within a given task - could also be implemented here by allowing missing nodes to be created and linked by clicking the corresponding **X**.

The same two of Nielsen's heuristics that failed for inklewriter are likewise not implemented here. Accelerators are not provided beyond overriding a warning message when deleting nodes. Since Twine relies so heavily on a visual graph, it is ideal to include accelerators such as context menus to speed up development. Similarly, reversal of actions is not supported outside of modal content editing dialogs, which is scoped to the current modal session. After closing and reopening a modal dialog, the undo state resets, and requesting undo will delete all of the node's content. When implementing reversal of actions, we must ensure that they successfully return to a previous state.

The total time spent authoring the story was 20m48s. This is significantly longer than inklewriter. It is likely that this is due to Twine requiring each node to have a unique name, and for the links to be manually typed using the exact names. Supporting autocomplete or adding an accelerator to create named nodes could potentially decrease the authoring time.

While the scope of our analysis is limited, we can still draw observations on the common problematic areas encountered. Authoring flow is an important concept and we must design our UX to minimize interruptions. In the case of Twine, having to manually open and rename each passage, and then remember the names to setup the links creates cognitive friction and can obstruct the authoring experience. With inklewriter, the delay for elements to enter the author's view, while less of a factor, can still contribute to slowing down the authoring process. We must consider our design decisions and prevent, or at least mitigate potential blocks in disrupting the authoring flow. Effective state communication ensures that the system and author are on the same page with regards to the story state; if there is a misunderstanding of state, it could leave the author guessing. Supporting the ability to undo and redo story state changes facilitates and encourages experimentation, which is important with creative works. Twine could better support experimentation if it better supported undoing, and had less faults when reverting textual content. In inklewriter, information and errors



(a) (b)

Fig. 3: A segment of Life is Strange written in (a) inklewriter and (b) Twine 2.

being conveyed in the same visual manner result in possible confusion for the author as to the current state of the story, making it difficult to differentiate mistakes from information. In a similar manner, we must strive to maintain the drafting process than an author expects by avoiding things that disrupt editing expectations. For instance, inklewriter's graph system at first appears as an alternative editing method, but is actually a static preview of the story state and connectedness.

3 Novella

We have taken the new game-centric model of interactive narrative from our previous work [18] and implemented it alongside an accompanying standalone authoring tool prototype. This model focuses on game-specific narrative structure, but can readily represent traditional interactive fiction also. The authoring tool incorporates the UX lessons we have learned from this analysis, and represents a first prototype towards solving the challenges we have identified. Fig. 4 demonstrates an example of the editing interface.

Our standalone tool primarily incorporates a graph-based paradigm, but also includes elements of both form and text-based solutions. The majority of the authoring is done through the central node-line graph. This was chosen as we felt that graphs provide the most accessible visualization of connectedness. Each node represents some form of narrative element from our model, such as describing discourse, providing context, and so on. Content of nodes is edited through detachable popover windows, reducing UI clutter. Variables and Entities are edited through separate interfaces accessible from the toolbar.

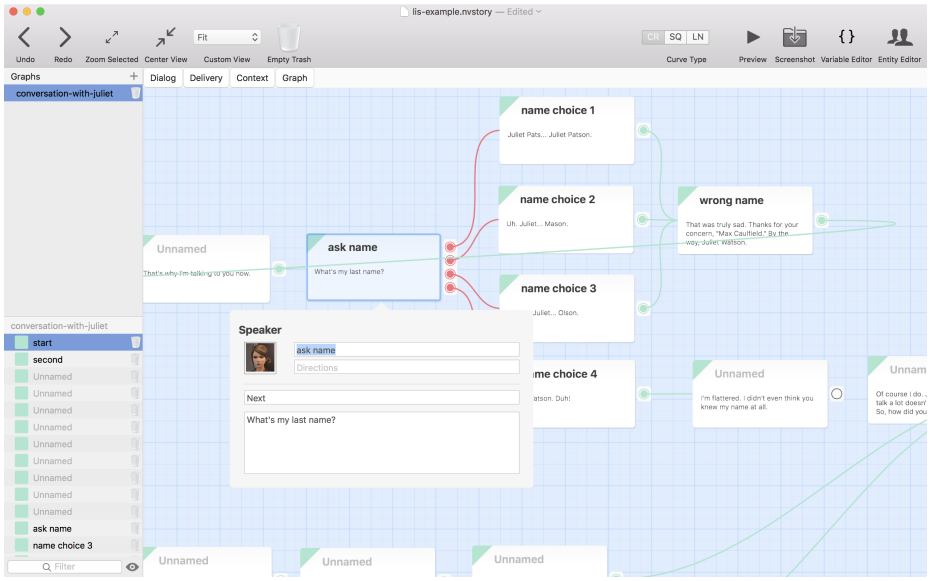


Fig. 4: Life is Strange story segment created in our own tool, Novella.

We should always strive to present our stories and their states as clearly as possible to reduce cognitive friction of authors and help them focus on writing rather than figuring out the system. Providing visual aids and reducing task complexity can contribute to maintaining author focus.

In Novella's graph system, each node has a distinctly colored flag in the top-left corner that represents the abstracted data from the internal model. For instance, dialog nodes are always colored teal, and delivery nodes orange (Fig. 5). These colors are used consistently throughout the interface which eases recognition at a glance of which kind of data is being handled thanks to the Law of Similarity. These flags follow the Serial Position Effect, where extremities are often more likely to be observed and remembered. Similarly, the green triangle in the top-right, representing that a given node is an entry point, likewise implements this law.

If nodes have outgoing links, we append a disconnected floating *pinboard* which appears grouped with the node due to the Law of Proximity (Fig. 5b). Pinboards contain pins representing each outgoing link. Pins are bordered to separate them from their neighbors, which is especially important for branch links, which have two pins (true/false). The pins and their curves are colored based on the type of node they connect to. This means that the type of node that follows can be rapidly identified at a glance based on the color of the pin and its curve. In dense graphs where curves are difficult to follow, this becomes especially useful. To further reduce complexity following curves in dense graphs, selecting a node will highlight all of its pins and curves to make them stand out

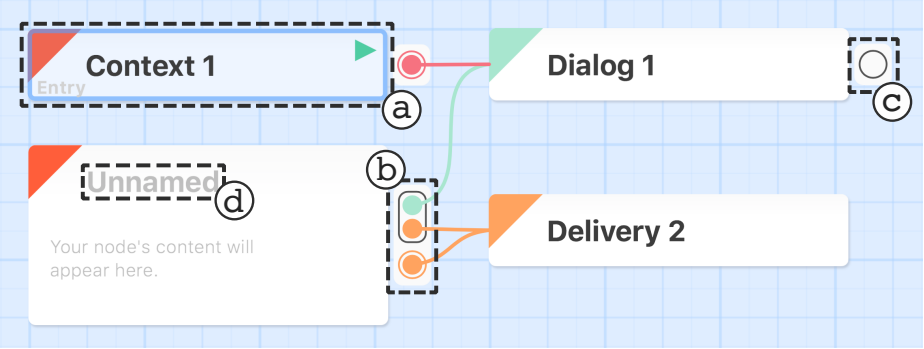


Fig. 5: Some Novella node UX features. a) A selected node. b) A pinboard of outgoing links. c) An incomplete link. d) An unnamed node.

from the rest of the graph. These efforts are to aid the author's identification of the story state.

It is also important to convey and highlight incomplete states to authors. In Novella, we use the Zeigarnik Effect to remind users of incomplete links without a destination by not coloring in the pins (Fig. 5c). As connected pins are filled and brightly colored, those that are not filled in stand out. Similarly, we distinguish nodes that are not named with placeholder text that is lighter than the normal font to signal an incomplete state (Fig. 5d), although node names are optional in Novella.

Challenges

To ensure Novella's accessibility and longevity, the project's complete source code is available on GitHub[6]. Releases will be subsequently available, accompanying the source code as development continues.

Steps have been taken to reduce the interruption of authoring flow and to diminish UI clutter to ensure that writers can maintain focus on authoring rather than UI management without hindrance. Content for nodes is edited in popovers that can be temporarily opened and easily dismissed without using too much screen real-estate, which contributes to a reduction in UI clutter. Popovers can be optionally detached so that their presence is maintained if the author prefers. Animations in the interface, in particular with popovers, have been scoped to reasonable times as to not disrupt the authoring experience. Hiding the Story Preview, Variable Editor, and Entity Editor windows, which are lesser used than the primary graph editing interface, also helps to reduce UI clutter.

Communication of story state has been crafted to better inform the author in various ways, many of which we have discussed above. Additionally, Novella has a trashing feature akin to an operating system's recycle bin. Most content within

[6] `https://github.com/KasumiL5x/novella`

the interface can be trashed before it is deleted. When items are trashed, they appear a color that is consistent with disabled UI elements, and most interactions are not available. This allows for visualization of removal without committing such changes instantly, and therefore the state communication must be clear.

The creative process an author takes is not linear; stories are not entered line-by-line without mistake and experimentation is integral to the authoring experience, especially with interactive storytelling where choices matter. To reduce cognitive friction and help authors maintain focus on writing, nodes need not be named as they do in Twine, but instead rely on unique identifiers hidden from the author. This means that nodes can be rapidly created, edited, and connected without having to manually book-keep the names of nodes. Additionally, undoing of actions is widely supported throughout our system, which helps to facilitate experimental works. Without such a feature, authors would have to rely on backups or even short-term memory, which would significantly limit and disrupt the authoring workflow and hinder experimenting.

4 Conclusion & Future Work

In this paper we have presented a categorized taxonomy of authoring tools for interactive narrative. We also identified methods of delivery as well as various interface design paradigms, and discussed the effect they have on the accessibility and UX of authoring tools. We then detailed at length challenges at the interactive fiction authoring tools research community faces, accompanied by an in-depth experiment. Concluding, we briefly presented our own authoring tool, Novella, and explained how the UX lessons we had learned were implemented in its interface design, and how it approaches the challenges we had identified.

In future work, we intend to further explore detailed analysis of the existing corpus of authoring tools to verify the observations made in this paper. Our own authoring tool will continue to be refined through further development and usability experiments.

References

1. HyperCard. Apple Computer, Inc. (1987)
2. Storyspace. Eastgate Systems, Inc. (1987)
3. Genarrator. Genarrator (2015), www.genarrator.org
4. articy:draft 3. articy Software GmbH & Co. KG (2017), www.nevigo.com
5. Fungus. Snozbot (2017), www.fungusgames.com
6. Inform 7. Community (2018), www.inform7.com
7. inklewriter. inkle Ltd. (2018), www.inklestudios.com
8. Quest. Community (2018), www.textadventures.co.uk
9. Ren'Py. Community (2018), www.renpy.org
10. Squiffy. Community (2018), www.textadventures.co.uk
11. Twine. Community (2018), www.twinery.org
12. Unity. Unity Technologies (2018), www.unity3d.com
13. Baio, A., McHatton, C.: Playfic (2018), www.playfic.com
14. Bernstein, M.: Collage, composites, construction. In: Proceedings of the fourteenth ACM conference on Hypertext and hypermedia. pp. 122–123. ACM (2003)
15. Cavazza, M., Pizzi, D., Charles, F., Vogt, T., André, E.: Emotional input for character-based interactive storytelling. In: Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems-Volume 1. pp. 313–320. International Foundation for Autonomous Agents and Multiagent Systems (2009)
16. Donikian, S., Portugal, J.N.: Writing interactive fiction scenarii with DraMachina. In: Technologies for Interactive Digital Storytelling and Entertainment. pp. 101–112. Lecture Notes in Computer Science, Springer (2004)
17. Glock, F., Junker, A., Kraus, M., Lehrian, C., Schäfer, A., Hoffmann, S., Spierling, U.: "office brawl": A conversational storytelling game and its creation process. In: Proceedings of the 8th International Conference on Advances in Computer Entertainment Technology. pp. 88:1–88:2. ACE '11, ACM (2011)
18. Green, D., Hargood, C., Charles, F., Jones, A.: Novella: A proposition for game-based storytelling. In: Narrative and Hypertext 2018. ACM (July 2018)
19. Guarneri, A., Ripamonti, L.A., Tissoni, F., Trubian, M., Maggiorini, D., Gadia, D.: GHOST: A GHOst STory-writer. In: Proceedings of the 12th Biannual Conference on Italian SIGCHI Chapter. pp. 24:1–24:9. CHItaly '17, ACM (2017)
20. Göbel, S., Salvatore, L., Konrad, R.: StoryTec: A digital storytelling platform for the authoring and experiencing of interactive and non-linear stories. In: 2008 International Conference on Automated Solutions for Cross Media Content and Multi-Channel Distribution. pp. 103–110 (2008)
21. Göbel, S., Schneider, O., Iurgel, I., Feix, A., Knöpfle, C., Rettig, A.: Virtual human: Storytelling and computer graphics for a virtual human platform. In: Technologies for Interactive Digital Storytelling and Entertainment. pp. 79–88. Lecture Notes in Computer Science, Springer, Berlin, Heidelberg (2004)
22. Hargood, C., Weal, M.J., Millard, D.E.: The storyplaces platform: Building a web-based locative hypertext system. In: Proceedings of the 29th ACM Conference on Hypertext and Social Media. HT '18, ACM (2018)
23. Kim, S., Moon, S., Han, S., Chan, J.: Programming the story: Interactive storytelling system. Informatica **35**(2) (2011)
24. Koenitz, H.: Extensible tools for practical experiments in IDN: The advanced stories authoring and presentation system. In: Proceedings of the 4th International Conference on Interactive Digital Storytelling. pp. 79–84. ICIDS'11, Springer (2011)

25. Kriegel, M., Aylett, R.: An authoring tool for an emergent narrative storytelling system. In: AAAI Fall, Symposium on Intelligent Narrative Technologies (2007)
26. Leinonen, J., Munroe, J.: TextureWriter (2018), `www.texturewriter.com`
27. Nielsen, J.: Enhancing the explanatory power of usability heuristics. In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems. pp. 152–158. CHI '94, ACM (1994)
28. Poulakos, S., Kapadia, M., Schüpfer, A., Zünd, F., Sumner, R.W., Gross, M.: Towards an accessible interface for story world building. In: Eleventh Artificial Intelligence and Interactive Digital Entertainment Conference (2015)
29. Roberts, M.: TADS (2013), `www.tads.org`
30. Thue, D., Bulitko, V., Spetch, M., Wasylishen, E.: Interactive storytelling: A player modelling approach. In: Proceedings of the Third AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment. pp. 43–48. AIIDE'07, AAAI Press (2007)
31. Ursu, M.F., Cook, J.J., Zsombori, V., Kegel, I.: A genre-independent approach to producing interactive screen media narratives (2007)
32. Zünd, F., Poulakos, S., Kapadia, M., Sumner, R.W.: Story version control and graphical visualization for collaborative story authoring. In: Proceedings of the 14th European Conference on Visual Media Production (CVMP 2017). pp. 10:1–10:10. CVMP 2017, ACM (2017)