THE CURSE OF MONKEY ISLAND: HOLDING THE ATTENTION OF STUDENTS WEANED ON COMPUTER GAMES*

Brian C. Ladd
Department of Mathematics, Computer Science and Statistics
St. Lawrence University
Canton, NY 13617
blad@stlawu.edu

ABSTRACT

As the Oblingers' survey of the Net Generation makes clear [15], the cohort of students now entering higher education are always connected and immersed in computing and communication technology. Engaging these students in a computer science course requires more than "Hello, World!" and calculating sales tax. This paper reports an attempt to leverage students' widespread interest in computer games while keeping the focus of the course clearly on the fundamental concepts of computer science. Programming a text adventure game challenges and motivates students in a project-based CS1.5 course. After three years using the text adventure game assignment it is our pleasure to report that it has met almost all of its goals. Due to student interest we have relaxed the standard C++ goal. Students are learning to communicate and have far more opportunities for speaking about computer programs than they ever did before.

1 INTRODUCTION

As the Oblingers' survey of the Net Generation makes clear [15], the cohort of students now entering higher education are always connected and immersed in computing and communication technology. Engaging these students in a computer science course requires more than "Hello, World!" and calculating sales tax. Yet engaging these students

is just what we must do if we are to entice more of them into the field and they are to succeed as computer scientists.

Many of these students' interest in computer science was sparked by their interest in computer games [1]. They were raised on 16-bit game consoles and computers in every classroom [19, 5]. This paper reports an attempt to leverage that widespread interest in computer games while keeping the focus of the course clearly on the fundamental concepts of computer science. Programming a text adventure game challenges and motivates students in a project-based CS1.5 course. It also provides opportunities to introduce topics as varied as the history of computers, the fundamental separation of data and computation, and writing clear, concise prose.

This paper presents the background of our decision to use a text adventure game (or, using the shinier new term, "interactive fiction" or IF) in our CS1.5 course, related work in computer science education and digital games research, the structure of our text adventure game project, and some observations on results. Reflecting on three years of experience with this course structure, we discuss both diffculties and benefits of the approach.

2 EDUCATIONAL CONTEXT

Our University is a small liberal-arts institution with two decades of experience with a combined mathematics/computer science degree and a six year old computer science major. The standard CS1/CS2 introductory sequence is stretched across three courses, CS1, CS1.5, and CS2. This structure was adopted to ease the slope of the learning curve in CS1, which provides university distribution credit in addition to an introduction for potential computer science majors, and to permit CS1.5 to be taught as project course.

For more than a decade CS1.5 has served as a miniature software engineering course. Beginning with a rapid refresher of CS1, the course quickly focuses on a substantial, multi-phase, individual project spanning the remainder of the semester. Working on a single project for more than two months, students learn the importance of designing for extension, good documentation, and how to refactor and reuse their own code.

This general model has survived two language changes and had, historically, relied upon the instructor to come up with an interesting project. The projects were changed often to avoid cross-term plagiarism, varying from CGI-based book databases to operating system simulators to polynomial calculators. The effort involved in defining an appropriately scaled project every term meant that this was a difficult course to staff. It also meant that the student workload and learning often varied.

After some success with individualized assignments in CS1 we looked at applying that approach in CS1.5. We were also interested in the use of open projects as Becker and Sindre *et. al.* use the term [1, 18]. We sought a single project that met five goals: use standard C++; include opportunities for student communication in written and verbal form; require creative, student-designed content; provide a context for the introduction of a wide range of software engineering topics; and engage our students.

Our upper-level courses are taught in C++ so the language of CS1.5 was a given. The focus on using only standard C++ and, in turn, a simple text-based interface for the project, was made to keep the students' focus on the functions, containers, and classes they were using rather than on loading animated images on each of their buttons. GUI programming is an important skill to have but student success in CS2 is related to their mastery of the fundamentals of problem solving. Also, though our labs are MS Windows, the use of only standard C++ permits students to use almost any development platform they like.

The inclusion of a strong, creative, student-designed component of the project serves two different purposes: it increases student engagement with the project and it limits opportunities for plagiarism. Students "buy in" to a project that they have designed. This increases their interest in the success of the project. This supports our goal of engaging the students. Our recent history indicates that student copying, either from others in the course or off of the Web, is on the rise. Since each assignment is customized *by the student* (with guidance from the instructor) they are less likely to take code from another source and if they do it is obvious.

Focusing on a single project over the course of a semester runs the risk of making the lectures narrow and workman-like. A well-selected project avoids this problem by serving as a natural jumping-off point for forays across the breadth of computer science. While answering project questions will occupy some of every class period, we will touch on the history of computing, software design patterns, stacks, and the separation of computation and data.

Engaging our students was the most important goal in our search for a standard project. Nation-wide, interest in computer science as a major has been dropping over the last five years [22]. We also wanted to make sure that the students' emotional payoff was proportional to the amount of work they did. To date our choice of a text adventure game has been very successful in meeting these five goals and bringing out the best in our students.

3 RELATED WORK

The impact of computer games on computer science students has been discussed for more than a decade. Soloway talked about it at the beginning of the Nineties [19] and, with Guzdial, revisited the topic a decade later [5]. These two suggest that since the "Nintendo generation" is bored by simple text-based programs, introductory courses should rely on visual programming environments and sophisticated, preprogrammed widgets that students can link together to make interesting programs.

The success of our text-only approach shows that this conclusion is not necessarily true. Rather than building Squeak objects the students can assemble into video editors or synthesizers, we sought an assignment that was interesting, challenging, and fundamental, while sticking to the use of an industry standard language. Our work is more in line with that suggested by Wareham in his article in *OSNews*, "The Command Line - The Best Newbie Interface?" [23]. While Wareham considers beginning computer users and we consider beginning computer programmers, much of his argument remains true.

Becker [1] reports enthusiastically on using games to challenge and reward computer science students. Using C++ and **`curses`**, students create text-based versions of video games. Huang [6] talks about finding useful assignments across the CS curriculum in traditional games such as chess and checkers. Huang notes that game programming offers opportunities to introduce concepts from across computer science making it a viable approach in different levels of classes. Both of these approaches are similar to ours in intent and reported success. One major difference is our emphasis on written and oral communication as an explicit goal of the project.

The use of video games by the generation entering our college classrooms has piqued the interest of academics in multiple disciplines. Newman's *Videogames* [14] offers a survey of much of the research while Gee's recent *What Video Games Have to Teach Us About Learning and Literacy* [4] examines gameplay and game design in an educational light.

The interest in computer games is not limited to video games. Monfort's *Twisty Little Passages* [11] presents a broad overview of the history of interactive fiction as well as a literary approach to understanding the narrative nature of text adventure games. The approach presented can be used to place the students' projects in a broader context. That broader context includes the traditional use of games and puzzles as educational tools, an approach endorsed in the CS literature by Ross [16] among others.

Ju and Wagner addressed the use of text adventure games as educational tools almost a decade ago [9] though they were focused on teaching management skills *with* a computer game. At the same time Moser looked at making a text adventure game to teach programming concepts [12]. We use the text adventure game *system* as the vehicle to teach computer programming concepts and the content of the student games to encourage them to practice writing to communicate.

The class notes for Summit's Intermediate C Programming course [20] use a text adventure game in a second programming course to teach standard C. Major differences between the two approaches are our reliance on serializing complex data structures and reading them from data files and Summit's development of a scripting system. One nice thing about our approach is that much of Summit's time is spent developing dynamic dispatch of functions that C++ programmers get when they inherit.

One non-game project in computer science education that was influential in our project design was that of Sindre, *et. al*. [18]. They report success in motivating students by giving them an open assignment and letting them customize it. Becker, cited above, talks about the idea of having a threshold level of success for a passing grade and then defining some extensions to move the grade higher. We use an approach between these two: we define a threshold level of functionality (with a great deal of student flexibility) but then leave the extensions open for the student to design themselves.

4 LET THE ADVENTURE BEGIN

As stated above, our primary goals in selecting a text adventure game project for CS1.5 were that it: use standard C++; include opportunities for student communication in written and verbal form; require creative, student-designed content; provide a context for the introduction of a wide range of software engineering topics; and engage our

students. This section outlines the schedule for CS1.5, discussing the various phases of the project and how they support these goals.

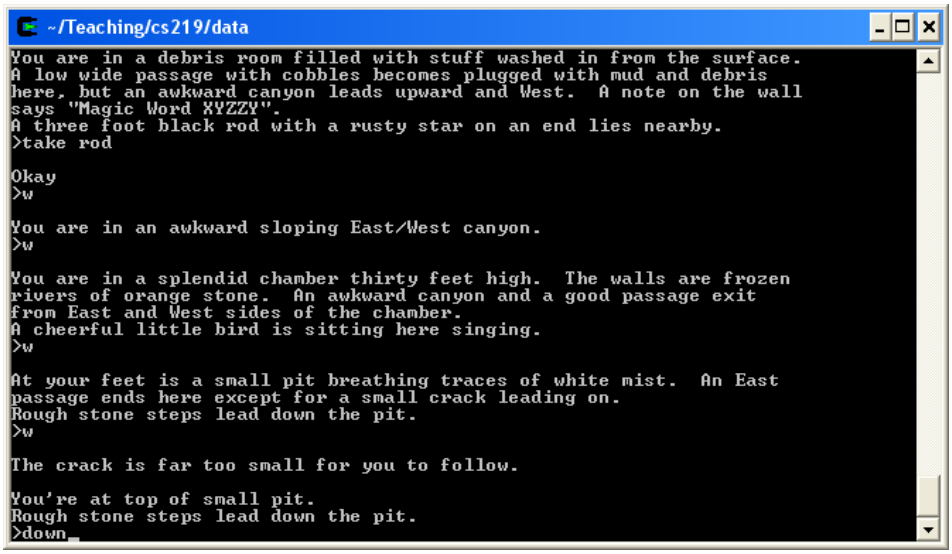

Figure 1: Getting started in Adventure

The first day's lecture provides an overview of the project including a short history of *Adventure* [3], the first of the text adventure games. A *text adventure game* is a simulated world in which players use text commands to control their character's actions. Input and output limited to text fit very well with the capabilities of early home computers as well as early programmers. Purely text adventure games enjoyed a commercial heyday through the early 1980's. Companies such as Infocom, Level 9, Magnetic Scrolls, and Adventure International enjoyed financial success with games like *Zork, The Hitchhiker's Guide to the Galaxy*, and *Amnesia*[1]. As the power of home computers grew, consumers and producers of computer entertainment began to move from text-based interactive stories to more and more graphically intensive games.

As the commercial viability of text adventure games waned, a crop of community-produced *interactive fiction* rose. Free interactive fiction authoring kits were developed and deployed including *Inform* and *TADS*. Over the past decade the quality of interactive fiction produced by the on-line community has surpassed the best commercial games and the Interactive Fiction Archive [7] holds the winners of the **Interactive Fiction Awards** (for winners of *the Comp*, an annual competition for short works) and the **XYZZY Awards**.

The students are introduced to the breadth of the field of interactive fiction and text adventure games. The course project, then, is about constructing both a *game* and a *game engine* in parallel. The separation of the game data and the interaction engine is a major thrust of the entire semester. As each new or review topic is introduced, part of the discussion is how that topic can be applied to their game. The text adventure game project provides a context for all of the knowledge presented in CS1.5.

[1] 1987, by Thomas M. Disch, the only text adventure game ever published by Electronic Arts

4.1 Review Assignments

CS1.5 begins with a series of review assignments. These short lab programs introduce students with one semester of programming in Java to C++. They also refresh primary CS1 topics: selection, iteration, function declaration, file I/O, strings, and arrays (`vector`). Before the text adventure game project, review assignments were *ad hoc* affairs unlikely to relate to the semester project. With a consistent, well-defined project, these assignments have been redesigned to support the creation of a text adventure game system.

Review assignments now emphasize string and file processing in the context of working with attribute/value pairs. Attribute/value pairs form the basis of game configuration files and the simplified serialization used for game data. The review assignments culminate in the creation of a `Dictionary` class mapping attributes to values. The dictionary is used to translate abbreviated commands to their full names. Knowing that their review program will be part of their game's user interface keeps students motivated through the rapid review of concepts.

4.2 Phases of Adventure

Phase 1: Introduction to Interactive Fiction The first day's introduction to text adventure games ends with an assignment to go and find a text adventure game, play it, and present the game to the class. Students are directed to the Interactive Fiction Archive [7] and to entries in *the Comp* in particular; these games are designed so that experienced text adventure gamers can complete them in two hours. The generally high quality, short storyline, and the limited number of puzzles makes any of these games a great introduction to the genre for students who are not gamers (or those who have never played a text-based game). Students unfamiliar with the conventions of IF must invest more than two hours in these games but most are able to finish them during the first two weeks of class.

Each student writes a short essay on the story, gameplay, and conventions they encountered interacting with their game. They also prepare a short presentation on the same topics, including a short demonstration of the game interface. This class presentation is the first of three presentations (the other two are outlined below), a chance for students to get comfortable with oral presentation. It also lets students see a broader cross-section of text adventure game conventions than were present in the one game they played.

This phase of the project is focused on communication and engaging students. The set-up for it, showing off the original *Adventure*, leads naturally into a discussion of the history of the form, the prehistory of the Internet, and even a history of computers. Hearing students talk about how compelling the story in a game is or how they want to incorporate similar plot twists indicates that the engagement part of this project works right from the first.

Phase 2: Game Design After their introduction to interactive fiction, students design their own. The game design assignment is wide open; this is where customization and individual creativity come to the fore. Students' game worlds need approximately one

hundred locations, twenty inventory items, and ten "critters" with which the player will interact.

While students are considering their design, they are assigned Nelson's *The Craft of Adventure* [13]. This document puts them inside the head of a very good interactive fiction and interactive fiction **system** author. It provides masterful insight into the design of text adventure games.

The design assignment specifies as little as possible; students select *who* the player is, *what* the goal of the game is, and *how* the player will interact with critters (combat, conversation, something in between). We have been using different game assignments for almost ten years. Research shows that one reason female students are not attracted to computer science is the level of violence in computer games [10]. In response we have always permitted students to select the level of conflict in their games.

Students prepare a five page game design and an initial map (containing about a quarter of the locations) and give another presentation. Initially the presentation was not part of the assignment. When it was added, the interaction between the students took a quantum leap. Knowing what sort of story the other students are working on and the "special features" they hope to include excites the entire class. It also sets up the final presentation of the finished games.

Students customize their assignment; research suggests that such customization has positive effects on student outcomes [21]. It lessens the chance of plagiarism while permitting students to discuss their general approaches. Students have selected "similar" game worlds but we have never had them create similar game systems. Topics presented during this phase include the difference between a "choose your own adventure" book where there is no memory of previous actions and a good adventure game where choices have consequences.

With nine phases during the semester, quick turn-around in grading is essential. This is especially true with the game design. The students' enthusiasm often leads to overly ambitious designs and much of the feedback involves finding where they can be scaled back or where features can be implemented incrementally. Students need to see and absorb these changes as soon as possible.

Phase 3: Location Class With scaled-back game designs in hand, students design and code their `Location` class. They also write a temporary driver program and develop the first 10% of their game world. The locations in a text adventure game are the nodes of a digraph and students have to work out how to serialize such a structure. The text data file is required to be self-describing as per Bentley [2] with field-order independence and support for comments. While a file format based on object markers and attribute/value pairs is suggested in the assignment, students are free to design their own within certain parameters.

This phase, the first where the students write code, supports our goal of standard C++ since the serialization does not depend on any particular toolkit. It also leads to discussions of serialization in those toolkits. This is a perfect opportunity to discuss the idea of a data driven program, a text adventure game system where all features of the game are specified in data files. Many current students have experience with the

modification ("mod") communities around various commercial games and find the idea that their game could be "modded" exciting.

Phase 4: Detailed Design Students next take their game design, rewrite it to reflect the guidance they were given in Phase 2, and they expand it. This time the design focuses on the gameplay; the commands players will use to interact with the world. They also determine the attributes of both critters and items (locations were designed as part of implementing them).

This phase permits the introduction of different views of games. Some reading and lecture material is taken from Salen and Zimmerman's *Rules of Play* [17] including discussion of games as simulations, entertainment, narrative, and the study of games as literary artifacts (some of Monfort's analysis is used here, too). Being able to take a step back from a summary of a game to what made the game fun (Phase 1) is the first step to being able to design games; while this course is focused on programming in C++, an introduction to game design is necessary so that student games are, in some sense, fun to play. That fun factor is one thing that engages the student. This phase also has works on the students' written communication skills.

Phase 5: The Game Class As the next phase begins, students are shown a simplified UML diagram of the overall architecture of their game (see Figure 2).

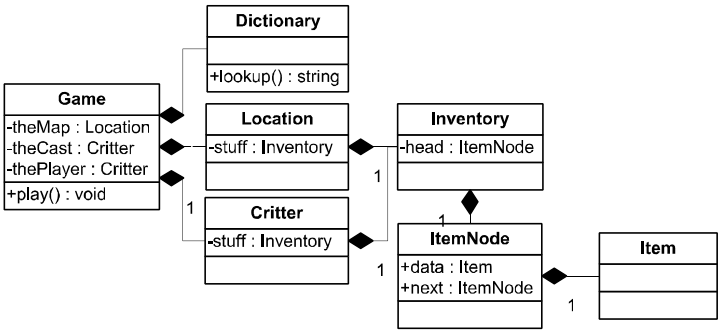

Figure 2: Text Adventure Game Architecture UML

Proposing an architecture to the students pushes all of the programs to have the same structure, running counter to the goal of student customized projects. This is a necessary compromise because CS1.5 students are not yet prepared to design a multi-class solution and without an architecture they would be unable to fit their work into the overall project.

This phase is the design and construction of the **`Game`** class. It is in standard C++. This phase leads to discussion of an infinite "game loop" that displays the game state, processes user input, and then updates the game state. Students are surprised to hear that this loop is at the heart of all interactive programs. This phase also introduces the idea of using pictures as part of documentation.

Phases 6—8: Critter, Item, and Inventory Classes to a Playable Game Going from having a navigable map to a playable game could be done in a single phase. With about five weeks, students could be set loose to implement **`Critter`**, **`Item`**, and **`Inventory`** on their own schedule. This would have the advantage of giving them a

feeling of control over their workload. Unfortunately, students facing five weeks with nothing "due" are unlikely to spread the work across the time, instead procrastinating and trying to complete a lot of code in a little time.

To help students spread their effort, the work is divided into three phases. Students implement the **`Critter`** class, serialize it, and extend the display routines. Next, students implement **`Item`**, serialize it, and implement a linked-list based **`Inventory`** class. This is the one non-STL container used in the course. Pointers are one of the primary new concepts introduced in CS1.5 so having an integral programming phase that uses them is important. Finally, students implement the additional commands they planned permitting interaction with the whole game world, including any scoring options.

Phase 9: Extended Game By the end of Phase 8, students have a working text adventure game. As with the grading described in Becker [1], a working text adventure game with locations, items, and critters, constitutes a threshold. Completing it by the end of the semester (while it is "due" two weeks *before* the end of the semester) is worth 2.5/4.0. The final phase of the project, due at the end of the semester, is to extend the game using some technique presented in class: inheritance, templates, conversation trees (really just "mini-maps" with different labels on the movement directions), combat, saving/restoring games in progress, etc.

We also present MS Windows specific GUI programming at this point and offer students the chance to extend their project by adding an MFC-based GUI to it. The change from having control of the game loop to an event-driven program is a real challenge but many students want to have a "pretty" program when they are done. Student feedback on this led us to bend our rule to use only standard C++.

As can be seen in Figure 3, the interfaces are often crowded with buttons. This

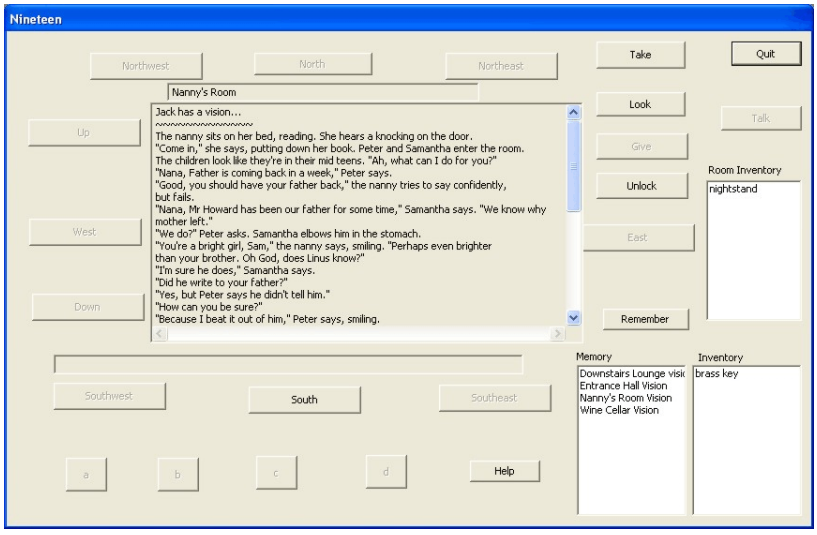

Figure 3: Student Ghost-hunting Adventure Game "Nineteen"

particular game is one of the most complex ever completed in this class with inventory items that can be containers, a separate inventory of "memories" (dying returns all items to their original places but leaves memories in tact), and conversation (the row of disabled numbered buttons across the bottom of the dialog box).

During the final exam period, all of the students return to the classroom for one last time, each having prepared a slightly longer presentation on their completed game. They highlight the extensions they completed and lament the extensions they could not get working. These presentations are well received by everyone in the class. They are the capstone on the course, showing students just how far they have all come.

5 EVALUATION

Developing a text adventure game assignment is a lot of work. Writing a text adventure game system *and* a text adventure game in one semester is also a lot of work. Is it worth it?

Ours is a small liberal arts college with a short history with a computer science major. The national decline in interest in computer science is reflected in falling enrollments in our CS1 course. One hope in introducing this project was that it would draw more students into CS1.5; results to date are mixed with no clear trend. There is no question, however, that students who complete CS1.5 are motivated and prepared to go on to CS2. Only one student has chosen not to continue and only one student who went on failed data structures.

Another measure of the course is the anonymous student course evaluation given in every class every semester. Looking at the last two semesters taught by the author *before* the text adventure game and the most recent two semesters, every measure of student satisfaction is better. The only measure that might be troubling is perceived student workload.

This project is very large. Even with high-level architectural design and many useful snippets of code presented in class lectures, students work very hard in this course. The amount of work and new material requires a considerable time commitment from the instructor for office hours and other outside-class contact time. It also requires the selection of a good teaching assistant to provide additional time for questions to be answered. We are examining using a Wiki or similar shared editing space to assist students in asking, answering, and finding previous answers of questions; the efficacy of such a system is pure speculation at this point.

The integration of writing, oral presentation, program design, and coding makes this course a fantastic introduction to software engineering. This helps to overcome students' tendency to compartmentalize, thinking writing is for English class, coding is for computer science, and never the twain shall meet. Our university has a strong commitment to teaching students how to communicate across the entire liberal arts curriculum and this project begins addressing that right at the beginning of the computer science curriculum.

Student engagement has improved. More students in the middle of the class seem to be motivated to work a little harder than they did before. Seniors who completed the class three years ago still talk about their text adventure games both among themselves and with more junior students. Several seniors have even commented that every program they have written here has had pieces of the design of their text adventure game.

6 CONCLUSION

After three years using the text adventure game assignment it is our pleasure to report that it has met almost all of its goals. Students have far more opportunities for speaking about computer programs than they ever did before. The descriptions of the locations, items, and critters exercise both the writing skills and creativity of each student.

The assignment is modified by each student and that made the one unfortunate case of plagiarism that we have seen obvious. The customizing has also had positive effects on student motivation. The game provides a chance to introduce patterns, design documents, the graph data structure, designing for extension, self-describing data driven programs and a host of other "advanced" topics in CS1.5.

We plan to examine how to integrate more game design and game studies into this project. Many of the readings on Jerz's Interactive Fiction Bibliography [8] are good candidates. We are now offering an upper-level games course and it would be good to provide students with a slightly deeper understanding of game studies so that they can decide whether or not to take that course. We also hope to find ways to work together with our Film Studies department (the current home of media studies) to see if we can arrange cross-curricular sharing of ideas.

The use of a project-based CS1.5 has proven its usefulness over the years. The text adventure game project has, also, proven its worth. It has met all of our pedagogical goals *and* eased the preparatory burden for teaching CS1.5 while still keeping the project fresh for both the Atari and the Nintendo generations.

ACKNOWLEDGEMENTS

Thanks to Dr. Collen Knickerbocker who came up with the idea of using a text adventure game in CS1.5. Thanks also to all anonymous reviewers of drafts of this paper; their insightful questions and comments greatly improved the presentation of our work.

Finally, many thanks to Lucas Arts for the many hours wasted on their best-selling (non-text) adventure game, 1997's Adventure Game of the Year, *The Curse of Monkey Island*. Information at http://www.lucasarts.com/products/monkey/default.htm

REFERENCES

[1] K. Becker. Teaching with games: the minesweeper and asteroids experience. *J. Comput. Small Coll.*, 17(2):23–33, 2001.

[2] J. Bentley. *More Programming Pearls*. Addison Wesley Publishing, Reading, MA, USA, 1988.

[3] W. Crowther, D. Woods, and K. Black. Colossal cave adventure. ftp://ftp.gmd.de/if-archive/games/pc/adv350kb.zip. MS-DOS Executable.

[4] J. P. Gee. *What Video Games Have to Teach Us About Learning and Literacy*. Palgrave Macmillan, New York, NY, USA, 2003.

[5] M. Guzdial and E. Soloway. Teaching the nintendo generation to program. *Commun. ACM*, 45(4):17–21, 2002.

[6] T. Huang. Strategy game programming projects. In CCSC ’01: Proceedings of the sixth annual CCSC northeastern conference on *The journal of computing in small colleges*, pages 205–213, USA, 2001. Consortium for Computing Sciences in Colleges.

[7] Interactive fiction archive. http://ifarchive.org/.

[8] D. Jerz. Interactive fiction: An introduction to scholarship. http://jerz.setonhill.edu/if/bibliography/intro.htm, August 2001.

[9] E. Ju and C. Wagner. Personal computer adventure games: their structure, principles, and applicability for training. *SIGMIS Database*, 28(2):78–92, 1997.

[10] S. Kiesler, L. Sproull, and J. S. Eccles. Pool halls, chips, and war games: women in the culture of computing. *SIGCSE Bull.*, 34(2):159–164, 2002.

[11] N. Monfort. *Twisty Little Passages: An Approach to Interactive Fiction*. MIT Press, Cambridge, MA, USA, 2003.

[12] R. Moser. A fantasy adventure game as a learning environment: why learning to program is so di_cult and what can be done about it. In *ITiCSE ’97: Proceedings of the 2nd conference on Integrating technology into computer science education*, pages 114–116, New York, NY, USA, 1997. ACM Press.

[13] G. Nelson. The craft of adventure. Web, http://www.ifarchive.org/ifarchive/info/Craft.Of.Adventure.T1.letter.pdf, 1995.

[14] J. Newman. *Videogames.* Routledge Introductions to Media and Communications. Roultledge, New York, NY, USA, 2004.

[15] D. Oblinger and J. Oblinger, editors. *Educating the Net Generation*. Educause, June 2005. http://www.educause.edu/educatingthenetgen/.

[16] J. M. Ross. Guiding students through programming puzzles: value and examples of java game assignments. SIGCSE Bull., 34(4):94–98, 2002.

[17] K. Salen and E. Zimmerman. *Rules of Play : Game Design Fundamentals*. MIT Press, Cambridge, MA, USA, October 2003.

[18] G. Sindre, S. Line, and O. V. Valvåg. Positive experiences with an open project assignment in an introductory programming course. In *ICSE ’03: Proceedings of the 25th International Conference on Software Engineering*, pages 608–613, Washington, DC, USA, 2003. IEEE Computer Society.

[19] E. Soloway. How the nintendo generation learns. *Commun. ACM*, 34(9):23–ff., 1991.

[20] S. Summit. Intermediate c programming assignments. http://www.eskimo.com/scs/cclass/asgn.int/index.html, 1999.

[21] B. Toothman and R. Shackelford. The effects of partially-individualized assignments on subsequent student performance. In *SIGCSE '98: Proceedings of the twenty-ninth SIGCSE technical symposium on Computer science education*, pages 287–291, New York, NY, USA, 1998. ACM Press.

[22] J. Vegso. Interest in cs as a major drops among incoming freshmen. *Computing Research News*, 17(3), May 2005. http://www.cra.org/CRN/articles/may05/vegso.

[23] R. Wareham. The command line - the best newbie interface? *OS News*, 2004. http://osnews.com/story.php?news id=6282.