University of the Sunshine Coast

Katryna Starks

Branched and parsed: The tools of interactive narrative writing

Abstract:

Interactive narrative presents an opportunity to create inviting experiences that challenge readers to become players within narrative rather than observers of it. It is an emerging form of narrative, with most research focusing on the content and structure of the narrative: the craft of writing interactive narratives, discussion on invention as applied to the narrative process, exploration of innovations in narrative, and suggestions for interactive narrative structure. However, interactive narrative requires more than writing skills; some knowledge of computer code is necessary as well, but there is little to bridge the gap between creative writing knowledge and coding practice to help creative writers become interactive narrative artists. There are several easy and free tools available to help creative writers practice non-linear writing and therefore enhance their skill level in potential game writing. These tools can also assist traditional writers in showcasing their works in non-traditional ways, progressing their writing technique and expanding their artistic repertoire. This article presents an overview of the interactive narrative technology available to writers, as well as commentary on their ease of use and experimental potential.

Biographical Note:

Katryna Starks is the Discipline Lead for the Serious Games program at the University of the Sunshine Coast. She designed the founding curriculum developer Interactive Narrative minor and currently teaches within the program. She is also completing her PhD as a member of University of the Sunshine Coast's ENGAGE lab. She holds a Master's degree in Psychology with a focus on how video games can foster health-promoting behaviours. Her current research foci include the effects of narrative within games on identification, self-efficacy and agency in adolescent and young adult females; as well as using choice-based narrative for mental wellbeing.

Keywords:

Creative Writing – Interactive narrative – Choose your own adventure – Choice – Interactive – Branching – Game – Technology

Introduction

Interactive narrative offers tremendous creative freedom for creative writers, allowing an author to create possible stories that are only realised when experienced. Stories which include choices, branches, and other interactive input differ in narrative structure from traditional stories, and most research focuses on these differences, such as: the craft of writing interactive narratives, discussion on invention as applied to the narrative process (Bold 1998), exploration of innovations in narrative (Brotchie 2005; Short 2000), and suggestions for interactive narrative structure (Brown et al. 2011). However, interactive narrative requires more than writing skills; some knowledge of computer code is also necessary. However, there is little to bridge the gap between creative writing knowledge and coding practice to help creative writers become interactive narrative artists. This results in many interactive narratives written by the programmers who created the authoring tools, and the few writers who have mastered them (Koenitz 2014; Spierling et al. 2009; Ryan et al. 2015). This paper draws on the work of Hartmut Koenitz (Koenitz 2014), who proposed five components of interactive narrative practice that would assist creative writers in working with the interactive narrative authoring systems. He proposed a new narratology that properly described interactive narrative, as well as authoring systems that provided interoperability and sustainability, and were both author and user focused. Interoperability refers to the ability for the authoring system to work with other software. Sustainability refers to the ability for narratives created by the authoring system to remain accessible considering periodic changes in technology. Author-centred and user experience indicate whether the authoring system itself and the narratives it produces are easily operated by practitioners and players with little technical experience. This paper provides a few introductory terms that are essential for interactive narrative authorship, and provides analysis of three authoring systems in accordance with the Koenitz (2014) proposals.

Types of interactive stories

Authors may choose to use interactivity to enhance the diegetic experience by allowing the player to direct the narrative; however, the system producing the narrative may promote extradiegetic experiences which remove the player from the world. This dual engagement with both the story itself and the machinery that enables it provides two simultaneous types of immersion. Narrative immersion, in which the player is immersed in the story itself, and mechanical immersion, in which the player is immersed in the software system that creates the story (Mason, 2013).

Parser-based

There are two basic types of interactive narratives: parser-based and branching. Parser-based stories, or interactive fiction, involve the reader typing in specific commands that represent movement or action. These stories require careful readings of location and object descriptions to discern clues about what commands to give to move the story along. For instance, a room description might include a fireplace along with exits to the north, south and east. The reader can type ‘examine fireplace’ and possibly discover a

note among the ashes. The player then types 'read note', etc. Should the player type a direction such as 'south', they will move in that direction and find themselves in a different room.

Parser-based stories are highly interactive and provoke thought by having the reader think of commands rather than being presented with them; however, these experiences must be designed in a way that leads the player to type the exact word combination that will move the story forward. For instance, a parsed narrative may include a character who talks about having been pregnant. For the player, that is a clue to ask about a child, or perhaps a son or a daughter. However, the author may have only programmed the game system to respond to the word 'baby'. In this case, the player must think of the word for child that the author would use in order to continue the story. Typing the wrong word and receiving an error message moves the player from a diegetic to an extradiegetic experience within the story, and forces them to engage mechanically by re-typing words until they reach an acceptable one, thereby potentially lessening narrative immersion.

Similarly, it is important to understand the basic conventions of verb use within interactive fiction to be consistent with other works. Most parser-based stories use similar verbs such as examine, look, take, wear, etc. There are also a common set of movements, specifically compass movements (north, south, east, west, northeast, southeast, up, down, etc.). Authoring tools for parser-based stories contain standard libraries of these common commands, so the author doesn't need to define them to use them.

Branching

Branching stories have a narrative which proceeds according to reader choice. Some choices are action based involving options for a location or battle tactic, while others are relationship-based and affect how characters interact with the protagonist later in the game. Some relationship choices are one time encounters involving a choice to engage (or not) with another character, while others involve a series of encounters that result in an alliance or conflict.

Branches provide interactivity and a feeling of control for the reader, while inherently providing a reason for replay as every reader is aware of the roads not taken and will play it again to make different choices. Choices are presented as a set of hyperlinks, so players will not need to guess the author-preferred vocabulary. However, they provide both more opportunity and more work for authors, as each story must essentially contain multiple stories. Branching stories also present organisational challenges, as each possible set of choices must converge into a coherent story path.

The hyperlinked choice structure allows for more inherent narrative immersion than parser-based stories; however, if players become bored with the text, they can move into mechanical immersion by simply skipping to the choices and clicking randomly. With both types of interactive narrative, authors must balance narrative pacing with considerations of screen size, as scrolling is also a mechanical action that can lessen narrative immersion.

Essentials of Coding for Interactive Narratives

Though not suggested by Koenitz (2014), the vocabulary of practice for interactive narrative should include a few terms from information technology. Specifically, programming terms that are necessary for understanding authoring software. While authoring tools are based on various programming languages, the concepts of 'if statements' and 'variables' are the core of interactivity in any system, and are essential for interactive narrative authors to understand.

If statements

'If' statements are the equivalent of the phrase 'if this, then that'. While coding or using an authoring system, some 'if' statements are implicit. For instance, a linked choice for a branching narrative is the implicit 'if' statement: 'if the player clicks this, then they will (read this, go to a location, look at an object, etc.)'. However, some 'if' statements must be explicitly written, such as 'if the player has a red key while trying to open the red lock, then the lock opens successfully'. 'If' statements can be used on their own or combined with 'else' statements, which provide instructions for the failure of an 'if' statement. In the red lock example, the 'else' statement 'else a puff of poisons springs from the lock and the player loses a life' can be added to give the player a penalty for trying to open the red lock without the appropriate key.

Variables

Variables are used as placeholders for when the player needs to add information to the game. For instance, a game might refer to the player by name. Because each player has a different name, the player's name cannot be included in the game programming. Instead, the placeholder variable '$name = you' might be added as a default so that the player who chooses not to enter a name will be referred to as 'you'. However, once a player enters a name, the variable changes (e.g. '$name = Morgan').

State-tracking variables are used to keep track of background information such as whether objects are opened, closed, or held by the player; as well as the status of player relationships, weapons and armour, player health, and game scores.

Word-based variables, such as names, are called string variables, while number-based variables are called numeric. However, numeric variables are numbers that will be used in arithmetic, such as scores that will increase throughout the game. Numbers in addresses, telephone numbers, etc. are string variables. It is important to understand how variables are handled within the coding language or authoring system used for a particular game. A common way of handling each type of variable is to put string variables in quotes while numeric variables are without quotes.

Extras

For some authors, the addition of graphics and/or sound is essential to the narrative. However, not all engines provide graphic and sound capabilities. Engine features should be reviewed carefully before choosing one, to make sure it has the proper specifications for each project.

Creating interactive narratives: Languages and authoring systems

Some computer programming is required for creating interactive narratives; however, the languages used are often simplified and user-friendly, designed to appeal to writers. Similarly, authoring systems remove some programming tasks, enabling non-programmers to create complex tasks with little programming skill. Some interactive narrative authors never use programming except when building narratives, while others move deeper into programming, learning languages like JavaScript or more advanced game engines such as Unity.

This section contains reviews and code examples for three popular interactive narrative engines: Twine (Klimas 2009) , Ren'Py (Rothamel 2004) and Quest (Warren 2006). All of them are free at this writing. A few have online versions as well as downloadable files. While these authoring systems are established, the owners can change or the original programmer can become too busy to maintain the program. For this reason, it is recommended to download a version of preferred programs to preserve the ability to edit stories.

In addition, each system will be analysed for the four proposed requirements presented by Koenitz (2014): interoperability, sustainability, author focus, and user experience. For ease of comparison, the same story will be used for each programming example. The premise is a woman running through the woods, who runs into a dwelling to escape pursuing wolves. In the dwelling, she must complete a simple maze in order to get out. Italicised sections represent programming language that is used in the authoring system.

Twine

Twine creates branching narratives with a very simple flowchart-like interface. The language used is Twine specific, but the output is a web page with HTML, CSS and Javascript. It is one of the easiest tools to use, with authors as young as eight. Though it is quite simple to start, it can be extended to create elaborate narrative games with graphics, sound, codes, and other types of puzzles. If an author knows HTML, CSS or Javascript, they can extend Twine far beyond its inherent capabilities. There is plenty of documentation and a vibrant author community which provides assistance with basic tasks, programming examples, and templates.

Twine is available at http://twinery.org/ and includes both an online version that saves stories within the browser as well as downloadable versions for Windows, Mac, and Linux. Though the Twine download is an .exe file, it extracts and runs from a single folder, meaning the entire program can be copied and run from a portable stick drive. As internet standards are universal, the online presence of the authoring system fulfils

the interoperability component of the Koenitz (2014) recommendations, and the HTML output fulfils the sustainability component, as web pages remain accessible indefinitely.

Creating in Twine involves linking boxes, called passages. The names of the passages are used for links, and links from passage to passage are visually represented as arrows, creating a flowchart-like structure for the narrative. Though Twine workspaces can get messy, passages can be dragged around so that thematically linked passages are near each other.

To begin a story in Twine, the user must double-click the first passage in a story and start typing. Narrative text is typed plainly, while hyperlinks are enclosed within [[double square brackets.]] The text inside the brackets becomes the name of the next passage. When the author closes the current passage, a new one is automatically created and linked. An example is given in Figure 1.

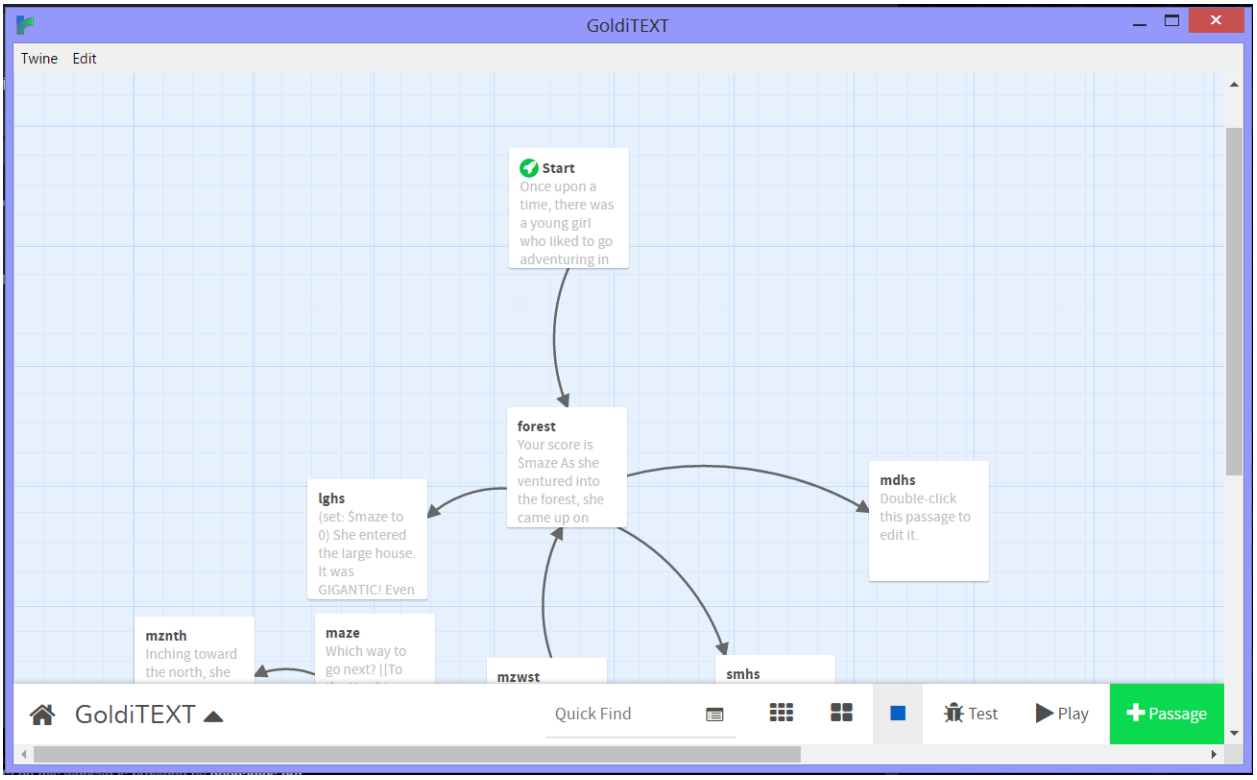

Figure 1: Twine editor screen with connected passages

Passage Name: Start

The wolf sounds faded as she tumbled through the [[forest]].

Passage Name: forest

The trees opened into a clearing. All was dark except three trees, each with a glowing symbol on its trunk. One press of her hand, and she would be home. But which one?

[[The scorpion ->scrp]]

[[The archer -> arch]]

[[The lion -> lion]]

The second set of links looks different to the first. In the first case, the word 'forest' is in brackets and it becomes the title of the next passage. In the second set of links, the reader sees the text on the left, but the passage name becomes the text on the right. In this way, authors can create abbreviated passage names for coding while maintaining the narrative experience for the reader.

In the next section, the player will enter the large house, which consists of a maze. For simplicity, there are only four directions (north, south, east and west) and the only correct way is west.

This section also introduces two elements: variables and mazes. To get out of the maze, the player has to go west four times. The variable $maze is set on the 'scrp' passage so that the maze counter starts at zero. This means that every time the player enters the large house, the maze starts over. Furthermore, since the only correct way is west, there is no need to provide a custom set of directions for each path. A quick setup with one passage for all directions will do. For that, the passage 'maze' was created, and four directions were added. Because the basic compass directions may be used later in the game, the directions for the maze are abbreviated and combined with an abbreviation for 'maze'. Finally, the (display) macro is used to redisplay the 'maze' passage at the bottom of each passage within the maze, enabling the protagonist to choose a different direction. The actual code for each passage is:

Passage Name: scrp

(set: $maze to 0)

As she pressed the scorpion, the tree opened, depositing her into its base.

(display:'maze')

Passage Name: maze

She felt around the walls, deciding on a direction.

[[North ->mznth]]

[[South ->mzsth]]

[[East ->mzest]]

[[West->mzwst]]

Now that the basic maze is set up, each direction needs its own response. The west maze direction, mzwst, will do two things. One is to move the player in the right direction, and the other is to increase the $maze variable by one. When the $maze variable reaches four, the player will escape the maze. The escape is accomplished with an 'if' statement, designed so that if the $maze variable is greater than or equal to four, then the player has reached the end of the maze, otherwise do something else. Though there are only four fixed lines with no random directions, if a reader clicks the direction links in random order, this will appear to be a quite complicated maze.

Passage Name: mznth

Inching toward the north, she felt an intense heat.

She backed away and decided to go in a different direction.

(display:'maze')

Passage Name: mzsth

Inching toward the south, she felt an intense shock.

She backed away and decided to go in a different direction.

(display:'maze')

Passage Name: mzest

Inching toward the east, she heard an intense (either: 'growl', 'shriek', 'roar').

She backed away and decided to go in a different direction.

(display:'maze')

Passage Name: mzwest

{(set: $maze to $maze+1)

(if:$maze>=4)[Inching her way to the west, her hand grasped a branch.

With a quick twist, she was back in the lush green of the [[forest]].]

}

(else:)[Inching her way to the west, she felt ... nothing.

She walked quickly down a long corridor before reaching a dead end.

(display: 'maze')]

The user can test play the game by clicking the play button on the bottom right and going into the large house and through the maze. Upon finishing the maze, entering the large house should reset it, requiring the player to move west four times again to get out. Note the different wording on the 'mzest' passage. The 'either' macro randomizes responses so that each time the player goes East, she might encounter a different sound.

Twine creates web page files that can be shared with a simple upload. Free websites like GameJolt (Decarmine 2002) (https://gamejolt.com/) and Itch.io (Corcoran 2013) (https://itch.io/) allow authors to create online portfolios of their works, while simultaneously publicising them to a built-in audience. Though there are many free games available, these sites are online stores, allowing authors to build an audience via experimental free games and then move to paid offerings without changing websites.

The visual interface in Twine, as well as the simplified programming language consisting mostly of hyperlinks, provide an author-centred experience. Twine was

made for writers rather than programmers. However, it remains flexible enough so that writers can learn more complicated programming concepts which result in more varied levels of interaction. One caveat is that the authoring system can become cluttered if a story gets very large. Passage boxes can be moved around and arranged into story groups, but this is done manually and a lot of scrolling is required to see the entire story.

Similarly, the pre-styled web page output and hyperlinks create a familiar user experience. For most users, instructions for gameplay are not required, as the hyperlinks glow with the familiar blue underlined text that users encounter on other web pages. They simply read and click.

Ren'Py

Ren'Py is a visual novel engine that uses a simplified version of Python. The base engine includes quite a few customisable features, and its capabilities can be extended with Python. The language itself is quite simple, and the engine includes an extensive interactive tutorial as well as a sample game that contains simple code to review. Unlike other engines, Ren'Py is not used to create or edit the code itself, but rather to organise files, launch test games for review, and compile the final game for distribution. Ren'Py is available at https://renpy.org/.

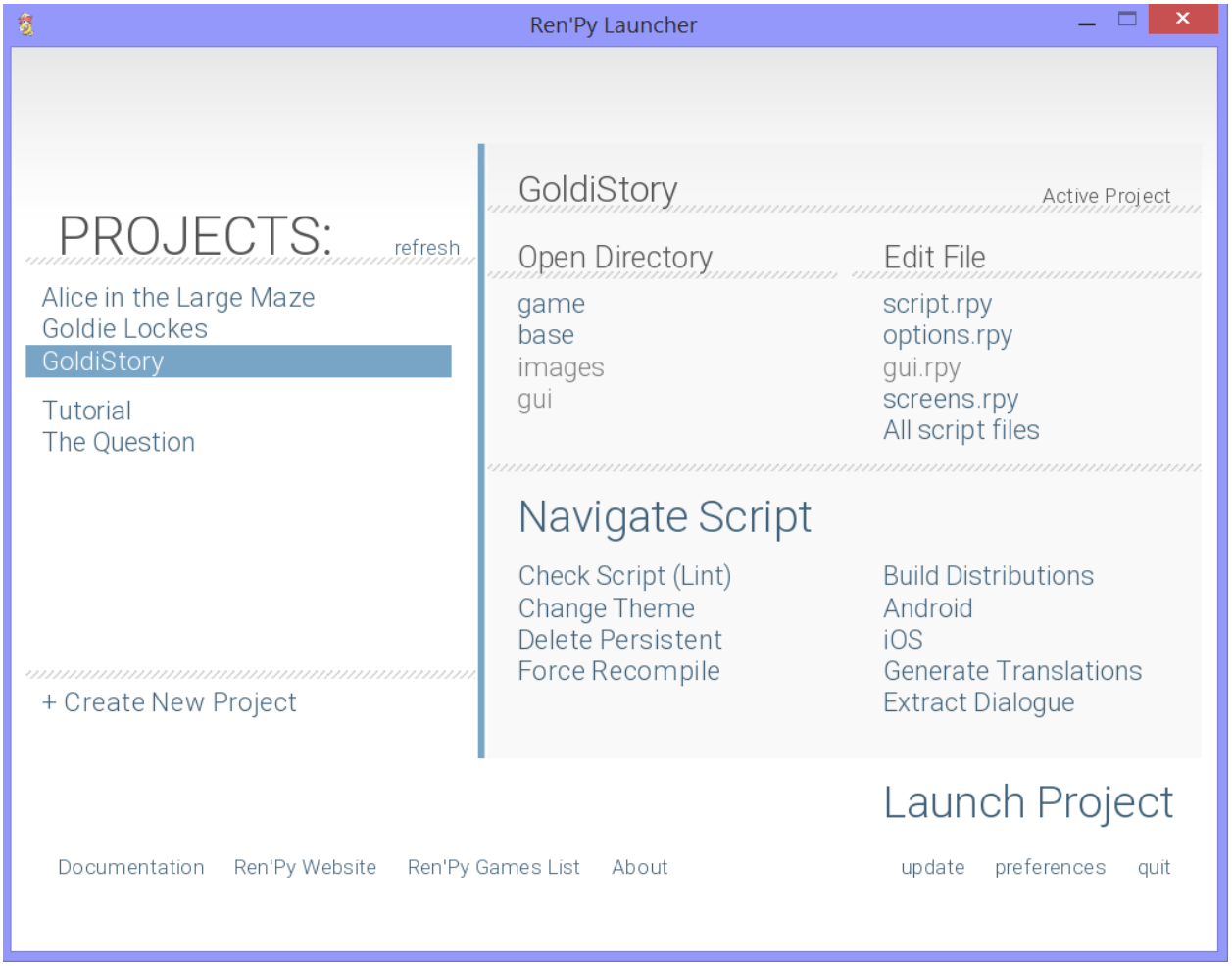

Figure 2: Renpy project launcher screen

New Ren'Py projects contain template files, including a 'Script.rpy' file which needs to be edited to create a custom story. Ren'Py requires a separate editor which is either included or will download upon request. Editra is the preferred editor and the one tested for this article. Unlike the other authoring systems, Ren'Py outputs a game that contains text at the bottom of the screen and a transparent pane at the top. For this reason, Ren'Py should only be used to create games with visuals.

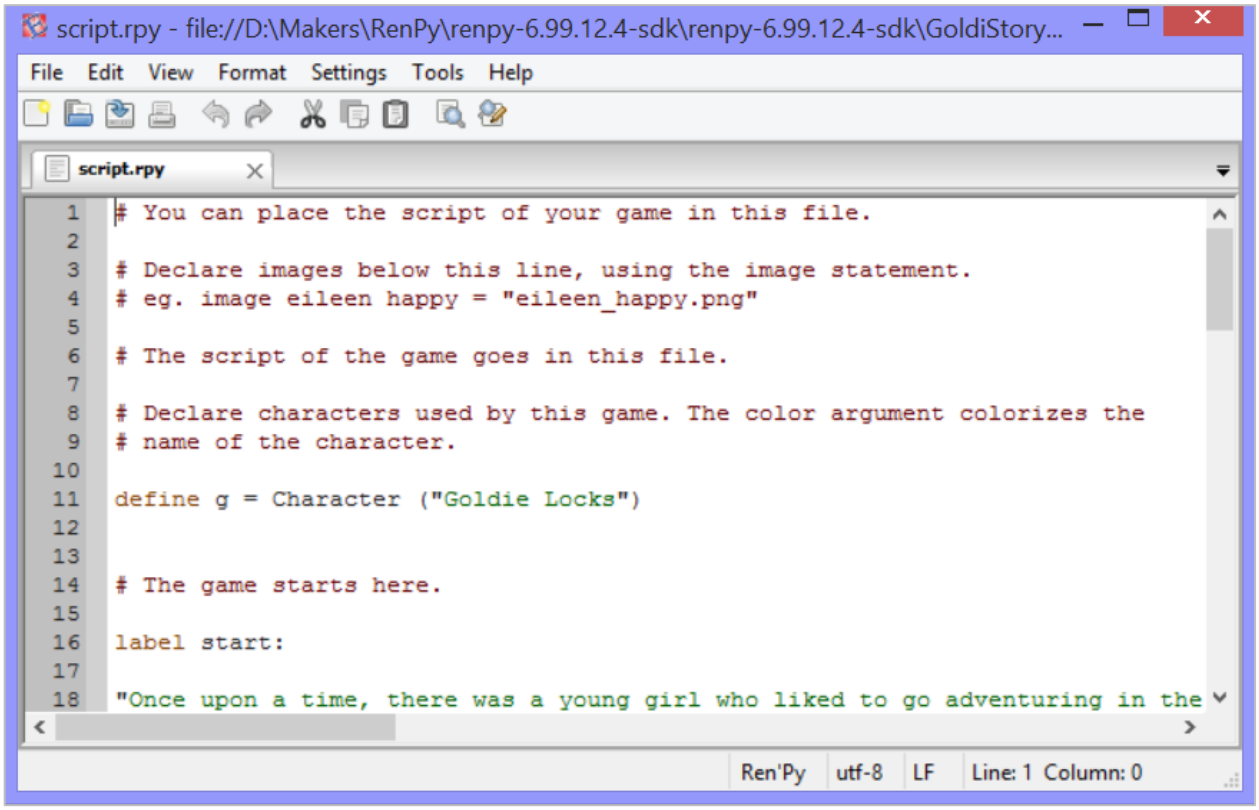

Figure 3: Editra screen with Ren'Py code

Several Ren'Py games of professional quality have been created and are available on the Steam game platform. One of the most notable is Magical Diary (Hanoko Games 2011).

Ren'py uses labels to keep track of story sections. The first label in a story is called 'start'. Choice sections are designated with the label 'menu'. Once in the menu label, the choice itself is declared, followed by a 'jump' statement which tells Ren'py which label continues the story after the choice is made.

To add images and characters to Ren'Py, they must be declared before the first label of the game. Once declared, they can be repeated by name in order to call them. However, character names can be long and cumbersome to retype, so Ren'Py features an abbreviated naming system, whereas a short name can replace a longer one. For example, instead of typing the character's name before each line of dialogue she says, we can declare 'g' as a designation for the name by defining it with the line 'define g = Character ('the character's name')'.

define g = Character ('Character name goes here')

 # The game starts here.

label start:

'The wolf sounds faded as she tumbled through the forest near her home.'

label forest:

'The trees opened into a clearing. All was dark except three trees, each with a glowing symbol on its trunk. One press of her hand, and she would be home. But which one?'

g 'I ... can't ... remember'

menu:

'The scorpion':

jump scrp

'The archer':

jump arch

'The lion':

jump lion

label scrp:

$ maze = 0

'As she pressed the scorpion, the tree opened, depositing her into its base.

jump twisty

label twisty:

g 'She felt around the walls, pondering a direction.'

menu:

'North':

'Inching toward the north, she felt an intense heat. She backed away and decided to go in a different direction.'

jump twisty

'South':

'Inching toward the south, she felt an intense shock. She backed away and decided to go in a

different direction.'

jump twisty

'East':

'Inching toward the east, she heard an intense growl. She backed away and decided to go in a different direction.'

jump twisty

'West':

if maze < 3:

$ maze = maze+1

'Inching her way to the west, she felt ... nothing. She walked quickly down a long corridor before reaching a dead end.'

jump twisty

else:

'Inching her way to the west, her hand grasped a branch. With a quick twist of the branch, she was back in the lush green of the forest.'

jump forest

return

Ren'Py creates downloadable games that play on several major platforms including Windows, Mac, and Linux as well as Android and iOS. While the games won't play in a browser, they can be uploaded to game websites and offered for sale or free download to players.

Ren'py uses a simplified form of Python, an established language in the programming world. This possibly fulfils the interoperability component of the Koenitz recommendations, as it can be extended with proper Python programming. However, it creates files that may stop working properly when computers are inevitably updated. This reduces sustainability. This may be mitigated if authored files can be re-launched with new versions of Ren'py when the authoring system itself is updated.

The author interface can be confusing, as it opens a third-party editor. The language itself may be daunting to authors unfamiliar with programming, as the paragraph indent

levels in the examples are necessary for narratives to run properly. Furthermore, authors may not understand the error messages resulting from improper formatting, making it difficult to fix programming errors.

Conversely, when programmed properly, Ren'py offers one of the most satisfying user experiences. The requirement for graphical stories, and the plethora of theme designs make Ren'py stories look beautiful and professional. Hyperlinked choices are stylised buttons that allow the player to forget the web-like interface. Ren'py also features a layering system that allows authors to include different versions of characters, so they can change facial expressions while remaining in the same place. This adds to the narrative immersion possible with Ren'py stories.

Quest

Quest is different from the other engines in that it creates parser-based games as well as gamebooks. The other difference is that it is a tab-based system rather than a coding language. Quest comes with a library of basic tasks, such as 'directions', 'take', 'examine', etc., so the user doesn't need to code them. Quest also has a few simple features such as an automatic map that tracks the user, as well as a built-in inventory list. One caveat for using Quest is that it outputs special files that can only be run with the Quest program. Quest games can be hosted on the Quest website, http://textadventures.co.uk/, which also contains an online version of the Quest engine.

To make a game in Quest, the user starts by focusing on the navigation menu on the left of the screen. The first room of the game is there by default and is simply called 'room'. For our example, this will be renamed 'start' using the 'setup' tab on the left. The room description (the text for the game) will be enter on the 'room'. For efficiency, it's advisable to create the rest of the game locations and descriptions after setting up the first room. The '+Room' button on the top menu should be used to add: forest, large house, maze, mzNth, mzSth, mzEst, and mzWst. After adding all of the rooms, go back to the 'start' room and use the 'exits' tab to create an exit to 'forest' (any direction). If you want to keep the story moving forward, don't enable another exit back. Create exits for forest-to-large house, then create an 'in' exit for large house-to-maze. On the 'maze' room, create north/south/west/east exits as appropriate for the maze rooms.

There will be no exits added to the maze rooms. Instead, we will set a script. Go to the 'scripts' tab and add a script in the section 'After entering the room:'. In the script box, click the 'move' button at the top. Use the dropdowns to select to move object player to object maze. This will cause Quest to enter the room, show the room description (She inched north …) and then move the player back to the 'maze' room, displaying that description (Which way should she go?) and the directions so she can choose another.

For mzWst, a few more elaborate scripts are needed. Firstly, there is the variable setup in the forest that sets the maze to 0. In Quest, there are two types of variables. An attribute is a persistent variable that stays active throughout the game. A variable is a temporary variable that only exists in the section of code it's written in. Since we want the maze variables to be relevant throughout the game, we'll use attributes. An attribute is a variable assigned to an object while a variable is on its own. For Quest, that means

an attribute for maze is game.maze rather than a variable, which would be maze alone. For the 'forest' room, the script to add is 'set variable game.maze = expression 0'. The 'set a variable or attribute' command is in the 'variables' section of the add screen. The terms 'game.maze' and the number 0 must be typed.

For the mzWst room, we first want to check to see if the end of the maze has been reached. Set the following command in the 'Before entering the room' section. The command is 'if' with dropdown expression, then type 'game.maze >3'. Add a new script directly after that, which uses the 'print' function with 'msg' and shows the end of the maze (where the door knob is found). Finally, use the 'move' script to move the player back to the forest, in front of the three houses. The 'After entering the room' section will 'set variable game.maze to game.maze +1' as well as move the player to the beginning of the maze. Since this appears in the 'after entering a room' section, the description text for west (she inched west …) will appear before the player is transported back to the maze. A graphical depiction of mzWest appears below.

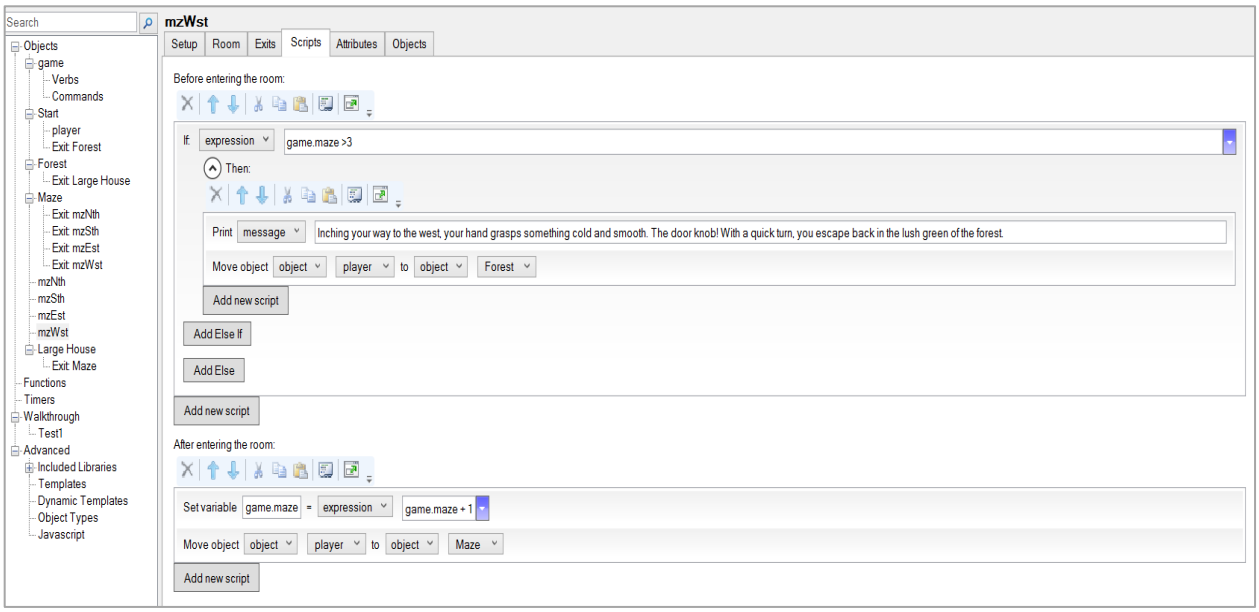

Figure 4: Quest screen with code for mzWst direction

Quest files must be played either in the Quest development software or on the Quest website. While it is possible to offer Quest games via other websites, it would need to be clear that other software is required to play.

While Quest provides an almost completely graphical authoring interface, it uses terminology that programmers are familiar with, but creative writers may not be, such as the term 'attribute' for certain types of variables. The naming convention, such as 'game.maze' may also be unfamiliar to creative writers, and can complicate the authoring experience. The proprietary interface prevents interoperability, and the fact that Quest stories must be hosted on the Quest website hinders sustainability, as all stories would become inaccessible if the Quest website ceased to operate.

Quest creates clean player files with helpful features such as a map and arrow buttons for directions; however, the parser-based narrative structure can create problems for the player if the game is written with confusing language and the player is hindered from

understanding the correct word to type for story progression. Quest does create the option of using hyperlinks in lieu of parsed text input; however, there are other programs with better author and player experiences which provide hyperlinked stories.

Conclusion

This paper presented a brief survey of research involving interactive narrative content and structure, as well as Koenitz's (2014) recommendations for a unified interactive narrative vocabulary and authoring systems that invite creative writers into interactive narrative creation. To this end, the paper included a brief explanation of two basic narrative styles, as well as two basic programming concepts. Furthermore, three interactive narrative authoring systems were presented, along with the features and code examples for each. Lastly, each system was analysed according to the Koenitz (2014) recommendations for authoring system interfaces.

Engine	Interactive Narrative Vocabulary	Interoperability	Sustainability	Author-Centred	User Experience
Twine	Uses common programming terms for links and variables; and familiar terms for narrative (passages)	HTML, CSS, and Javascript are standard programming languages which can be used to extend authoring system capabilities and/or interface with other tools	Creates HTML files which can be hosted easily and accessible indefinitely.	Easy author interface with visual boxes and arrows to help author keep track of story direction as well as easily created hyperlinks. Can become cumbersome with large stories.	Easy to play. Simple interface with hyperlinks. Looks like a web page.
Ren'Py	Uses terms such as label and jump, which are neither familiar within creative writing practice nor programming.	Based on Python, which is a common programming language that may allow for interface.	Creates Windows (.exe), Android, and iOS files which are prone to become outdated with computer updates.	Indentation is required, which may be confusing for creative writing authors. Graphics are required, which presents a deeper learning requirement for authors who	Creates beautiful, seamless, professionally presented games for a satisfying user experience.

				normally rely on text.	
Quest	Uses programmer-centric vocabulary which would likely be unfamiliar to creative writers.	Proprietary system which is closed and not available for interoperability.	Creates a '.quest' file (specialized) which is only able to be hosted on its own servers, presenting a risk to sustainability.	Visual authoring interface; however, locations and terms can be confusing. Much more programmer-friendly than author-friendly.	Pleasant user interface with helpful features. Nature of parser-based games may present a challenge to narrative immersion.

Figure 5: Narrative engine comparison table

Of the authoring systems analysed, Twine appears to be the system that meets the most the Koenitz (2014) recommendations for interactive narrative authoring systems. While there is no standard vocabulary for interactive narrative, Twine uses terms that are established within the programming industry and internet users, as well as a few terms used in standard creative writing practice. Its basis in common internet standards provides opportunity for interoperability. The webpage as an output provides the potential for sustainability. The visual authoring system with simplified hyperlinks is author-centred, and the output of a simply styled web page is easy for players to use.

Works cited

Bold, Stephen C 1998 ‘Labyrinths of invention from the new novel to OuLiPo.’ *Neophilologus* (82) 4, 543-557

Brotchie, Alastair *Oulipo compendium* Harry Mathews (ed) London: Atlas Press

Brown, Neil CM, Barker, Timothy S, and Del Favero, Dennis 2011 ‘Performing digital aesthetics: The framework for a theory of the formation of interactive narratives’ *Leonardo* 44 (3), 212-219

Corcoran, Leaf 2013 ‘Itch.io’, at https://itch.io/ (accessed 16 September 2017)

DeCarmine, David and Yaprak 2002 ‘GameJolt’, at https://gamejolt.com/ (accessed 3 September 2017)

Hanako Games and Spiky Caterpillar 2011 ‘Magical Diary’, at http://hanakogames.com/magical_diary.shtml (accessed 19 September 2017)

Klimas, Chris 2009 ‘Twine (Software)’, at http://twinery.org/ (accessed 30 September 2017)

Koenitz, Hartmut 2014 ‘Five theses for interactive digital narrative’ *International Conference on Interactive Digital Storytelling* Cham: Springer, 134-139

Mason, S 2013 ‘On Games and Links: Extending the Vocabulary of Agency and Immersion in Interactive Narratives’ *ICIDS*, 25-34

Rothamel, Tom 2004 ‘Ren’Py Visual Novel Engine’, at https://renpy.org/ (accessed 24 September 2017)

Ryan, JO, Mateas, M, and Wardrip-Fruin, N 2015 ‘Open design challenges for interactive emergent narrative’ *International Conference on Interactive Digital Storytelling* Cham: Springer, 14-16

Short, Emily 2000 ‘Galatea’ *Electronic Literature Collection: Volume One* 1

Spierling, Ulrike, and Nicolas Szilas 2009 ‘Authoring issues beyond tools’ *Joint International Conference on Interactive Digital Storytelling* Berlin: Springer

Valve Corporation 2003 ‘Steam (Software)’, at http://store.steampowered.com/ (accessed 28 September 2017)

Warren, Alex 2006 ‘Quest (Software)’, at http://textadventures.co.uk/ (accessed 5 September 2017)