It's Not Just a Class, It's an Adventure: Teaching Web Development Through Game Creation

George M. Wyner
Boston University School of Management
gwyner@bu.edu

Benjamin Lubin
Boston University School of Management
blubin@bu.edu

Introduction

Web application development, however arcane it might have once appeared, is becomingly increasingly relevant to a business education in general and an information systems education in particular. There is an increasing interest by educators in making "computational thinking" available to all college graduates as part of a notion of 21st century literacy (Stross 2012). This accompanies an increasing perception among the general public that it is important to understand how the web works and to be familiar with the technologies that are used to build web applications, with some professionals viewing this as an essential component of their skill set (Wortham 2012). Information systems (IS) faculty have expanded the IS curriculum to include courses on web development, and more recently web application development.

In this paper we describe our approach to teaching this topic to masters students concurrently pursuing an MBA and a Master of Science in Information Systems. Our approach is intended to address the challenge of teaching a complex topic to students with a limited technical background, and to do so in an extremely brief time frame (less than three weeks). Our method consists primarily of a hands-on exercise in which students create their own web application. This exposes them to an important set of web technologies — HTML, CSS, web server, programming language, and application server — without plunging them into an overwhelming morass of technical detail.

For the exercise, we provide student teams with a set of tools for building a text-based adventure game using the Google AppEngine hosting service and application framework. Students then write their project using the Python programming language. Because the specific nature of the game is left completely unspecified, students develop something that reflects their own imagination even as they are learning web application software engineering and architecture.

The initial version of this exercise was conducted in the spring of 2012 with 44 students. In this paper we describe the exercise and summarize the teaching materials we have developed. The code for our framework and other teachings materials, including a demonstration game, have been made available to the IS community under the MIT open source license (www.opensource.org/licenses/MIT). Links to resources will be provided throughout this paper.

Teaching Goals

The goal of this exercise is for students to develop an understanding of web technologies and the architecture of a web application. We believe this focus on a web application is important for several reasons:

- Web applications are an important arena in which companies compete and innovate.

- Web applications bring together the entire range of topics which students are studying in their technology courses, including programming, database, and the full range of web technologies including HTML, CSS, and web services.
- Web applications embody important architectural concepts which students have been studying, including modularity, the notion of an application stack, as well as APIs.
- Creating a working web application would be a significant accomplishment and one that students could show to friends and family, as well as discuss with potential employers.

Teaching Strategy

Teaching web application development brings with it significant challenges, especially the need to provide working knowledge of a large number of technologies (Yue and Ding 2004). This challenge is increased when the students being taught have limited technical background, as in our program. We have the added challenge of a compressed time frame in which we are expected to deliver this learning experience (Our course is delivered in a 3-week long "intensive" session that occurs after the spring semester). We have consequently developed a different approach than that used in existing curricula. Two key differences in our method concern the choice of project and the choice of infrastructure:

Choice of project

Hauser et al. have taught web development using a business case involving currency trading (Hauser et al. 2010). One advantage of their domain choice is that the case could then be used in other courses, such as database and user interface design, giving students an integrated experience. In the short time we have available, we do not have the time to unpack such a complex business example, and in moving to a simpler project we run the risk of a less compelling student experience. In order to compensate for this we sought a project domain which: (1) is immediately familiar to students, allowing them to plunge into a design conversation with few preliminaries, and (2) allows for lots of creativity and variation. After considering several alternatives, we decided that these criteria were particularly well met by having students create their own adventure games. Not only are students almost certainly familiar with online games, the content of the game would be wide open for invention, motivating active participation.

Information Systems and computer science faculty have identified numerous benefits from using game design in the classroom. Jones observes that implementing a game allows students to apply many computer science concepts in an engaging manner (Jones 2000). Becker experienced a number of additional benefits from assigning game development to students: (1) Students inherently know (because they decide) how the game should work and are motivated to get it to work correctly. (2) Students want to show off their games and typically go well beyond what is required in order to make their games even better. (3) Students "cross a significant perceptual boundary: they become the 'creators'. They go from experiencing the 'magic' to being the 'magicians'" (Becker 2001).

By "adventure game" we mean the original text-based games in which a player finds him or herself in a cave or house or other environment and can issue commands to move from place to place. In each location in the game world the user may encounter objects and perhaps creatures and obstacles. The user can add objects to an inventory and can do things with the objects. For

example, if a user has a lamp in her inventory she could turn the lamp on, which would in turn allow her a view of what might otherwise be a dark chamber. Possessing a key might allow a user to open a door to gain access to new parts of the game world.

Early examples of such adventure games include Colossal Cave (Jerz 2007) and Zork (Lebling et al. 1979). Originally such games were purely text based: locations were described with text only, and users entered commands on a command line. One might argue that games such as Myst (Miller and Miller 1993) represent the logical evolution of such games into a graphical form in which the game world is rendered as a virtual environment and users employ the mouse to explore the world and interact with its objects.

In our case staying with the text based approach makes sense because, while less flashy, it builds on previous classwork in which students learned how to work with text in Python and does not require a knowledge of animation techniques or complex user interfaces. Instead, the interface is a fairly straightforward use of basic HTML form elements. We do provide students the ability to include graphics such as images of rooms and the objects in them. This gives students a chance to learn how to use images in a dynamic web page and introduces options for visual creativity.

Choice of infrastructure

We have chosen to use Google AppEngine (developers.google.com/appengine) for several reasons:

- AppEngine is hosted at no cost by Google. All students need to do is sign up for an AppEngine account with Google and download a software development kit (SDK). There is no need to configure a hosting environment.
- The AppEngine SDK is platform independent and can be installed easily on student laptops, which allows students to try out their apps on their own computer and then upload those apps to Google's servers.
- AppEngine is integrated into the development environment students are already using. AppEngine supports Python, which we use as our teaching language. Our students use Eclipse (www.eclipse.org) as their programming environment. To make Eclipse work with Python, students install the PyDev plugin (pydev.org), which includes support for Google AppEngine. This means that students are able to run AppEngine applications on their laptop and also deploy them to the Google platform, all from within the Eclipse IDE.
- The "webapp" framework, which is included in the AppEngine SDK, is much simpler than Rails, Django (www.djangoproject.com), and other web application frameworks. While webapp offers less functionality out of the box to developers, it also offers a gentler learning curve for students. Given the limited time students have to work in this environment, we find the limited functionality a worthwhile tradeoff for the increased understandability Google has built in.

The Game Framework

We provide students a framework on which they can build a game, along with a simple game demonstrating how the framework can be used. The framework consists of an object model including base classes for rooms, inventory items, and actions. Students add rooms, actions, and

inventory items by creating subclasses of these base classes and modifying their attributes and methods. A consequence of this design decision is that we need to give students a brief introduction to the concept of class and object so they are not lost when navigating the code we provide for them. The source code for the framework and a demonstration game can be found at code.google.com/p/msmba-ae-game/.

In addition to the set of objects we provide to students, the game framework makes use of the webapp framework included with AppEngine. This framework includes a set of classes that allow a developer to define how URLs are mapped to a set of *handlers*, each of which handles get and post requests. We also make use of the template framework included with AppEngine, which allows a developer to generate web pages that include a mix of static html and dynamic content.

The game display includes a description of the current location. While students are free to create any setting they want we will generally use the term "room" to refer to each location in the game world. The display also includes a list of items in the room and the items currently in the player's inventory. To avoid the complexity of a command line parser, the game displays available actions as a series of HTML buttons. These actions include the ability to: pick up items and place them in the inventory, interact with objects already in the inventory, drop items from the inventory, interact with the room, and navigate in various directions. An example of this interface, taken from the demonstration version of the game is shown in Figure 1.

A key design decision we have made is to avoid the use of a database. It would seem natural to store the game world in a database, since this would allow for changes to the game without changing the code. We elected not to do this for two reasons: First, we wanted to avoid the need for students to learn how to use the AppEngine database features, focusing their initial efforts on learning the architectural, web and programming technologies and techniques needed for dynamic web design. Second, we wanted ensure our materials facilitated students working in the code, rather than simply editing content in a database. As a result of this decision, the structure of the game world is stored in the code itself. The state of the game for any player is stored in a session cookie, which affords the opportunity to teach the students about stateless/stateful protocols and the mechanism by which HTTP sessions operate.

Teaching Materials

Prior to encountering our exercise students have been exposed to the basics of Python programming, including variables and simple data structures, control flow, and functions. We have also spoken about the importance of APIs and the efficacy of stack-based architectures. The students have all installed the Eclipse IDE with the PyDev extension, which allows them to use Eclipse to develop and run Python programs. Consequently, we begin the course by having the students download and install the AppEngine SDK on their personal laptops.

Prior to the start of the game development exercise, we give the students a chance to revisit their Python programming skills and then briefly expose them to the core concepts and syntax of object orientation so they are equipped to read the project code we provide them. We also provide lectures that cover the key building blocks of web applications including HTML, CSS, as well as the architecture of web applications including databases and web servers. The lectures are self-contained but students are strongly encouraged to rely on *Using Google AppEngine* as a reference (Severance 2009). This book provides students with concise discussions of each of

these technologies in an AppEngine context. We also found great utility in the slides, screencasts, sample code, and handouts provided by Chuck Severance on his website www.appenginelearn.com.

We then walk students through creating an AppEngine account and building a very simple application of their own. By the time students have finished their first exercise, they have a basic familiarity with the AppEngine environment and understand the crucial idea of mapping each URL to a handler object that responds to get and post requests.

While students are rekindling their programming skills and learning the basics of web application programming, we divide the class into teams and brief them on the game assignment. Students are given access to a simple demonstration game that they can use both as a reference and as a basis for their own creations. We also have each team create a project repository on Google Code (code.google.com), which enables them to share their code with each other (and us).

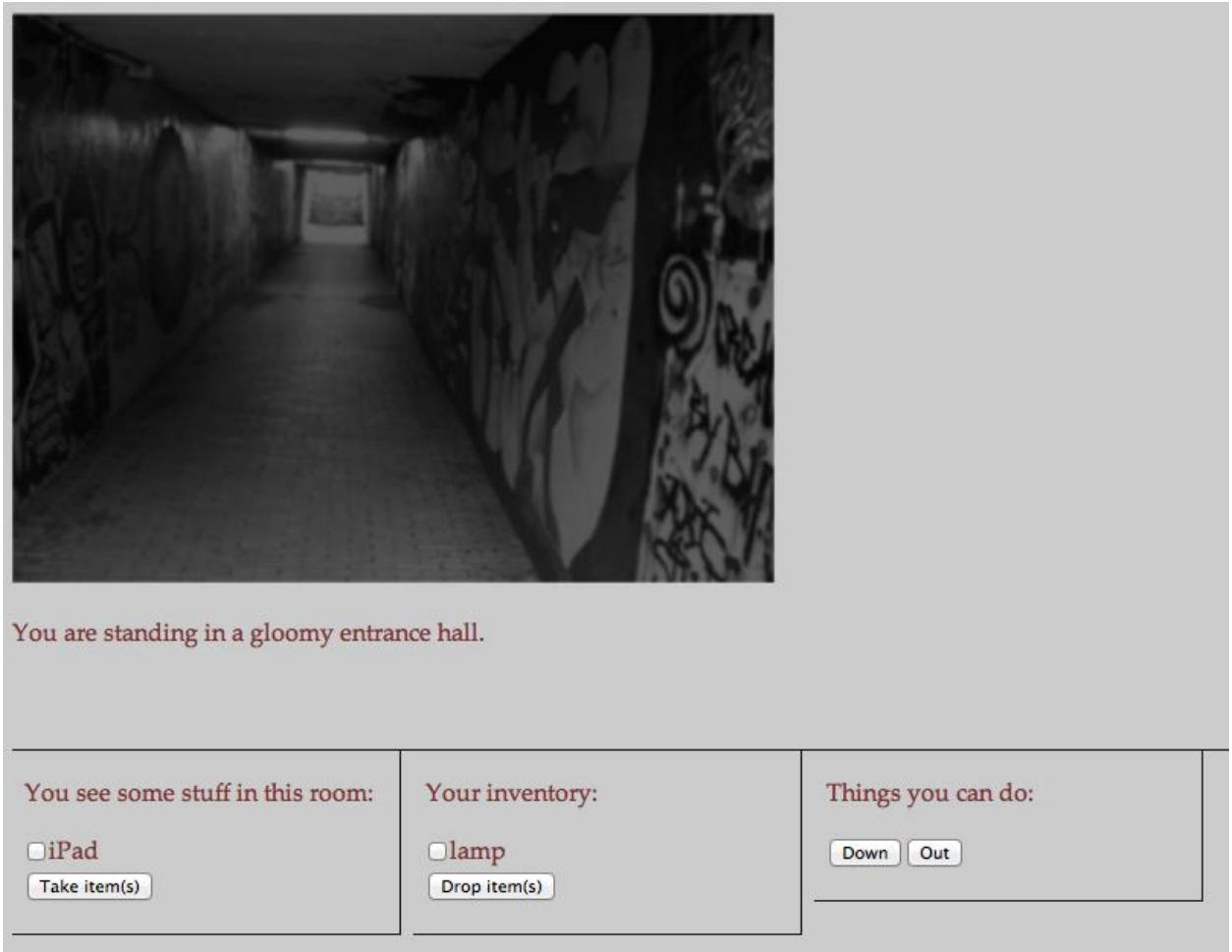

Figure 1. Sample Game Interface

Each team is expected to come up with a basic design for their own game, including the theme, a set of rooms (or other locations), objects in the various rooms, and a set of puzzles which involve collecting objects and using them to overcome obstacles. We provide documentation on how to create rooms, objects, and object behaviors, and how to manage stateful and conditional

outcomes. This documentation can be found in the code repository and consists of a class diagram summarizing the most important attributes and relationships in the framework, basic documentation of the framework, and a set of hints on how to develop a project.

As the course progresses students are given large blocks of time in the classroom to work together developing their game. During these periods, the teaching staff make themselves available to provide guidance and to help resolve technical issues. At the conclusion of the course, each team presents its game concept and offers a demonstration to the entire class. All of the games created are then publicly accessible through AppEngine. Links to the games developed by students this past semester are available on the Google Code site.

We have found this team format to be effective in the past for several reasons. The number of teams is small enough so that each team can present its work to the entire class and faculty, which creates considerable motivation to be creative and to deliver a working application. Further, by including students with a range of technical abilities on each team we allow students to learn from each other. We believe this student interaction is particularly fruitful in the web application context, as different students can contribute to the application in different ways, not only by programming, but also by designing the game's challenges and the algorithmic reification thereof, as well as creating web content including text, markup and images.

References

Becker, K. 2001. "Teaching with games: the minesweeper and asteroids experience," *Journal of Computing Sciences in Colleges* (17:2), pp. 23-33.

Hauser, K., Olsen, D., and Fadel, K. 2010. "An Integrated Approach To Teaching Web Development," *The Review of Business Information Systems* (14:1), pp. 43-59.

Jerz, D. G. 2007. "Somewhere Nearby Is Colossal Cave: Examining Will Crowther's Original ''Adventure' 'In Code and in Kentucky," *Digital Humanities Quarterly* (1:2).

Jones, R. M. 2000. "Design and implementation of computer games: a capstone course for undergraduate computer science education," *ACM SIGCSE Bulletin* (32:1), pp. 260-264.

Lebling, P. D., Blank, M. S., and Anderson, T. A. 1979. "Special Feature Zork: A Computerized Fantasy Simulation Game," *Computer* (12:4), pp. 51-59.

Miller, R., and Miller, R. 1993. "Myst," Cyan.

Severance, C. R. 2009. *Using Google App Engine*," Sebastopol, CA: O'Reilly Media.

Stross, R. 2012. "Computer Science for Non-Majors Takes Many Forms," in *The New York Times.*

Wortham, J. 2012. "A Surge in Learning the Language of the Internet," in *The New York Times.*

Yue, K. B., and Ding, W. 2004. "Design and evolution of an undergraduate course on web application development," in *ACM SIGCSE Bulletin*, pp. 22-26.