Available online at www.sciencedirect.com

ScienceDirect

Computers and Composition 37 (2015) 31–43

Computers and Composition

www.elsevier.com/locate/compcom

Textual Adventures: Writing and Game Development in the Undergraduate Classroom

Brian D. Ballentine[*]

Coordinator, Professional Writing & Editing, Department of English, 1503 University Ave. West Virginia University, Morgantown, WV 26506

Abstract

Building on scholarship in video game studies, this article argues that rather than writing about games as textual artifacts or playing games to reveal what they can teach us about writing, we should design curricula that introduce students to all of the writing that supports a successful game. The broader claim is that good writing leads to good software. In this case the software in question just happens to be a video game. The software and information technology industries value skilled writers, and those involved with writing pedagogy should continue to explore new methods for preparing students for these opportunities. Building on existing narratological approaches to teaching games is recommended as a place to begin. This article documents a specific endeavor to teach writing through software and game development. The article demonstrates how students experience the entire software development life cycle and its many writing challenges while developing their own text-based adventure games from the ground up.
© 2015 Elsevier Inc. All rights reserved.

Keywords: Video games; Writing pedagogy; Development; Software; Narrative; Professional writing; Technical writing

Prior to completing his Ph.D. at Case Western Reserve University, Brian Ballentine was a senior software engineer for Marconi Medical and then Philips Medical Systems designing user-interfaces for web-based radiology applications and specializing in human computer interaction. This past work experience ties to his current research interests that include open source software, technical and professional writing, digital humanities, and intellectual property. Ballentine is currently an associate professor and coordinator for the Professional Writing and Editing program at West Virginia University.

For researchers who pursue video games, 2008 was a banner year. The journals *Computers and Composition* and *Computers and Composition Online* published special issues dedicated to "Reading Games: Composition, Literacy, and Video Gaming." Likewise, the journal *Technical Communication* released a special issue on "3D Virtual Worlds and Technical Communication." Most of the authors in those pieces noted milestones in the broadening field of game studies including work tying games to learning and literacy (Gee, 2003), work demonstrating the rhetorical aspects of games and how they construct arguments (Bogost, 2007), work exploring ludology and the study of gameplay (Juul, 2005), and work positioning games in a larger complex of transmedia storytelling (Jenkins, 2006). And, scholars in the humanities continued with important research theorizing video games (Perron & Wolf, 2009) as well as why they "matter" culturally (Bissell, 2011) and how they can continue helping us learn (Squire, 2011). Humanities scholars, in

[*] Corresponding author. Tel.: +304 293 9704.
E-mail address: brian.ballentine@mail.wvu.edu

http://dx.doi.org/10.1016/j.compcom.2015.06.003
8755-4615/© 2015 Elsevier Inc. All rights reserved.

particular, have mined games and game play for what they reveal about race, class, gender, identity, and the body (Burn, 2006; Carr, 2006; Gregersen & Torben, 2009). Conferences such as the Association of Teachers of Technical Writing, the Conference on College Composition and Communication, and Computers and Writing have seen a consistent number of panels dedicated to games each year. Indeed, scholarly work on games and game theory continues to be published in collections and major journals related to rhetoric and composition as well as professional and technical communication.

As for the future trajectory of games, in a 2011 piece in *Computers and Composition* titled, “Computers and Composition 20/20: A Conversation Piece, or What Some Very Smart People Have to Say about the Future,” Douglas Eyman wrote:

> Two locations of writing that I think will become more prevalent and more important to our development of writing pedagogies are on mobile devices and within virtual environments (including digital games). Indeed, these two locations may well intersect, as the practices and literacies of gaming move fluidly from the virtual space of the game to the physical realm. (Walker et al., p. 329)

I, too, believe that more can be done with games in terms of pedagogical advancements for writing curricula, and the remainder of this article discusses the design, implementation, and the outcomes from teaching an undergraduate writing course dedicated to video game development. The course was designed to specifically target connections between writing, video games, and software development broadly defined. Teaching video game development at the undergraduate level was at least partially inspired (and justified) by an observation Eyman made in the 2008 special issue of *Technical Communication* dedicated to virtual worlds:

> games are all constructed systems that go through the same development cycles and business processes that any computer application requires; and as such, they provide the same opportunities for technical communicators to provide the skills and expertise that they bring to any systems development project. (p. 243)

While the bulk of Eyman’s article was dedicated to fleshing out a “theoretical model that links multiplayer game ecologies to the work and concerns of technical communicators” (2008, p. 246), my focus is explicating a course that capitalized on Eyman’s systems observation and exposed students to the entire software development cycle as well as all of the writing activities that are tied to that cycle. In this case, the software in question just happened to be a video game. The connection between the development cycle and writing is important not just because the software industry employs a great number of technical communicators (Lanier, 2009) but because I wanted students to leave the course with the larger understanding that good writing leads to good software. That is, I was less concerned about the possibility that technical communicators “can write themselves into the game industry” and more interested in using games to model development broadly defined (Eyman, 2008, p. 246).

The qualifier “broadly defined” appended to the software development life cycle is necessary because there is not one single approach to the development process, and as I remind my students, no two development environments and their resources will be the same. Development processes may be driven by a particular workplace or corporate culture, but how teams of developers approach building software is influenced heavily by a team’s project manager and his or her development philosophy. Borrowing from Alice Robison’s (2008) ethnographic scholarship and her findings revealing how “the principles of video game design are surprisingly similar to the principles of creating any successful curriculum” (p. 362), this article demonstrates how a writing instructor may position him or herself as both “a teacher and a manager” (p. 364) in a course dedicated to writing and game development. This article will proceed by providing an overview of the course, a discussion of the development cycle, and an examination of the text-based adventure development software selected for the course, as well as offer details on how student writing and presentation requirements were tied to game development. The article concludes with observations on the course outcomes.

1. The undergraduate gaming course

The formal title for the gaming course was “Narrative and Video Game Design.” The course was offered for the first time in the spring 2012 semester as part of my Department of English’s Professional Writing and Editing program. As I discussed with my 22 person class, I am not at all suggesting that by exploring narratological features at play in video games that the course was doing anything new. There is a long history of scholarly debate on connections and influences among and between narrative and new media such as video games (Aarseth, 1997; Manovich, 2001; Meadows, 2003; Ryan, 2004). As Eyman and others note, narrative is now understood as a "starting point" for approaching games (2008, p. 246). However, for an undergraduate course that assumed no prior experience with games or their development, a comfortable place to begin was much needed.

Briefly, narrative theory wrestles with the ways in which a particular medium communicates events, characters, settings, and perspectives, and the particular medium that concerned our course was the video game. I have discussed the potential benefits of employing narrative while composing software requirements and design documents elsewhere, and I wanted students to become familiar with some basic tenets of narrative theory (Ballentine, 2010). Our readings early in the semester included chapters from H. Porter Abbott's (2008) widely referenced introduction to narratology, so the students had some common vocabulary. And while we were all ready to share how our favorite games conveyed what we believed to be compelling stories or what Abbott would call a "series of events" (2008, p. 13) I focused the class on the larger goal of developing their own games. That is, I wanted to keep the class focused on both how our own creative narratives would be embedded in our game's play as well as seeing narrative's role in the development processes of our software. These pursuits are quite different from performing a critical analysis of a game to reveal its narrative structures. Our employment of narrative was much more in line with *developing* specifications for commercial software than it was, for example, with trying to theorize how elements of C. S. Lewis's *Through the Looking-Glass* were present in *American McGee's Alice*. This is not to say that such discussions don't have merit. However, given the confines of a single semester and the mission to expose students to a start-to-finish development experience, the course made brief time for the many debates regarding narrative and narratology (see Bal, 1997; Mitchell, 1980; Prince, 1982).

2. Software development life cycles

Different companies and their development teams use different methods and even language to talk about their approaches to software and what is often referred to generically as the software development life cycle (SDLC). Agile development, waterfall, spiral, and rapid prototyping are well-known specific examples for approaching development. Discussions on development can be found in the Institute of Electrical and Electronics Engineers's (IEEE) affiliate journals and conference proceedings. For example, Paetsch, Armin, & Frank's (2003) paper, "Requirements Engineering and Agile Software Development," is an excellent primer for understanding how these two development methods overlap and how they can part ways. Students in the gaming course were introduced to a framework for software development from two sources. The first is a whitepaper by Microsoft that outlines what the company calls the "Microsoft Solutions Framework" or MSF (Microsoft, 2003). Their approach recognized the strengths of existing approaches and described a method for combining those strengths:

> Every project goes through a life cycle, a process that includes all of the activities in the project that take place up to completion and transition to an operational status. The main function of a life cycle model is to establish the order in which project activities are performed.... The MSF Process Model combines concepts from the traditional waterfall and spiral models to capitalize on the strengths of each. The Process Model combines the benefits of milestone-based planning from the waterfall model with the incrementally iterating project deliverables from the spiral model. (2003, p. 18)

In a very real sense, Microsoft's rendering of their development cycle served as a map for students in the gaming course. While I have my own requirements-driven development experience to draw on, it is not necessary to design and teach a course on writing and gaming. Having Microsoft's model (see Fig. 1) will benefit both the students and the writing instructor in adhering to the course's calendar.

The second model that students were introduced to comes from one of the course's primary texts, *Level Design for Games: Creating Compelling Game Experiences* (Co, 2006). The author, Phil Co, is an established game designer, and he saw the game development cycle as broken into two major parts: pre-production and production. Each part had phases roughly similar to the MSF, and those phases had sub-categories. For example, the conceptual work of Co's *high concept* phase corresponded roughly with the MSF's *Envisioning* phase. Both Co's framework and the MSF used this moment for determining the project and establishing a *vision*. Because Co was specifically addressing video game development he added settling on art direction as a key component to a project's vision (2006, p. 3). These details are

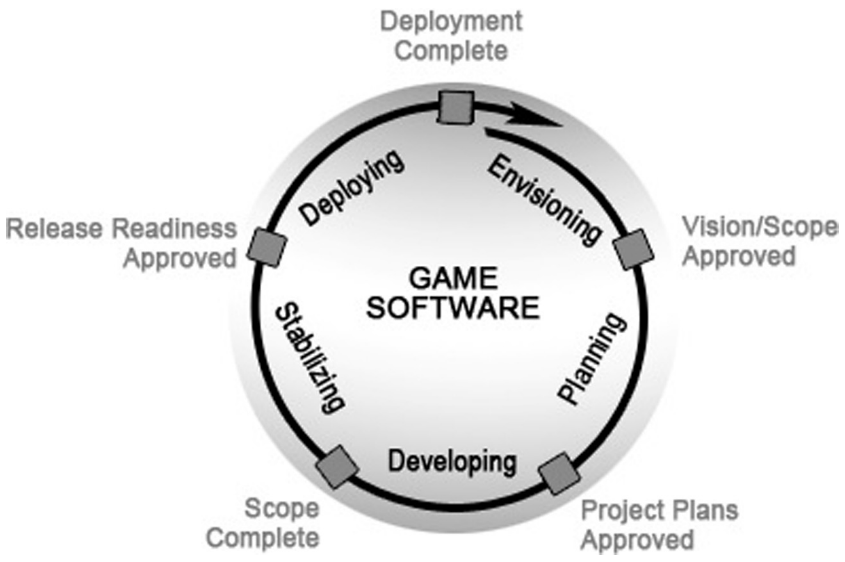


Fig. 1. Author's rendering of the Microsoft Solutions Framework.

important to point out to students, but the major difference between the Microsoft whitepaper and Co's model is that Co rendered his process as a linear model. It is reproduced here as a bulleted list:

- Pre-Production
  - Hire Team
  - Brainstorm Ideas
- High Concept
  - Determine Project
  - Establish Art Direction
- Design Document
  - Develop Design
  - Create World Map
- Prototype/Demo
  - Establish Technology
  - Establish Tools
- Production
  - Create Levels
  - Create Art Assets
  - Add Features
  - Revise Content
- Alpha
  - Improve Content
  - Replace Art Assets
  - Fix Problems
- Beta
  - Fix Problems
- Final Candidate
  - Fix Major Problems
- Gold Master
  - Take Vacation
  - Brainstorm Next Project (Co, 2006)

Co noted that depending on the project (e.g., a game sequel with existing artwork versus a brand new venture), different phases would require different amounts of time. Typically, however, the production phase was the longest, followed closely by quality control and improvement measures found in the alpha stage.

Co's model and the MSF are both extremely useful tools, but neither speak directly to a writing instructor preparing to teach a gaming course. Robison's (2008) research provided insight to game development's creative processes from the perspective of a gaming industry project manager and in turn offered writing instructors strategies for leading teams of students through a game's life cycle. Writing instructors will appreciate the audience-centered, or in this case *player-directed*, approach to the development discussion. As interactive texts, the primary goal for any game should be "to enable players to generate meaningful social narratives as a result of their gameplay experience" (Robison, 2008, p. 361). That is, writing instructors must teach students how to use their collaborative writing to generate what an audience would deem as rewarding or "fun" gameplay. And, as difficult as it may be to capture something so subjective as "fun," Robison stressed, "it is hugely important to try" (Robison, 2008, p. 363).

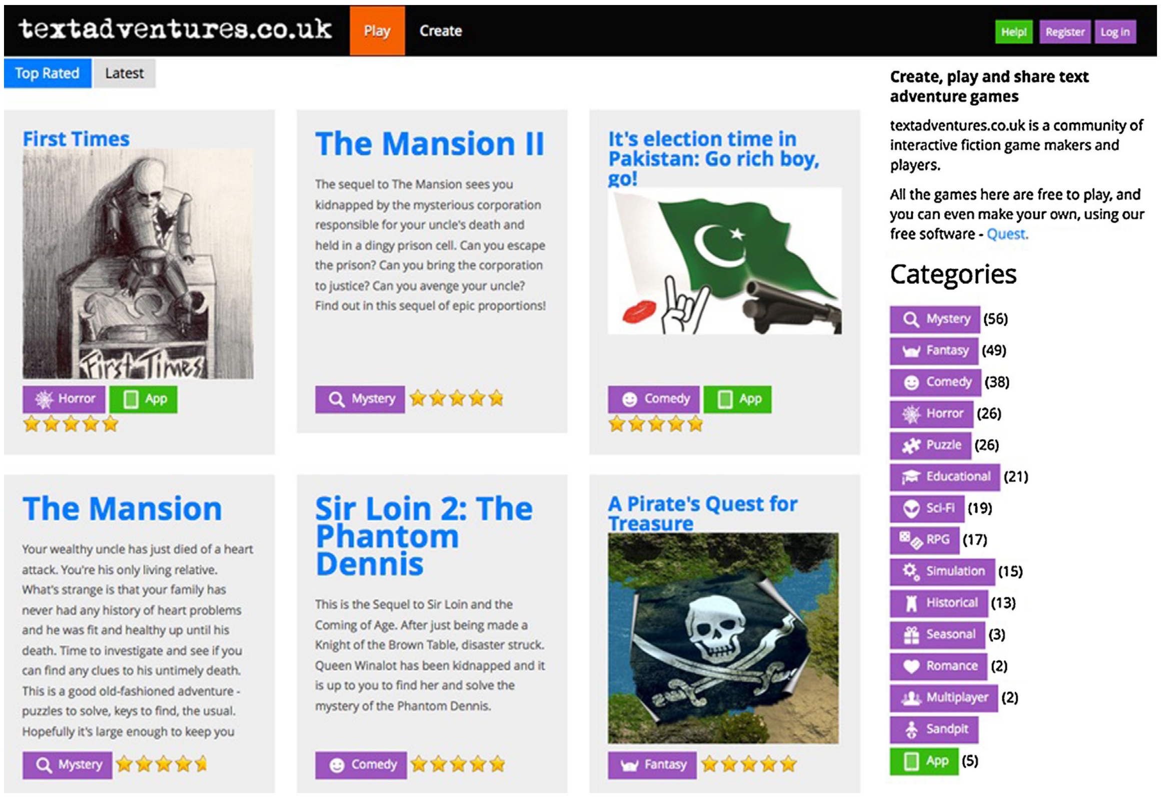
Fortunately, a writing instructor's training includes the ability to intervene in highly subjective spaces including the invention and discovery portions of the writing process. But, just as there is no single theory that guarantees writing will have its desired rhetorical effect, there is no single development strategy to guarantee a game will provide a player with an appropriate degree of *challenge* or the even more abstract *sense of accomplishment* (Co, 2006, p. 67). It will be the writing instructor's job as *teacher and manager* to lead students through the creative and subjective messiness of development. To do so, it is useful to understand how a successful project manager uses a combination of *top-down* and *bottom-up* methods to complete a SDLC and release a game (Robison, 2008, p. 364).

The project manager interviewed by Robison began his[1] process by writing out a "thesis or mission statement based on what he wants players to say about their experiences playing his game, then he has to also work with a team to craft an outline that acts as a checklist and plan for every part of the game" (2008, p. 363). While this highly scripted or top-down approach to development does provide focus for the team, it does not necessarily prevent the project manager from also utilizing a bottom-up strategy. The bottom-up method is a distillation of smaller in-game events and may simply involve "describing a series of fun moments in a game as a string of enjoyable puzzles or activities might be enough to help a team reach their goal of articulating what a game is about and what makes it fun" (Robison, 2008, p. 364). The manager may coach a team to switch between methods anytime throughout the development process including switching to a bottom-up method if, for example, the team cannot clearly conceptualize and articulate a vision for their game at the outset. Robison was clear that the project manager saw a crucial aspect of his job as "helping design teams think about both theoretical approaches to building games" (2008, p. 364). Coincidently, the project manager in Robison's study was identified as working at Microsoft, although the details of his development philosophy don't specifically mention the MSF noted above. It is clear, however, that his strategy was a hybrid model, and his combined top-down and bottom-up approach had application in an undergraduate writing classroom dedicated to gaming. Each of my teams struggled and stalled at different points in their SDLC, and it was no small task to guide teams through the process of shifting their theoretical approach to development to lead them to "those 'aha' moments in the creative process" (Robison, 2008, p. 364). Once again, I needed to consider how a version of a complete development cycle, one that could accommodate the messiness of the creative process, could be fit into a semester-long course as well as how and where I could streamline a game's development.

3. Game development software

The first step toward that streamlining was to select the software the students would use to develop their games. The course was taught in a department-controlled computer lab, so installing and configuring new software was an option. And, there were many options for teaching game development. Co's book came packaged with a limited version of Epic Games' level editing software titled UnrealEd. The included CD installed and ran on a Windows-based PC and also contained sample textures, game themes, and concept art. Development software like GameSalad [http://gamesalad.com/] offered a free drag and drop editor that required no coding or programming skills to produce

[1] There are extremely talented and successful female game designers, Kim Swift (*Portal* and *Quantum Conundrum*), Roberta Williams (co-founder of Sierra Online and designer for all the *King's Quest* games), and Jane Jenson (*Gabriel Knight*), but currently within the gaming industry "women remain outsiders" (Burrows, 2013).



First Times

Horror App

The Mansion II

The sequel to The Mansion sees you kidnapped by the mysterious corporation responsible for your uncle's death and held in a dingy prison cell. Can you escape the prison? Can you bring the corporation to justice? Can you avenge your uncle? Find out in this sequel of epic proportions!

Mystery

It's election time in Pakistan: Go rich boy, go!

Comedy App

The Mansion

Your wealthy uncle has just died of a heart attack. You're his only living relative. What's strange is that your family has never had any history of heart problems and he was fit and healthy up until his death. Time to investigate and see if you can find any clues to his untimely death. This is a good old-fashioned adventure - puzzles to solve, keys to find, the usual. Hopefully it's large enough to keep you

Mystery

Sir Loin 2: The Phantom Dennis

This is the Sequel to Sir Loin and the Coming of Age. After just being made a Knight of the Brown Table, disaster struck. Queen Winalot has been kidnapped and it is up to you to find her and solve the mystery of the Phantom Dennis.

Comedy

A Pirate's Quest for Treasure

Fantasy

Create, play and share text adventure games

textadventures.co.uk is a community of interactive fiction game makers and players.

All the games here are free to play, and you can even make your own, using our free software - Quest.

Categories

Mystery	(56)
Fantasy	(49)
Comedy	(38)
Horror	(26)
Puzzle	(26)
Educational	(21)
Sci-Fi	(19)
RPG	(17)
Simulation	(15)
Historical	(13)
Seasonal	(3)
Romance	(2)
Multiplayer	(2)
Sandpit	
App	(5)

Fig. 2. The Quest home page.

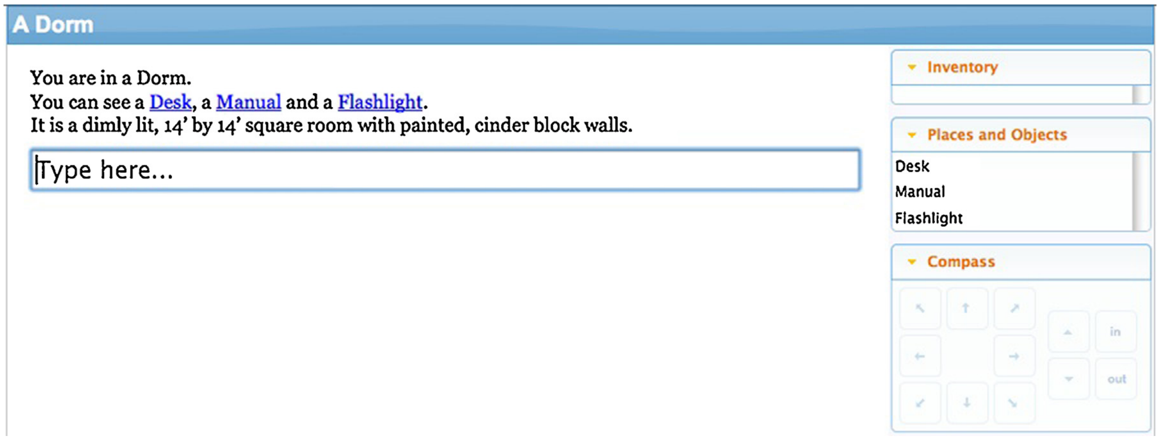
games for Mac and Windows PC platforms as well as Apple and Android devices (the Pro version sold for $299). YoYo Games offered a free version of their GameMaker software [http://yoyogames.com/studio] also for both Mac and Windows PC platforms with the option to buy their professional version for $99.99.

Among the many options, I selected software that would foreground writing. The games developed for this course deliberately downplayed aesthetic concerns and were built as classic text adventures using free and open source software called Quest. Quest (www.textadventures.co.uk) and its development environment had a moderate learning curve, comprehensive tutorials, and an active developer's forum. Quest installed and ran on Windows-based PCs, or there was a web-based version. Gaming files could be exported and played in most any web browser. Games could also be exported as Apple or Android apps. In short, Quest was an optimal choice for the course's objectives in that it was robust enough to build a worthwhile game but not so complex as to make the learning of the software itself dominate the course.

Quest did have the ability to import images and sounds into games for a richer aesthetic experience beyond a more traditional text-based adventure. However, I made it clear to the students that their writing from the pre-production and production phases was to drive their game designs by focusing on story and not on art. I didn't want students to get overly caught up in the visual or auditory aspects of their games. Precisely because Quest was text-based, I found it easier to focus students on the writing that spanned the development cycle and to impart to them that the contributions of a writer were valuable across the entire development cycle.

4. Using Quest

Before I could introduce students to writing assignments supporting their game development, students needed to get a sense of the text-based adventure game software as well as actually play Quest games. Quest's main page (see Fig. 2) provided links to free, playable games as well as the user forum and tutorials posted in the Quest wiki. The *Education* link at the right of the Quest homepage contained information on schools and instructors that have used


A Dorm

You are in a Dorm.
You can see a Desk, a Manual and a Flashlight.
It is a dimly lit, 14' by 14' square room with painted, cinder block walls.

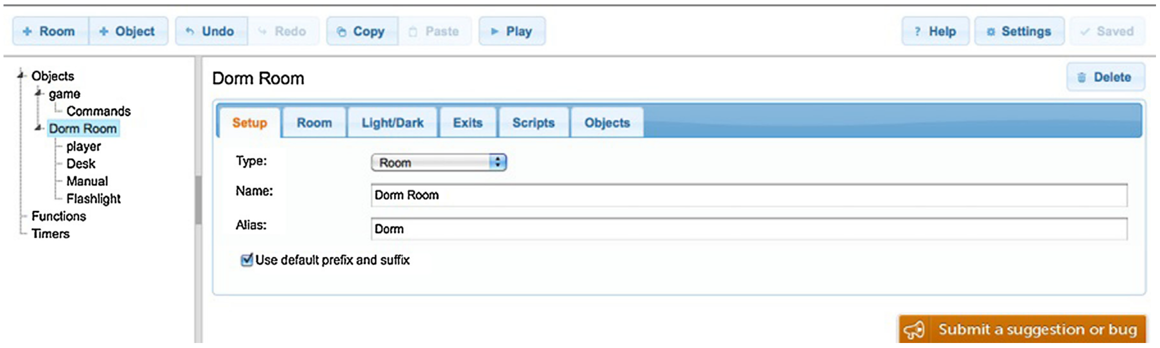
Fig. 3. The Quest gameplay interface.

Quest in the classroom. (There is a link on the page to this gaming course.) As part of their assigned readings, students were required to work through the wiki tutorials and familiarize themselves with the forum. At the beginning of the semester, I didn't know how much students would ultimately rely on the support of the Quest forum. By the end of the semester, it became clear to me (and the students) that the forum was an invaluable resource for posting questions and receiving assistance from a small but dedicated group of text-based adventure enthusiasts. As a result, students had practice with some unplanned professional writing experiences by having to interact with the forum and its members.

The gameplay interface for Quest (see Fig. 3) may look familiar to anyone who has played classic text adventure games such as the *Zork* trilogy. Quest also offered a few improvements to the genre. The interface had a clickable compass navigation at the bottom right as well as the standard textbox for entering commands. Players might also click on linked or underlined objects and activate any number of prescribed actions. The real challenge, of course, was not so much getting students interested in playing a text-based adventure games but teaching a group of students (some with very limited technology skills) how to develop their own games.

The remainder of this section attempts to demonstrate Quest's moderate learning curve and provide an overview of its basic features. Fortunately, students didn't have to master all of Quest's capabilities to produce a functional game. A new developer's primary concerns included learning how to add rooms with exits and entrances between those rooms, objects, verbs or actions, and scripts. Adding rooms or objects to a game required just a single click, and developers would be prompted to name and describe either their new rooms or objects. For example, a developer might add a new room and call it "Dorm Room." The description could be as simple as, "You can see a desk, a manual, and a flashlight. It is a dimly lit, 14′ by 14′ square room with painted, cinder block walls." Because the description mentions specific items (block walls, a desk, a manual, and a flashlight), the developer might want to introduce those items into their game as objects, especially if the player needs to interact with them. So, after clicking the *Object* tab and naming the new object "flashlight," a developer might add a description such as, "This old, heavy flashlight has an on and off switch." The manual could also be added as an object with a similarly brief description. As the game developed, the menu on the left side of the development environment (see Fig. 4) kept track of a developer's rooms and the objects in them.

To enable interaction with an object, a developer needed to determine the possible actions those objects could perform and then add the appropriate verbs. To add a verb a developer could click on Quest's *Verbs* tab and enter any word that he or she thought a player might try to use in association with a particular object. For example, in association with the flashlight object, the developer might want to add a number of verb phrases such as "turn on," "turn off," or perhaps "shake." Similarly, the manual object might need verbs like "use" or "read." While an unlimited number of verbs might be attached to a particular object, a developer also needed to be prepared to add descriptions to show the player once an action was performed. So, the player's command to "Read the manual" might return the response, "You dutifully read the manual's contents from start to finish." But what if the developer wanted the ability to read the manual to be contingent on whether or not the flashlight was in use? This is where scripts come in.



Type: Room
Name: Dorm Room
Alias: Dorm
☑ Use default prefix and suffix

Fig. 4. The Quest online development interface.

Quest's real power as development software came with the ease with which a developer could generate scripts. For this example to function, the player would need to be able to first pick up or "take" the flashlight and add it to his or her inventory. To do so, the developer first selected the *Inventory* tab for the flashlight and clicked the check box next to the option "Object can be taken." Next, a developer would return to the *Verbs* tab for the manual object. Instead of just returning a basic message when a player tried to read the manual, the developer could select the "Run a script" option from a drop down menu and then click the button "+ Add new script." The scripting options were too many to detail here, but a developer could create an engaging gaming experience with a simple *if*, *then*, *else* script. By selecting the now-visible radio button "if," the developer now saw a number of other scripting options. In this example, the developer would select "Player is carrying an object" from a menu and then select the flashlight object. The penultimate step was to add a description of what happens if a player tries to read the manual while the flashlight is in his or her inventory. In this case text such as, "You switch on the old flashlight and you can now see the manual well enough to read" would suffice. The final step was the "else" element that accounted for what the player saw if the condition of having the flashlight was not met. Quest's intuitive interface offered another textbox for the developer to add a description such as, "You try to read the manual but can't make out the words in the dark." While there was no actual coding, students began to get a solid understanding of how programming logic could function. That is, *if* the player had the flashlight *then* read the manual, otherwise or *else*, return a message saying it is too dark.

Students in the course would ultimately report that they did experience some initial frustrations with Quest but went on to say that the learning curve was worth the investment to build games with some element of sophistication. Most of the frustrations with Quest came when students were attempting to develop complex in-game puzzles that, for example, required players to collect a series of objects in a particular order all with a time constraint. Students also claimed to be able to see the connection between their technical writing assignments and the development of their games.

5. Writing assignments in a video game development course

Most of the writing assignments for the course were derived straight from Co's text and the series of prompts he offered at the end of each chapter. It should be noted, too, that the very title of Co's book suggested that game developers usually set out to build multiple levels for a single game. For the purposes of this course, students understood that they would be working toward composing a single-level text adventure game. In order for the students to tackle all of the required work, they were organized into teams of three or four. Writing instructors should be prepared to work with teams as they draft and revise assignments just as they would in any writing course. While Co's prompts were a useful mixture of top-down and bottom-up approaches to development, it is the role of the writing instructor to coach teams to use their writing as a means to think through their game's development. Equally as important, instructors need to help teams recognize when a method is not producing results. For example, if a team can't produce a refined mission statement at the outset of the development process, then the instructor needs to shift the team's focus to a bottom-up method to generate "strung-together snippets of compelling moments in the game" (Robison, 2008, p. 364). The instructor can then assist the students in articulating a larger vision for their game based on those moments. Without the instructor playing an active role as project manager, student teams can stall anywhere in a game's SDLC and fail to complete their games by the end of the semester.

Moving students from activities into producing their own games is a difficult process. For one, there is the concern of time. As Co cautioned early in his text, "A project can take anywhere from six months to six years from concept to completion" (2006, p. 2). He did qualify that these ranges represented extremes, but that is little solace here. With just a semester for novice game developers to complete the entire development cycle, students felt the pressure of their production deadlines. Fortunately, students had been working all semester with Quest and even prototyping versions of scripted puzzles that they could add to their games. That said, the major sequence of my assignment sequence streamlined some of the SDLC so that students and I could focus our attentions more productively. The sequence of assignments in this course began with defining the game, then moved to concept planning, gameplay planning, feedback, pre-production, production, and finally play-testing.

Their first task came from the end of Co's chapter 2 on "Defining the Game" where students were asked to identify their proposed game's genre, theme, target audience, and platform (2006, p. 63). They were also asked to write a short, "back of the box" description for the game. That is, they had to compose text that a player would see when he or she physically picked up the video game at a store like Game Stop or Best Buy or read about it online. The instructor should introduce this assignment as a top-down approach to development designed to provide a focus and what amounts to a mission statement for student games. Of course, the genre is determined for them: Everyone is developing a text-based adventure game. Because Quest games could be exported to run in web browsers as well as on iPhones and Android powered devices, the question about platform was also answered for them. Popular themes included sci-fi and fantasy, post apocalyptic scenarios, and medieval times. One group classified their theme as an underwater adventure that would go on to be titled *Nessie's Big Adventure*. Their description invited players to help the Loch Ness monster, also known as Nessie, escape from the lake and rejoin her family in the North Sea. The back of the box description promised a series of puzzles involving fishing boats and dams for the player to solve. While this team rated their game as "E" for "Everyone," most of the other teams opted for a teen target audience.

Moving next into what would be the "Planning" stage of the Microsoft framework, students were asked to identify and report on boundaries or limitations for their games and game characters. As Ian Bogost (2011) wrote:

> Video games are a medium that lets us play a role within the constraints of a model world. And unlike playground games or board games, video games are computational, so the model worlds and sets of rules they produce can be far more complex. (p. 4)

At this point in the semester, students were just getting acquainted with Quest's scripting abilities, but it wasn't too early to ask students to capture the constraints they wanted to develop in their games. This writing assignment was modified from Co's "Limiting What Your Players Can Do" exercise and was in two parts (2006, p. 86). First, students were to identify all the obstacles they would like to incorporate into their games, and that meant teams were forced to think about the narratives (series of events) that would comprise the telling or playing of their games' stories. Students could be coached to use this bottom-up exercise to string together these series of events to reinforce and refine their games' larger missions. Secondly, Co stressed the importance of cataloging character skills. However, because students were building text adventure games, they weren't so much concerned with a character's physical abilities such as climbing, running, crouching, and jumping that would be important in, for example, first or third-person shooter games. Instead, this second portion asked students to catalog all of the objects a player might encounter or be required to engage with to successfully complete the game. So, students created a spreadsheet of the objects most likely found in their game and the verbs or action words that characters would be able to employ to interact with those objects. For example, in a game titled *Zombie Apocalypse*, team members identified obstacles such as burning cars, locked doors, and, of course, crowds of zombies. They cataloged objects such as fire extinguishers, crowbars, flashlights, first-aid kits, and batteries. Each object was then married to its respective verb or verbs such as use, take, open, turn on, turn off, throw, and swing.

Still in the "Planning" stages of the development cycle, student groups were asked next to build on some of the decisions they had made about theme and the objects that would be incorporated into their games. Co called this exercise "Planting Level Ideas" but again, because we were building text-based adventure games it required a few modifications (2006, p. 113). The assignment was in five parts and it began by asking students to expand on their selected themes and describe the kinds of spaces, or in this case rooms, included in their Quest adventure. That is, students should have been building off their thesis or "global statement about the game" and "its overall user goal" at this stage of development (Robison, 2008, p. 362). For example, the student game *Master Chef* was set in a castle, and the player had to search its many rooms for ingredients to cook a three-course meal for the king. This group noted they would build rooms like a kitchen, a great hall, a bakery, a butcher shop, and a dungeon. Second, to help write their descriptions of those rooms, students searched online for images that captured the architectural style they hoped to convey in their game's narrative. While students did reach a level of proficiency with Quest that enabled them to display images and sounds in their games, incorporating graphics was not a requirement. Next, students identified several objects that could serve as in-game landmarks. Co suggested objects like "fountains, statues, distinct buildings, and large machines" that players could use to orient themselves within the game (2006, p. 101). The *Master Chef* group decided that a water well would be at the center of the castle, while the *Zombie Apocalypse* team used a wall of crushed cars as the northern border of their game. Fourth, groups elaborated on at least one puzzle that they wanted to incorporate into their game. The Nessie group promised a puzzle involving a fishing boat on their back of the box description, but now they offered details about getting trapped in a fishing net and what Nessie would need to do to free herself. Finally, groups were tasked with drafting a longer description of their game's opening scene. Because students would not be able to program elaborate, cinematic cut scenes, the writing for text adventure games was crucial for the player's gaming experience.

In addition to this assignment's writing and research requirements, students needed to adapt their work into a Prezi presentation to be given to their peers. As I have discussed elsewhere, Prezi's non-linear structure lends itself well to visualizing the many paths and possibilities of an adventure game (Ballentine, 2012). The presenter is free to add content anywhere across a limitless canvas while weaving connections between that content with Prezi's pan and zoom features. The presentation served the important purpose of getting students talking about each other's games and sharing ideas about how to tackle the challenges they were already experiencing with learning Quest.

The last assignment of what Co would term the pre-production phase was derived from his "Laying Out Your Level" task that involved students diagramming their game spaces start to finish. As Co instructed, "There is no standard for creating diagrams. Diagrams can be quick pencil drawings or they can be elaborate presentations done in computer programs such as Adobe Illustrator, Adobe Photoshop, or Microsoft Visio" (2006, p. 129). I provided the students with basic instruction with Photoshop and how to create layers, shapes, and text for a diagram. However, I recommended that groups begin their diagramming with graph paper, a ruler, and a pencil. Co's text provided a number of excellent models for students to work with, but his advice stressed building on the earlier work of the development cycle: "If you have created a level narrative for your level and broken down the level according to functions of each area... you're already close to laying out your level" (2006, p. 131). That is, writing instructors might need to coach student teams to think through this assignment with a bottom-up method that strings together their already developed puzzles, narratives, and character. Student diagrams showed all of their game's rooms as well as the entrances and exits between them. The rooms were labeled and/or numbered, and that label corresponded with the narrative text the groups had already been drafting. The diagrams also showed where certain objects would reside and where players would encounter puzzles.

Once students had a working version of their games, the groups formed testing partnerships. Of course students had been playing, testing, and improving on their games while they developed; however, as most developers will attest, it is another experience entirely to have someone else use your work. Co offered insight into some of the nuances between testing phases, such as alpha and beta, but for the purposes of our class we attempted to capture all of a game's concerns at once, including general gameplay issues and errors and bugs. For example, while the alpha stage does suggest that most of the creative work is complete, it traditionally does afford the opportunity for optimizing a game's performance by working with balancing how challenging different tasks are at different stages of the game. In a student game titled *Pandora's Box*, players had to escape from a moon base before it exploded. The students were proficient enough with Quest to add timers to their game, but testing would reveal whether or not the allotted escape time was both realistic and challenging. At the same time testers were also writing bug reports for specific errors encountered during gameplay. According to Co, "every developer should know how to write a bug correctly" (2006, p. 319). Given the roles that testing of all kinds plays in the software industry (e.g., even product manuals and online help systems go through usability tests), everyone involved with a project, especially the writers, should know how to test and report the results. Students were asked to use a basic form to track bugs and report them back to the game's respective team. Here is a sample bug report:

BUG REPORT
Bug Name: Space suit
Room: Air lock
Severity: High
Priority: High
Assigned to: Student A
Reported By: Student B
Reported On: 4/18/12
Reason: Error running script
Status: New
Description
Was able to take and put on space suit for space walk. However, when attempting to leave the air lock for the space walk player is killed.
Steps To Reproduce
1) Take space suit
2) Wear space suite
3) Open air lock
Expected result: Safely enter space.

The genre of the bug report would take a variety of forms from company to company but most all would ask the tester to detail the steps necessary to reproduce the error. Testers were also often asked to rate and rank a bug. That is, if the bug crashed a program, or in this case if it ended the game without the possibility of ever winning, the bug was rated severe and assigned the highest priority for repairing. At the end of the semester, students submitted all of their game development documentation as a final portfolio. Groups gave brief presentations on their de-bugged “gold master” versions of their games, and the entire class was invited to share and play each other’s text-based adventures.

6. Assessment

At the end of the semester student teams compiled the final, revised versions of their writing materials into an end-of-semester portfolio. They also submitted a final, playable version of their Quest games. While the team’s written documentation could be assessed on its own merits (organization, effectiveness, adherence to the genre), students were told that their writing would be evaluated in tandem with their Quest games. Assessing “the creative production of play” was exceptionally challenging but, as many writing instructors would agree, so is assessing “the (sometimes significantly less fun) production of writing” (Robison, 2008, p. 360). This was not to put writing and game development in competition with one another but rather to reinforce for students the connection between the two. The features, limitations, puzzles, and maps that were detailed in their writing must have come to life in their games. For both student and instructor this approach to assessment helped relieve some of the tension surrounding the perceived subjectivity of a “fun” or enjoyable game. Certainly, there were games that came out of this class that were more popular than others. However, even the less-popular games were successful by the standards of the course because of the effectiveness of the student writing and the team’s deft ability to use that writing to see through the creation of a challenging interactive experience.

7. Conclusions

As often as possible throughout the semester connections were presented between the work of the student developers and where their work would fall in relation to a software development life cycle such as Microsoft’s framework. Eyman’s original observation that games were “constructed systems” just like “any computer application” was reinforced frequently. Establishing that connection meant that students could see more practically how the writing they produced in support of their games was transferable and useful to the development of any constructed system.

Of course students also met the challenge of conceptualizing and managing the development of their own systems. But, they did not meet that challenge alone. The role of the writing instructor as project manager meant the students had assistance navigating the creative development of their games and had access to a leader to show them how to manipulate a combination of top-down and bottom-up methods. I believe that the skills students acquire in a course dedicated to writing and game development are valuable to employers and that the software industry has long recognized the difficulty of the tasks students complete to begin and finish a game.

> The hardest single part of building a software system is deciding precisely what to build. No other part of the conceptual work is as difficult as establishing the detailed technical requirements, including all the interfaces to the people, to machines, and to other software systems. No other work so cripples the resulting system if done wrong. No other part is more difficult to rectify later. (Brooks, 1987, p. 17)

Even though computer technology has advanced significantly since Brooks's original publication of "No Silver Bullet," much of what he said regarding the challenges associated with a software application's conceptual work remains true. That is, just because students have access to free, web-based development software like Quest does not mean that their games are somehow created for them. Instead, advanced technology brings advanced expectations. Stated more bluntly, in his introduction to *Dreaming in Code*, Scott Rosenberg (2007) remarked, "software is heap of trouble" and developing successful software from start to finish is no small feat (p. 10). Despite the challenges of developing a system, software also represents an enormous opportunity for professional and technical writers to make a difference. In 2008, David Michael Sheridan and William Hart-Davidson asked, "What if you could learn to write by playing a video game?" (p. 323). That's a great question, but what if you could learn to write for the software and technology industries by developing a video game? The more students understand how projects come to life and rely on writing, the better off they will be in a job interview, the workplace, or a graduate program. The message: Good writers are needed to make good software.

Acknowledgements

The author would like to thank the students in his class who graciously granted permission to have their work discussed as part of this article. He would also like to thank his college and department for supporting a new course in writing and game development. The feedback provided by outside reviewers also proved invaluable.

References

Aarseth, Espen. (1997). *Cybertext: Perspectives on ergodic literature.* Baltimore, MD: The Johns Hopkins University Press.

Abbott, H. Porter. (2008). *The Cambridge introduction to narrative* (2nd ed.). Cambridge, UK: Cambridge University Press.

Bal, Mieke. (1997). *Narratology: Introduction to the theory of narrative* (2nd ed.). Toronto, Canada: University of Toronto Press.

Ballentine, Brian. (2012, October). High concept and design documentation: Using Prezi for undergraduate game design. In *Paper presented at Professional Communication Conference (IPCC), 2012 IEEE International, Orlando, FL.*. Abstract retrieved from http://ieeexplore.ieee.org/xpl/mostRecentIssue.jsp?punumber=6391318.

Ballentine, Brian. (2010). Requirements specifications and anticipating user needs: Methods and warnings on writing development narrative for new software. *Technical Communication*, *57*(1), 26–43.

Bissell, Tom. (2011). *Extra lives: Why video games matter.* New York, NY:: Vintage Books.

Bogost, Ian. (2007). *Persuasive Games: The expressive power of video games.* Boston, MA:: MIT Press.

Bogost, Ian. (2011). *How to do things with video games.* Minneapolis, MN: University of Minnesota Press.

Burn, Andrew. (2006). Playing roles. In Carr Diane, David Buckingham, Andrew Burn, & Gareth Schott (Eds.), *Computer games: Text, narrative, and play* (pp. 72–87). Cambridge, UK: Polity Press.

Burrows, Leah. (2013 January 27). *Women remain outsiders in video game industry.* The Boston Globe. Retrieved from http://www.bostonglobe.com/business/2013/01/27/women-remain-outsiders-video-game-industry/275JKqy3rFylT7TxgPmO3K/story.html.

Brooks, Fred P. (1987). No silver bullet: Essence and accidents of software engineering. *IEEE Computer*, *20*(4), 10–19.

Carr, Diane. (2006). Games and gender. In Carr Diane, David Buckingham, Andrew Burn, & Gareth Schott (Eds.), *Computer games: Text, narrative, and play* (pp. 162–178). Cambridge, UK: Polity Press.

Co, Phil. (2006). *Level design for games: Creating compelling game experiences.* Berkeley, CA: New Riders Games.

Eyman, Douglas. (2008). Computer gaming and technical communication: An ecological framework. *Technical Communication*, *55*(3), 242–250.

Gee, James. (2003). *What video games have to teach us about learning and literacy.* New York, NY:: Palgrave McMillan.

Gregersen, Andreas, & Torben, Grodal. (2009). Embodiment and interface. In Perron Bernard, & Mark J. P. Wolf (Eds.), *The video game theory reader 2* (pp. 65–83). New York, NY:: Routledge.

Jenkins, Henry. (2006). *Convergence culture: Where old and new media collide.* New York, NY: New York University Press.

Juul, Jesper. (2005). *Half-real: Video games between real rules and fictional worlds.* Cambridge, MA: The MIT Press.

Lanier, Clinton. (2009). Analysis of the skills called for by technical communication employers in recruitment postings. *Technical Communication*, *56*(1), 51–61.

Manovich, Lev. (2001). *The language of new media.* Cambridge, MA: The MIT Press.

Meadows, Mark Stephen. (2003). *Pause & effect: The art of interactive narrative*. Indianapolis, New Riders.

Microsoft Solutions Framework version 3.0 Overview. (2003). *Microsoft Download Center*. Retrieved from http://www.microsoft.com/en-us/download/details.aspx?id=4117.

Mitchell, W. J. T. (Ed.). (1980). *On narrative.*. Chicago, IL: University of Chicago Press.

Paetsch, Frauke, Armin, Eberlein, & Frank, Maurer. (2003, June). Requirements engineering and agile software development. In *Proceedings of the Twelfth IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises. IEEE Computer Society.*. Abstract retrieved from http://ieeexplore.ieee.org/xpl/mostRecentIssue.jsp?punumber=8713.

Perron, Bernard, & Wolf, Mark##J. P. (Eds.). (2009). *The video game theory reader 2.*. New York, NY: Routledge.

Prince, Gerald. (1982). *Narratology: The form and functioning of narrative.* Berlin, Germany: Mouton Publishers.

Robison, Alice. (2008). The design is the game: Writing games, teaching writing. *Computers and Composition*, *25*(3), 359–370.

Rosenberg, Scott. (2007). *Dreaming in code: Two dozen programmers, three years, 4,732 bugs, and one quest for transcendent software*. New York, NY:: Three Rivers Press.

Ryan, Marie-Laure. (2004). Will new media produce new narratives? In Ryan Marie-Laure (Ed.), *Narrative across media: The languages of storytelling* (pp. 337–359). Lincoln, NE:: University of Nebraska Press.

Sheridan, David Michael, & William, Hart-Davidson. (2008). Just for fun: Writing and literacy learning as forms of play. *Computers and Composition*, *25*(3), 323–340.

Squire, Kurt. (2011). *Video games and learning: Teaching and participatory culture in a digital age.* New York, NY: Teachers College Press.

Walker, Janice R., Blair, Kristine L., Douglas, Eyman, Bill, Hart-Davidson, Mike, McLeod, Jeff, Grabill, & Victor, Vitanza. (2011). Computers and composition 20/20: A conversation piece, or what some very smart people have to say about the future. *Computers and Composition*, *28*(4), 327–346.