THE PENNSYLVANIA STATE UNIVERSITY
SCHREYER HONORS COLLEGE

COLLEGE OF INFORMATION SCIENCES AND TECHNOLOGY

CLASSROOMS UNLOCKED: A SOFTWARE PROTOTYPE FOR SEARCHING CLASSROOM AVAILABILITY IN THE INFORMATION SCIENCES AND TECHNOLOGY BUILDING

BENJAMIN TYLER DODGE
SPRING 2013

A thesis
submitted in partial fulfillment
of the requirements
for a baccalaureate degree
in Information Sciences and Technology
with honors in Information Sciences and Technology

Reviewed and approved* by the following:

Guoray Cai
Associate Professor of Information Sciences and Technology
Thesis Supervisor

Xiaolong (Luke) Zhang
Associate Professor of Information Sciences and Technology
Honors Adviser

* Signatures are on file in the Schreyer Honors College
ABSTRACT

Undergraduate students in the College of Information Sciences and Technology (IST) at the Pennsylvania State (Penn State) University are interested in searching classroom availability in the IST Building but have difficulty effectively completing this task. This paper proposes the design of a software prototype called ClassroomsUnlocked that uses the geo-centric integration and presentation of classroom availability information in the IST Building in order to support more effective information seeking and sense making in a range of common student activities. This software prototype was inspired by the ideas of Donald A. Norman and Jakob Nielsen, guided by personas, scenarios, and user stories, and built from a modern mix of Java, Perl, PHP, SQLite, HTML5, and JavaScript. According to a preliminary Keystroke-level Model (KLM) analysis, the prototype may be as much as 85% more efficient than the existing paper-based system. The results of this analysis have demonstrated the merits of this novel approach to searching classroom availability in the IST Building. This approach has the potential to work well in both university and non-university settings with certain limitations.
# TABLE OF CONTENTS

List of Figures ........................................................................................................................................ iv

Acknowledgements ................................................................................................................................... vi

Chapter 1 Introduction .............................................................................................................................. 1

What are the IST Building and the College of IST? ................................................................. 1

*Origin Story* ........................................................................................................................................... 2

*How a School Became a College* ............................................................................................................ 2

Why Does Classroom Availability in the IST Building Matter? ..................................................... 3

*Specialized Software* .............................................................................................................................. 3

*Group Meetings* ..................................................................................................................................... 3

*Special Privileges* .................................................................................................................................. 4

The Problem .............................................................................................................................................. 4

*Schedule of Courses* ................................................................................................................................. 4

*Classroom Schedule* ................................................................................................................................. 5

Why Does Solving this Problem Matter? ......................................................................................... 7

Research Objectives .................................................................................................................................. 8

*Scope* ...................................................................................................................................................... 8

*Hypothesis* .............................................................................................................................................. 8

*Method* ................................................................................................................................................... 8

*Outline* .................................................................................................................................................... 9

Chapter 2 Literature Review ................................................................................................................. 10

Who are the Potential Users of ClassroomsUnlocked? .............................................................. 10

*Experience and Gender* ......................................................................................................................... 11

Anatomy of the IST Building .................................................................................................................... 11

*Ownership* ............................................................................................................................................ 13

*Layout* ................................................................................................................................................... 14

*Relevance to ClassroomsUnlocked* ......................................................................................................... 15

Spatial Design Influences ....................................................................................................................... 16

*Two-dimensional or Three-dimensional?* ............................................................................................. 16

*Visibility, Minimalism, and Feedback* ................................................................................................. 18

Temporal Design Influences ..................................................................................................................... 19

*Traditional Temporal Interfaces* ........................................................................................................... 19

*Time Slider Interface* ............................................................................................................................... 19

User-centered Requirements Modeling ............................................................................................... 20

*Personas* ................................................................................................................................................. 20

*Scenarios* ............................................................................................................................................... 21

*User Stories* ........................................................................................................................................... 23
<table>
<thead>
<tr>
<th>Chapter 3 Method</th>
<th>Chapter 4 Analysis</th>
<th>Chapter 5 Discussion</th>
<th>Appendix A</th>
<th>Appendix B</th>
<th>Appendix C</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requirements Model</td>
<td>Keystroke-level Model</td>
<td>Results</td>
<td>Source Code for DownloadSchedule.java</td>
<td>Source Code for ScheduleToSqlite.pl</td>
<td>Source Code for ClassroomsUnlocked</td>
</tr>
<tr>
<td>Personas</td>
<td>Efficiency of the Primary Scenario</td>
<td>Conclusion</td>
<td>occupied.php</td>
<td>room.php</td>
<td>index.php</td>
</tr>
<tr>
<td>Scenarios</td>
<td>Efficiency of the Secondary Scenario</td>
<td>Recommendations for Future Research</td>
<td>SpatiotemporalUi.css</td>
<td>Main.js</td>
<td>IstBuilding.js</td>
</tr>
<tr>
<td>User Stories</td>
<td>Efficiency of the Tertiary Scenario</td>
<td></td>
<td>Time.js</td>
<td>Room.js</td>
<td>Time.js</td>
</tr>
<tr>
<td>User Interface Design</td>
<td></td>
<td></td>
<td>TypeCheck.js</td>
<td>TimeOfDaySlider.js</td>
<td>SlidingParagraph.js</td>
</tr>
<tr>
<td>Data Model</td>
<td></td>
<td></td>
<td>DownloadSchedule (Source Code in Appendix A)</td>
<td>ScheduleToSqlite (Source Code in Appendix B)</td>
<td>ClassroomsUnlocked (Source Code in Appendix C)</td>
</tr>
<tr>
<td>Implementation</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Screenshots</td>
</tr>
</tbody>
</table>

REFERENCES .............................................................................................................. 95
# LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Figure 1.</td>
<td>The IST Building at Twilight (“IST building,” 2012)</td>
<td>1</td>
</tr>
<tr>
<td>Figure 2.</td>
<td>Original Concept Sketch of the IST Building (“IST building,” 2009)</td>
<td>2</td>
</tr>
<tr>
<td>Figure 3.</td>
<td>IST Page on the Schedule of Courses</td>
<td>5</td>
</tr>
<tr>
<td>Figure 4.</td>
<td>Classroom Schedule for 110 IST Building</td>
<td>6</td>
</tr>
<tr>
<td>Figure 5.</td>
<td>IST vs. SRA</td>
<td>10</td>
</tr>
<tr>
<td>Figure 6.</td>
<td>The Cybertorium or 113 IST Building (“Cybertorium: Room layout,” 2009)</td>
<td>12</td>
</tr>
<tr>
<td>Figure 7.</td>
<td>110 IST Building</td>
<td>12</td>
</tr>
<tr>
<td>Figure 8.</td>
<td>202 IST Building</td>
<td>13</td>
</tr>
<tr>
<td>Figure 9.</td>
<td>IST-owned vs. ITS-owned Classrooms</td>
<td>14</td>
</tr>
<tr>
<td>Figure 10.</td>
<td>1st Floor of the IST Building [rooms from center: 110]</td>
<td>15</td>
</tr>
<tr>
<td>Figure 11.</td>
<td>2nd Floor of the IST Building [rooms from left to right: 202, 203, 205, 206, 208, 210, and 113]</td>
<td>15</td>
</tr>
<tr>
<td>Figure 12.</td>
<td>Two-dimensional Interface of the Penn State Lab Locations Map</td>
<td>17</td>
</tr>
<tr>
<td>Figure 13.</td>
<td>Two-dimensional Map of the IST Building (“IST building map,” 2013)</td>
<td>18</td>
</tr>
<tr>
<td>Figure 14.</td>
<td>Persona Format Used by Microsoft, Inc.</td>
<td>21</td>
</tr>
<tr>
<td>Figure 15.</td>
<td>Three Example Scenarios</td>
<td>22</td>
</tr>
<tr>
<td>Figure 16.</td>
<td>Examples of User Stories</td>
<td>23</td>
</tr>
<tr>
<td>Figure 17.</td>
<td>Early Wireframe</td>
<td>34</td>
</tr>
<tr>
<td>Figure 18.</td>
<td>Final User Interface</td>
<td>35</td>
</tr>
<tr>
<td>Figure 19.</td>
<td>ER Diagram</td>
<td>37</td>
</tr>
<tr>
<td>Figure 20.</td>
<td>Output of DownloadSchedule</td>
<td>40</td>
</tr>
<tr>
<td>Figure 21.</td>
<td>Server-side Files</td>
<td>45</td>
</tr>
<tr>
<td>Figure 22.</td>
<td>Client-side Files</td>
<td>48</td>
</tr>
<tr>
<td>Figure 23.</td>
<td>Screenshot of First Floor of IST Building</td>
<td>49</td>
</tr>
<tr>
<td>Figure 24.</td>
<td>Screenshot of Second Floor of IST Building</td>
<td>49</td>
</tr>
</tbody>
</table>
Figure 25. Screenshot of Clicking on Status of 110 IST Building on a Tuesday .................50

Figure 26. Screenshot of Day of Week Slider and Time Slider........................................50

Figure 27. Screenshot of Entire User Interface..................................................................51

Figure 28. KLM Operation Taxonomy..............................................................................53
ACKNOWLEDGEMENTS

This thesis would not be possible without the support and guidance of others. My honors thesis adviser Dr. Cai has helped me every step of the way and has provided me with invaluable guidance. My honors adviser Dr. Zhang has helped me refine my vision. Special thanks go to my mother, my father, and my brother for their emotional support throughout this endeavor.
Chapter 1

Introduction

What are the IST Building and the College of IST?

Figure 1. The IST Building at Twilight (“IST building,” 2012)

The Information Sciences and Technology (IST) Building at the Pennsylvania State (Penn State) University bestrides Atherton Street in University Park, Pennsylvania like the Ponte Vecchio spans the Arno River in Florence, Italy. That was the unique vision of architects Rafael Vinoly and Perfido Weiskopf, whom Penn State hired to design the 199,000 square-foot home of the burgeoning School of IST (“IST history,” 2013).
Origin Story

In 1997, Penn State President Graham Spanier convened a 15-member strategic committee to create a revolutionary 43-course curriculum intended to address the Information Age. This curriculum led to foundation of the School of IST in 1999. With the addition of the School of IST, Penn State joined the prestigious 41-member iSchools directory, which recognizes universities, such as Carnegie Mellon University and Humboldt-Universität zu Berlin, for providing quality education in the information sciences (“iSchools directory,” 2013). By the time the IST Building opened its doors in 2004, the School of IST had just graduated its first full class of undergraduate students (“IST history,” 2013).

How a School Became a College

In 2006, the Penn State Board of Trustees deemed the school worthy of greater distinction and renamed it the College of IST (“IST history,” 2013). Over time, the College of IST has become a vast, interconnected institution encompassing twenty campuses across the Commonwealth of Pennsylvania with an award-winning online education program. Today, the College of IST offers three undergraduate degree programs, three professional masters programs, one M. S. program, and one Ph. D. program. In 2010, the college had 120 faculty members, 2,400 undergraduate students, 289 graduate students, and over $60 million of cumulative research funding (“About IST,” 2013).
Why Does Classroom Availability in the IST Building Matter?

The IST Building is unique among academic buildings at Penn State. In other buildings, there are separate spaces for classrooms and computer labs. In the IST Building, the classrooms are the computer labs. In other academic colleges, computers simply enhance the learning experience. In the College of IST, computers are the learning experience. Students require computers to complete assignments and study for exams (“Information sciences and technology,” 2009). Therefore, student access to these classrooms is a non-trivial matter. Admittedly, some students do own personal computing devices, which may grant them some degree of self-reliance and independence. However, there are still three compelling reasons why students care about the availability of these classrooms: specialized software, group meetings, and special privileges.

Specialized Software

The computers in these classrooms contain specialized software systems that cater specifically to the needs of these students. For example, the course IST 210, an “introduction to [the] concept of databases”, requires the use of a professional database management software system such as MySQL or Microsoft SQL Server (“IST 210 organization of data,” 2005). While MySQL is open-source and free-to-download (“Top reasons to use MySQL,” 2013), Microsoft SQL Server is proprietary, and only the client is free-to-download (“How to buy,” 2013). IST classroom computers have both MySQL and Microsoft SQL Server available for use.

Group Meetings

Second, the College of IST aims to “engage students in sharpening their abilities to think critically and to work in teams” (“Information sciences and technology,” 2009). This penchant for collaborative learning has merit. In February 2010, Forbes Magazine listed “ability to play well with others” as one of “the seven most universal job skills” (Dempsey, B., 2010). Team projects require group meetings. The most convenient place to meet is the IST Building
because students are already required to visit this building for class. Within the IST Building, the most convenient place to work on team projects is a classroom where all group members are guaranteed to have access to the proper resources, such as computers, desks, and whiteboards.

**Special Privileges**

Students enjoy special privileges in IST-owned classrooms. Classroom computers are on the IST network, which provides specialized software and services. Card swipes at the classroom doors guarantee them exclusive access. Also, printing is free in these classrooms, a rare privilege at Penn State ("Sheets and pricing," 2013).

**The Problem**

Students’ classroom availability information seeking behavior as mediated by the existing information technology landscape is limited. Granted, Penn State does maintain publicly-available data on all class meeting times. However, Penn State is neither extensively connecting nor comprehensively aggregating these records based upon their geographic location. Students need class meeting times to be geocentrically connected and aggregated so that they can infer where the available classrooms are located. These issues manifest themselves in the design of the Schedule of Courses and the Classroom Schedule.

**Schedule of Courses**

The Schedule of Courses is a publicly-available website used by students when scheduling classes to find the class meeting times that best align with their needs. After a student selects a subject abbreviation (i.e. IST), the website enumerates all the courses within that subject area as well as all the class meeting times for each course. Associated with each class meeting time is a hyperlink to a Google Maps rendering of the building where that meeting occurs. This connection is shallow because it does not indicate the geographic location of the classroom relative to the building. Also, class meetings times can only be aggregated based upon course constraints or temporal constraints. The ability to aggregate based upon geographic constraints is
important because colleges other than the College of IST schedule classes in the IST Building. Without geocentric constraints, students must to manually search through all 216 subject abbreviations in order to prove the availability of a single classroom in the IST Building beyond any reasonable doubt (“Schedule of courses,” 2013). This is a Herculean task.

![Figure 3. IST Page on the Schedule of Courses](image)

**Classroom Schedule**

By posting a sheet of paper alongside each classroom door, the IST Building has made its own attempt at geocentrically connecting and aggregating class meeting times. Known colloquially as the Classroom Schedule, this paper-based system lists all the class meeting times that occur within a particular classroom in chronological order. However, this approach makes two unrealistic assumptions about students’ information seeking behavior. First, it assumes that students are in the geographic vicinity of a classroom door. Second, it assumes that students only care about the availability of a single classroom. Students’ real information seeking behavior is much more dynamic and complex. The IST Building is 199,000 square feet, and students are
constantly moving in and out of the building. The chances of a student conveniently being at the right geographic location at the right time are small. Also, students need to assimilate information from multiple classrooms in order to make meaningful decisions. Although a student may pick a single classroom over and over again, the underlying context of this behavior is still an implicit decision making process where multiple classrooms are evaluated and the single classroom is once again validated as the best choice.

Figure 4. Classroom Schedule for 110 IST Building
Why Does Solving this Problem Matter?

Students ask a relevant question: “During the limited time I have, where may I go in the IST Building to complete my assignments and study for my exams?” In the field of Geographic Information Science (GIS), this space-time query may be rephrased in a more general manner: “Given a certain temporal range, what are the spatial objects for which a certain condition holds true?” According to Yuan and McIntosh, this request for a simple snapshot of spatial objects is the most fundamental of the four types of spatiotemporal queries (2002). Spatiotemporal research has been rather prolific in recent years. However, most studies have focused upon the movement over time of spatial objects through a large-scale outdoor space (Frank, Grumbach, Gueting, Jensen, Koubarakis, Lorentzos, ... & Widmayer, 1999; Kim, Ryu, & Kim, 2000; Shaw, 2000; Yu & Shaw, 2008).

Room availability, an indoor version of the same general problem, has been of interest to GIS designers for years. Microsoft Outlook, an email client popular with many large corporations, includes a plug-in called RoomFinder that allows employees to find the names and attributes of available conference rooms and reserve the conference room that best suits their needs (“Schedule a meeting with other people,” 2013). RoomFinder results are often physical realized and visualized via an interactive conference room sign, such as MeetingMinder by Visix (Visix, Inc., 2013). In 1999, Eduardo Fernandez and Xiaohong Yuan of Florida Atlantic University formally identified a general design pattern for hotel reservation systems. One example of such a system is called eZee FrontDesk (Technosys Pvt. Ltd., 2013). Researchers are even exploring new mathematical programming approaches to schedule emergency rooms in less than a day (Beaulieu, Ferland, Gendron, & Michelon, 2000).

Students in the IST Building face a unique version of the room availability problem. While many universities have various solutions for computer lab availability, library study room reservations, and meeting room reservations, no university provides a comprehensive and
satisfying solution for classroom availability (Bohus & Rudnicky, 2003; Group study rooms,” 2013; “Lab locations,” 2013; “Penn state reservation system,” 2013). The closest solution, a “mobile indoor location-based GIS application” developed for the GIS Department of the British Columbia Institute of Technology, addresses classroom navigation, not classroom availability (Candy, 2007). By solving this novel problem, we reexamine spatiotemporal theories in a new light. The results will inform the design of future systems in both university and non-university settings.

**Research Objectives**

**Scope**

The scope of this thesis is limited to undergraduate students in the College of IST. For the purposes of this thesis, undergraduate students in the College of IST include both students pursuing a bachelor’s of science in Information Sciences and Technology (IST) and students pursuing a bachelor’s of science in Security Risk and Analysis (SRA). All other students at Penn State and at other academic institutions will be ignored.

**Hypothesis**

I hypothesize that the geocentric integration and presentation of classroom availability information in the IST Building will lead to more effective information seeking and sense making in a range of common student activities.

**Method**

I will test my hypothesis by first gathering and modeling requirements that reflect the range of common student activities. Second, I will use these requirements to design an innovative software prototype called ClassroomsUnlocked. The design of this software prototype should quickly demonstrate whether or not my research hypothesis is worth pursuing beyond the prototype phase. Finally, I will use the popular Keystroke-level Model (KLM) originally proposed by Stuart K. Card, Thomas P. Moran, and Allen P. Newell in 1980 to estimate the
effectiveness of ClassroomsUnlocked, using the existing Classroom Schedule paper-based system as a control. Grounded in years of human-computer interaction research, KLM is a straightforward method for estimating the time in seconds needed to complete simple data input tasks using a computer and a mouse.

Outline

In Chapter 2, we will review relevant literature that influenced the design of ClassroomsUnlocked. In Chapter 3, we will discuss the requirements and design decisions that ultimately led to the implementation of the software prototype. In Chapter 4, we will use a KLM analysis to estimate the effectiveness of the prototype as compared to the existing paper-based system. In Chapter 5, we will discuss the results, conclude, and recommend future research.
Who are the Potential Users of ClassroomsUnlocked?

As stated previously, the potential users of ClassroomsUnlocked are undergraduate students in the College of IST who are interested in searching classroom availability in the IST Building. To qualify as an undergraduate student in the College of IST, a student must be majoring in Information Sciences and Technology (B.S.) or Security and Risk Assessment (B.S.) or both. This thesis intentionally excludes students who are only minoring in IST or SRA because these students are assumed to be only weakly invested in the IST Building and to have complex ulterior motives not directly related to the research question. In terms of classroom availability, the differences between IST and SRA majors are negligible because these students use the exact same classrooms and have access to the exact same resources within those classrooms. The differences exist solely in the curriculum.

<table>
<thead>
<tr>
<th>Information Sciences and Technology (B.S.)</th>
<th>“This major is structured to provide students with the theoretical frameworks and skill sets necessary to compete and be productive in the information technology-intensive global context that defines the new ‘Information Age’” (“Information sciences and technology,” 2009).</th>
</tr>
</thead>
<tbody>
<tr>
<td>Security Risk and Analysis (B.S.)</td>
<td>“The Bachelor of Science in Security and Risk Analysis (SRA) in the College of Information Sciences and Technology is intended to familiarize students with the general frameworks and multidisciplinary theories that define the area of security and related risk analyses” (“Security risk and analysis,” 2011).</td>
</tr>
</tbody>
</table>

Figure 5. IST vs. SRA
Experience and Gender

The primary differences between user groups are experience and gender. Most students interested in a system like ClassroomsUnlocked tend to have extensive experience with the IST Building. Freshmen students tend to be too overwhelmed with learning to navigate the IST Building to demonstrate a strong interest in searching classroom availability. Sophomore, junior, and senior students, who have already mastered navigation, are much more likely to develop a strong interest in searching classroom availability. Gender plays a lesser role. Spatial reasoning is crucial to understanding the spatial component of classroom availability information. On average, researchers have observed that men tend to perform slightly better at spatial reasoning tasks than women (Kimura, 1992). However, the effect size is small and may be attributed to the placebo effect of persistent cultural stereotypes instead of a real physiological difference (McGlone & Aronson, 2006). Whatever the case may be, the implications are still significant. In 2000, Penn State enrolled 53.4% men and 46.6% women (Rowe, 2004). It is assumed that this enrollment pattern has remained fairly stable over the years. Within the College of IST, the percentage of male students is much higher than the University average as evidenced by the existence of an organization called Women in IST (WIST), which promotes the cause of a female minority (“Women in IST,” 2013).

Anatomy of the IST Building

The eight classrooms available to undergraduate students in the College of IST are each outfitted with “state-of-the-art telecommunications and multimedia infrastructure to meet the unique needs of today’s digital students.” The crown jewel of these classrooms is the Cybertorium, an auditorium for cyberspace. The Cybertorium, also known as 113 IST Building, seats 150 students and is the only auditorium at Penn State to have computers built into the desks. All classrooms have desktop computers, except 110 IST Building, which is the only laptop classroom. Online students can virtually attend classes physically held in 202 IST Building.
thanks to a custom-built video recording system that follows the instructor and captures the voices of non-online students (“IST history,” 2013).

Figure 6. The Cybertorium or 113 IST Building (“Cybertorium: Room layout,” 2009)

Figure 7. 110 IST Building
Classroom ownership is a matter of politics. As a general rule of thumb, the College of IST owns all classrooms east of Atherton Street, and the School of Computer Science and Engineering (CSE) owns all classrooms west of Atherton Street. Thus, the IST Building is literally a bridge between the College of IST and the College of CSE, two highly related disciplines. Penn State Information Technology Services (ITS) has taken ownership of three classrooms in the IST Building: 203 IST Building, 210 IST Building, and the Cybertorium. These classrooms may be used by any Penn State institution, and all Penn State students have access to them. Of the eight classrooms available to undergraduate students in the College of the IST, the college only has exclusive ownership rights to five classrooms: 110 IST Building, 202 IST Building, 205 IST Building, 206 IST Building, and 208 IST Building.

**Figure 8. 202 IST Building**

**Ownership**

<table>
<thead>
<tr>
<th>IST-owned Classrooms</th>
<th>1st Floor</th>
</tr>
</thead>
<tbody>
<tr>
<td>• IST network</td>
<td>• 110 IST Building</td>
</tr>
<tr>
<td>• card swipes</td>
<td></td>
</tr>
<tr>
<td>ITS-owned Classrooms</td>
<td>2nd Floor</td>
</tr>
<tr>
<td>------------------------------------------</td>
<td>----------------------------</td>
</tr>
<tr>
<td>• free printing</td>
<td>• 202 IST Building</td>
</tr>
<tr>
<td></td>
<td>• 205 IST Building</td>
</tr>
<tr>
<td></td>
<td>• 206 IST Building</td>
</tr>
<tr>
<td></td>
<td>• 208 IST Building</td>
</tr>
<tr>
<td>• Penn State network</td>
<td>• 113 IST Building</td>
</tr>
<tr>
<td>• no card swipes</td>
<td>• 203 IST Building</td>
</tr>
<tr>
<td>• printing fees</td>
<td>• 210 IST Building</td>
</tr>
</tbody>
</table>

Figure 9. IST-owned vs. ITS-owned Classrooms

**Layout**

The classroom layout is self-explanatory. 100-level classrooms are on the first floor, and 200-level classrooms are on the second floor. The only exception is the Cybertorium, which is labeled as 113 IST Building but has an entrance on both floors. The primary entrance is on the second floor. I created the following two images using Google Earth and a three-dimensional model of the IST Building provided by Guoray Cai in order to illustrate the layout of the eight classrooms available to undergraduate students in the College of IST.
The anatomy of the IST Building directly influenced the design of ClassroomsUnlocked. It was not enough to simply solve the general problem that students face. Not every building and
not every classroom is exactly alike. This solution needed to understand and adapt to the specific idiosyncrasies of the students’ environment.

**Spatial Design Influences**

The design of ClassroomsUnlocked’s spatial interface was influenced by prior art as well as the well-tested claims of two influential authors in the field of human-computer interaction: Donald A. Norman and Jakob Nielsen.

*Two-dimensional or Three-dimensional?*

One of the most important decisions when designing a spatial interface is whether to support a two-dimensional (i.e. x and y) or three-dimensional (i.e. x, y, and z) interface. It is especially important that ClassroomsUnlocked has a strong spatial interface because many existing state-of-the-art room availability systems tend to have very poor spatial interfaces. Microsoft Outlook’s RoomFinder has no spatial interface beyond cognitive inferences that can be made based upon the names of rooms (“Schedule a meeting with other people,” 2013). The Penn State Reservation System has a spatial blueprint for the inside of most rooms but no representation of the position of these rooms relative to each other (“Penn state reservation system,” 2013). The Penn State Lab Locations Map is perhaps the best example of a spatial interface. This system supports a two-dimensional Google Maps interface with clickable operating system icons for every building that contains computer labs (“Lab locations,” 2013).
Ultimately, however, ClassroomsUnlocked did not choose a two-dimensional interface. Instead, it pursued a three-dimensional interface in order to help users orient themselves within the IST Building. As Norman explained in his 1988 book *The Design of Everyday Things*, matching the users’ mental model is the key to designing usable systems that avoid the debilitating Gulf of Execution and Gulf of Evaluation. The primary user group (i.e. experienced students) already has a well-defined three-dimensional mental model of the structure of the IST Building. People naturally perceive buildings in three dimensions. While a two-dimensional interface clashes with the users’ mental model and increases the users’ cognitive burden, a three-dimensional interface empowers users to superimpose their own memories right into the virtual three-dimensional world. In the words of Nielsen in his 1993 book *Usability Engineering*, the system should match the real world. This phenomenon can be observed in everyday life when people can easily navigate through an area from memory but struggle to orient themselves within a two-dimensional map of that same area.
Visibility, Minimalism, and Feedback

A three-dimensional interface introduces an entire new dimension of complexity. Therefore, it is important to minimize the complexity and prevent unnecessary details from occluding the overall visibility of the design features. Nielsen stressed that the system status should always be visible and that the design should pursue aesthetically pleasing and minimalist goals (Nielsen & Hackos, 1993). Therefore, the design of ClassroomsUnlocked takes a rather minimalist approach inspired by the design of the Penn State Lab Locations Map (“Lab locations,” 2013). Instead of realistically modeling every single detail of the IST Building, ClassroomsUnlocked displays a bare-bones model of the IST Building with all the ceilings removed. The availability state of each classroom is represented with simplicity and high visibility as a transparent square covering the ceiling of each classroom, glowing green when available and red when occupied. The structure of the three-dimensional model is just detailed enough to be recognizable but not too much to be distracting. Norman argued that the feedback loop was crucial to good system visibility. Well-designed systems should provide instant feedback so that users understand the consequences of their actions. ClassroomsUnlocked’s floor
selector interface provides immediate feedback by removing or adding floors from the three-dimensional model as soon as a new floor is selected. There is no submission or confirmation button. It all happens instantaneously.

Temporal Design Influences

Prior art and the work of Donald A. Norman and Jakob Nielsen also influenced the design of ClassroomUnlocked’s temporal interface.

Traditional Temporal Interfaces

Microsoft Outlook’s RoomFinder sets the general standard for temporal interfaces. The user operates a standard calendar picker and a standard time picker in order to select a start time and an end time. RoomFinder’s temporal interface also suggests additional time ranges to explore (“Schedule a meeting with other people,” 2013). The Penn State Reservation System has a similar temporal interface but also introduces the concept of temporal recurrences, which allows users to define time ranges that repeat periodically (“Penn state reservation system,” 2013). The Penn State Lab Locations Map has perhaps the worst temporal interface. After selecting a computer lab, the system simply displays a list of all the times when the lab is available for general use (“Lab locations,” 2013).

Time Slider Interface

The design of ClassroomUnlocked’s temporal interface was based upon the traditional temporal interface. The concept of temporal recurrences was used because all class meeting times repeat on a weekly basis. Following Nielsen’s minimalism heuristic, the interface was simplified into a day of the week component and a time range component with no complicated calendars (Nielsen & Hackos, 1993). One of Norman’s most important ideas was perceived affordances. Good interfaces should naturally suggest to users all the actions that can be performed (Norman, 1988). Therefore, both the day of the week component and the time range component were designed as “time sliders” with prominent handles that afforded grasping and
sliding. Each slider was oriented horizontally in order to match the users’ mental model that time moves from left to right. Graduated measurements were strategically placed along the slider in order to create a scale for better visibility. Also, instant feedback was provided by having the value of the slider float above the handle or above the space between two handles.

**User-centered Requirements Modeling**

To promote usability, ClassroomsUnlocked gathered and modeled requirements according to Norman’s user-centered design principles (Norman, 1988). Specifically, ClassroomsUnlocked employed three user-centered requirements models: the persona, the scenario, and the user story.

*Personas*

In 1999, Alan Cooper, widely regarded as the “father of Visual Basic,” wrote *The Inmates Are Running the Asylum: Why High-Tech Products Drive Us Crazy and How to Restore the Sanity*. This book was highly influential in introducing the concept of personas to user-centered design. According to Cooper, a persona is a fictional character who represents a particular type of user within a target demographic. Despite being a single character, a persona is designed to reflect the hypothesized goals and behaviors of a large group of users. Personas are typically expressed by means of a short description of the character’s goals, behaviors, skills, attitudes, and environment. Often, multiple personas are created. However, there should be only one primary persona.

In his book, Cooper recommended the use of personas in order to solve what he called the “elastic user” problem. In the elastic user problem, different stakeholders keep stretching the definition of the user to satisfy their own personal agendas instead of the needs of the real user. Personas provide a firmer definition of the user that better reflects reality. Personas provide a method for prioritizing design features and a human face with which designers can empathize. Personas also help designers focus on the primary use cases that are most likely to occur instead
of the edge cases. Cooper argued that edge cases should never become the focus of user-centered design. The figure below demonstrates a persona format used by Microsoft, Inc. (Pruitt & Grudin, 2003).

![Persona Format Used by Microsoft, Inc.](image)

**Figure 14.** Persona Format Used by Microsoft, Inc.

**Scenarios**

If personas are biographies or back-stories, scenarios are the main narrative. More specifically, a scenario is a story that describes and predicts the interactions between the users (also known as actors) and the system. Each scenario has a goal. Scenario goals usually represent functional requirements (i.e. what the system does).
However, “negative” scenarios, which describe threats to the system, can be written to explore non-functional requirements (i.e. how the system does what it does) such as security, safety, and reliability. Scenarios are written in simple, non-technical language with minimal details so that all stakeholders can understand and discuss them. The figure below demonstrates three example scenarios for “a university student attending a club meeting online” (Rosson & Carroll, 2002).

<table>
<thead>
<tr>
<th>A. Science Fiction Club in a Web forum</th>
</tr>
</thead>
<tbody>
<tr>
<td>After three years at Virginia Tech, Sharon has learned to take advantage of her free time in-between classes. In her hour between her morning classes, she stops by the computer lab to visit the science fiction club. She has been meaning to do this for a few days because she knows she'll miss the next meeting later this week. As she opens a Web browser, she realizes that this computer will not have her bookmarks stored, so she starts at the homepage of the Blacksburg Electronic Village. She sees local news and links to categories of community resources (businesses, town government, civic organizations). She selects “Organizations”, and sees an alphabetical list of community groups. She is attracted by a new one, the Orchid Society, so she quickly examines their Web page before going back to select the Science Fiction Club page. When she gets to the club page, she sees that there are two new comments in the discussion on Asimov’s Robots and Empire, one from Bill and one from Sara. She browses each comment in turn, then submits a reply to Bill’s comment, arguing that he has the wrong date associated with discovery of the Zeroth Law.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>B. Science Fiction Club in a Community MOO</th>
</tr>
</thead>
<tbody>
<tr>
<td>After three years at Virginia Tech, Sharon has learned to take advantage of her free time in-between classes. In her hour between her morning classes, she stops by the computer lab to visit the science fiction club. She has been meaning to do this for a few days because she knows she'll miss the next meeting later this week. As she starts up the Blacksburg community MOO, she can see that the last person using this computer must have been interested in orchids, because the welcoming text describes her location as an orchid garden, along with Penny and Alicia, who are discussing some new exotic varieties. The text description mentions an exit to Main Street, so she leaves the garden and starts moving south. Along the street she runs into George, who is working on a banner for the fair. She gives him a quick hello, and continues southward until she sees an eastward exit will take her to Eastenders Pub; this is where the Science Fiction Club meets. She enters the room and is told that Bill and Sara are already there, along with a pitcher of Newcastle Brown. She can tell from their current comments that they have been discussing the timeline from Asimov’s Robots and Empire.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>C. Science Fiction Club in a Collaborative Virtual Environment</th>
</tr>
</thead>
<tbody>
<tr>
<td>After three years at Virginia Tech, Sharon has learned to take advantage of her free time in-between classes. In her hour between her morning classes, she stops by the computer lab to visit the science fiction club. She has been meaning to do this for a few days because she knows she'll miss the next meeting later this week. When she tries to start up the online collaborative environment, she finds that this computer does not have the client, so she waits for a minute or two while it is automatically downloaded and installed. After she logs in, she is taken back to her previous visit location, and sees the familiar panoramic view of her living room, her to-do lists and sketchpad, and the interactive map of Blacksburg. She positions and zooms in on the map until she can see downtown buildings. She enters the Eastenders Pub subspace, where the science fiction club usually meets. She sees a panoramic image of bar, faces that show Bill and Sara are here, a food and drink menu, and various standard tools. The map updates to show a layout of the Pub—the dining room, the dance area, the office, and the bar. Bill and Sara are using a chat tool and a shared whiteboard to sketch an event timeline for Asimov’s Robots and Empire. Joining Bill and Sara in the chat tool, she types “Based on the Zeroth Law, I’m afraid I must drink some of your beer.”</td>
</tr>
</tbody>
</table>

Figure 15. Three Example Scenarios
User Stories

Scenarios can take many different forms. One of the simplest forms is the user story. A user story is a sentence or two in the everyday language of the user that describes something the user does or needs to do. A user story is typically phrased as a single sentence, “As a <insert role here>, I want <insert goal or desire here> so that <insert benefit here>.” In general, user stories are designed to express who, what, and why concisely on a small handwritten notecard. Each user story serves as a reminder for planning purposes, a conversation starter for implementation purposes, and acceptance criteria for testing purposes. The figure below demonstrates several examples of user stories (Cohn, 2004).

<table>
<thead>
<tr>
<th>As a/an</th>
<th>I want to...</th>
<th>so that...</th>
</tr>
</thead>
<tbody>
<tr>
<td>moderator</td>
<td>create a new game by entering a name and an optional description</td>
<td>I can start inviting estimators</td>
</tr>
<tr>
<td>moderator</td>
<td>invite estimators by giving them a url where they can access the game</td>
<td>we can start the game</td>
</tr>
<tr>
<td>estimator</td>
<td>join a game by entering my name on the page I received the url for</td>
<td>I can participate</td>
</tr>
<tr>
<td>moderator</td>
<td>start a round by entering an item in a single multi-line text field</td>
<td>we can estimate it</td>
</tr>
<tr>
<td>estimator</td>
<td>see the item we’re estimating</td>
<td>I know what I’m giving an estimate for</td>
</tr>
<tr>
<td>estimator</td>
<td>see all items we will try to estimate this session</td>
<td>I have a feel for the sizes of the various items</td>
</tr>
<tr>
<td>moderator</td>
<td>see all items we try to estimate this session</td>
<td>I can answer questions about the current story such as &quot;does this include...&quot;</td>
</tr>
<tr>
<td>moderator</td>
<td>select an item to be estimated or re-estimated</td>
<td>the team sees that item and can estimate it</td>
</tr>
</tbody>
</table>

Figure 16. Examples of User Stories
This lightweight approach was originally inspired by the “Manifesto for Agile Software Development” signed by Beck et al. in 2001. The signers of the manifesto, all veteran software developers, valued “working software over comprehensive documentation” and “responding to change over following a plan.” Agile software development methodologies use quick iterations with lightweight documentation to meet the ever-changing needs of customers in the fast-paced software development industry.

Extreme Programming (XP), a particularly “extreme” agile software development methodology, was the first to write user stories (Cohn, 2004). XP developed user stories in stark contrast to the traditional Unified Process, which recommended use cases. Use cases are much more verbose way to express scenarios. In the Unified Modeling Language (UML), there is a diagram called the use case diagram. XP rebelled against the Unified Process and the concept of use cases because the signers valued, “individuals and interactions over processes and tools.” Use cases formally describe a process and its steps in minute detail often with little open to interpretation. User stories provide a small-scale, easy-to-use presentation of the same information that is open to innovative interpretation and discussion. Use cases seek to provide extensive details that clearly outline all the potential flows through the system. Meanwhile, user stories seek to provide just enough details so that software developers clearly understand the user’s goal but can implement the solution however they see fit. ClassroomsUnlocked is a perfect candidate for user stories for three reasons. First, the scale is relatively small. Second, the needs of the user are quickly changing as old students graduate and new students matriculate. Finally, it is a prototype that demands quick iteration instead of slow, careful planning (Cohn, 2004).
Chapter 3

Method

Requirements Model

The first step of designing of ClassroomsUnlocked was modeling the user requirements. User requirements were gathered through informal interviews with students in the College of IST and through my personal experiences as a student in the College of IST. Requirements were then formally addressed by means of the three effective user-centered design methods for modeling user requirements: the persona, the scenario, and the user story.

Personas

The primary persona chosen was a female junior majoring in IST. This persona represents a wealth of experience and promotes a healthy designer empathy toward female students in the minority. The secondary persona chosen was a male freshman majoring in SRA. This persona represents inexperience. The tertiary persona chosen was a male senior majoring in IST. This persona represents the maximum level of experience. The exclusionary persona chosen was a female junior majoring in Meteorology and minoring in IST. This persona represents students with minors in IST and SRA whose needs should be ignored. All four personas are listed below.

**Primary Persona**

Name: Emily Expert  
Gender: Female  
Age: 21  
Year: Junior (i.e. year three)  
Major: Information Sciences and Technology (B.S.)  
Minor: English  
Goals: To be gainfully employed after graduation. To pursue the career of my choice. To be treated fairly and equally with respect to my male colleagues. To gain academic knowledge and wisdom.
To earn the highest grades I can.
To get involved.
To make new friends.
To have fun.

My Story: Hello! My name is Emily Expert. I just turned 21 years old last month. This is my third year as an Information Sciences and Technology student at Penn State University, and I love it here. When I was little, I used to play for hours and hours with a toy computer my parents gave me for my birthday. Ever since then, I have been very passionate about computer technology. My mother went to Penn State, and I chose IST because I know that IST will help me find a great high-paying job after graduation. My parents and I are very glad that IST has an internship requirement. My little brother, who used to steal my toy computer, is considering IST as well. He still has two years of high school left. I have to admit it is a little intimidating being one of the few girls in class, yet I love rising to the challenge. I am glad for the existence of Women in IST. I am also glad that the gender ratio is gradually shifting as more women enter the field. The IST Building is my favorite place in the whole world for doing school work. Many of my friends in other majors prefer the peace and quiet of the library, but I prefer the fellowship and excitement of the IST Building. I enjoy commiserating with fellow IST and SRA majors over tough assignments and exams. Believe it or not, I have made lasting friendships and gotten involved in the IST Penn State Dance Marathon Club, which raises money to fight pediatric cancer, through these interactions. Most importantly, I appreciate the resources available to me in the classrooms. I have a laptop, but it is very heavy and difficult to carry from my apartment to the IST Building. Whenever I can, I use my student ID to swipe into a free classroom in the IST Building because there is always a computer available with all the software I need. Instead of lugging around my laptop, I just have a small USB flash drive in my purse. The free printing is a real lifesaver. As an English minor, I tend to print a lot of pages. Thanks to IST, I haven’t had to buy a printer for my apartment. As a member of the IST Diplomats Club, I know the IST Building like the back of my hand. I love giving tours of the building to prospective students. However, something really bugs me. No matter how much I learn about the IST Building, I have no idea when classrooms in the IST Building are available. There is something exhausting and slightly embarrassing about having to physically walk up to the sheet of paper next to each classroom and check the schedule only to discover that a class is already in progress. The worst is when the classroom has glass walls and all the students inside are staring at you. It makes you feel like such an idiot. There has to be a better way. After learning the Java programming language in IST 240, I’ve tried writing a program that solves this problem. I think my program needs to grab information from the schedule of courses website, but I have no clue how to do that. I am by no means an expert programmer. Also, I have a big IST exam coming up, so I do not have very much time either. In the meantime, I guess I’ll just have to memorize the schedule for eight classrooms. Maybe I’ll take pictures of all those sheets of paper. What a
low-tech solution! You would think the College of “Information” Sciences and Technology would be on top of this.

**Secondary Persona**

**Name:** Neal Novice  
**Gender:** Male  
**Age:** 19  
**Year:** Freshman (i.e. year one)  
**Major:** Security and Risk Analysis (B.S.)  
**Goals:** To figure out what I want to do with my life.  
To adjust to life on my own as a college student.  
To gain academic knowledge and wisdom.  
To earn the highest grades I can.  
To get involved.  
To make new friends.  
To have fun.

**My Story:** Hello! My name is Neal Novice. I am 19 years old, and this is my first year at Penn State University. I am an only child and the first person in my extended family to go to college. I chose Penn State because I felt right at home the minute I stepped on campus. I chose to major in Security and Risk Analysis because I am very interested in the growing presence of cyber-war and cyber-terrorism. We live in very exciting times, and I want to be on the forefront of this. I am struggling to figure out exactly what kind of career I want after college, but I still have plenty of time to decide. I hear SRA majors are highly sought after by government agencies like the FBI and the CIA, and I am very excited about that. The IST Building is really big and confusing. I got lost once trying to find 110 IST. It turns out it is way down on the first floor of the building. I had no idea there were even classrooms down there. I also ended up late to a class in the cybertorium in 113 IST because I forgot which side of the building it was on. Thankfully, no one noticed. I hear there’s a whole third floor to the building floating above everything else. I’ll have to explore that sometime. I think one of my professors holds office hours up there. I just joined the SRA Club. Maybe I can make friends with some of the older members, and they can answer my questions. Last week, I started noticing that every classroom has a sheet of paper next to it. It seems to be the classroom’s schedule. It kind of confuses me though. I’m not sure exactly how to read it. I wonder why the College of IST is using a paper-based system. Wouldn’t it make more sense for them to install a fancy computerized system for this? Am I allowed to use these classrooms when classes aren’t going on? I hear we get free printing as IST students, but I still haven’t taken advantage of that. Is free printing only available in these classrooms? Nevertheless, I’m much more concerned about not getting lost and making it to class on time. I can worry about silly classroom schedules later. I’ll just do all my homework and study for my exams in my dorm room with my laptop. It’s comfortable, and I always know exactly where it is. Well, at least most of the time.
<table>
<thead>
<tr>
<th><strong>Tertiary Persona</strong></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Name:</strong></td>
<td>Mike Oldtimer</td>
<td></td>
</tr>
<tr>
<td><strong>Gender:</strong></td>
<td>Male</td>
<td></td>
</tr>
<tr>
<td><strong>Age:</strong></td>
<td>22</td>
<td></td>
</tr>
<tr>
<td><strong>Year:</strong></td>
<td>Senior (i.e. year four)</td>
<td></td>
</tr>
<tr>
<td><strong>Major:</strong></td>
<td>Information Sciences and Technology (B.S.)</td>
<td></td>
</tr>
<tr>
<td><strong>Goals:</strong></td>
<td>To prepare for graduation.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>To prepare for my job after graduation.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>To make the most of my last year.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>To get involved.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>To make new friends.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>To have fun.</td>
<td></td>
</tr>
</tbody>
</table>

**My Story:**

Hello! My name is Mike Oldtimer, and, boy, do I feel like an old man. I am 22 years old, and this is my last semester as an Information Sciences and Technology student at Penn State University. My mother is a Penn State alumna, and my older brother recently graduated from Penn State with a degree in IST. I always tell him that I picked IST first because I got him interested in computer in the first place. I am excited to graduate, but I will miss this place. I’ve made so many lasting friendships and memories here. I have all my graduation requirements fulfilled and have accepted an offer to work full-time as an IT Analyst for Boeing in Seattle, Washington starting in late May. I can’t believe I’m already starting to look for an apartment in Seattle. Where has all the time gone? After four years in the IST Building, I have one major complaint, which I plan to voice at the Spring Senior Feedback Session at the end of the semester. My complaint is that I was never able to definitively figure out when classrooms in the building were available. I regret not taking advantage of this resource. If only there were a system to search classroom availability quickly and easily, I would have lived in the IST Building all day long. Instead, I was so afraid of getting awkward stares from a class in session that I usually only dared to enter a classroom when I was with a large group of students working on a group project. I remember that the classrooms in the IST Building were the perfect environment for work, and I really enjoyed working side by side with other IST and SRA majors. I am confused why the College of IST, a school devoted to the study of advanced computer technology, is using an old-school paper-based system to display its classroom schedules. Hopefully, the College of IST will listen and improve. Unfortunately, it’s far too late for an improved system to change my experience, but hopefully, this could revolutionize the experience for younger students who still have several years of assignments to complete and exams to study for. In the meantime, I look forward to enjoying the very last months of my senior year.

<table>
<thead>
<tr>
<th><strong>Exclusionary Persona</strong></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Name:</strong></td>
<td>Michelle Minor</td>
<td></td>
</tr>
<tr>
<td><strong>Gender:</strong></td>
<td>Female</td>
<td></td>
</tr>
<tr>
<td><strong>Age:</strong></td>
<td>21</td>
<td></td>
</tr>
<tr>
<td><strong>Year:</strong></td>
<td>Junior (i.e. year three)</td>
<td></td>
</tr>
</tbody>
</table>
**Major:** Meteorology (B.S.)  
**Minor:** Information Sciences and Technology  
**Goals:**  
To be gainfully employed after graduation.  
To pursue the career of my choice.  
To gain expertise relevant to meteorology in the field of information sciences and technology.  
To gain academic knowledge and wisdom.  
To earn the highest grades I can.  
To get involved.  
To make new friends.  
To have fun.  

**My Story:**  
Hello! My name is Michelle Minor. I am 21 years old, and this is my third year as a Meteorology student at Penn State University. I have a large family. I have an older brother who graduated from Temple University and an older sister who graduated from Drexel University. I also have two younger brothers and one younger sister. This year, after discovering the value of information sciences and technology to the field of meteorology, I decided to apply for a minor in Information Sciences and Technology. I am unfamiliar with the IST Building. However, I have a fairly good sense of direction, so I rarely get lost. I have been noticing that IST and SRA majors tend to hang out in the IST Building to do school work. I believe they even have access to the classrooms whenever class is not in session. However, I much prefer working with my fellow Meteorology majors in the Earth and Mineral Sciences library. Whenever my IST classes require specialized software, I usually just cave in and download or purchase the software for my personal laptop. Personally, I do not see any need for using the classrooms in the IST Building outside of class.

**Scenarios**

For ClassroomsUnlocked, the following three scenarios were defined. The highest priority scenario involves making decisions in the present by finding an available classroom starting now and ending at a specified time in the near future. The second highest priority scenario involves planning for the future by finding an available classroom at arbitrary start and end times. The scenario of least priority involves searching the availability of a single classroom. This scenario was included to prove that ClassroomsUnlocked in this scenario will not perform any more efficiently than the paper-based system. All three scenarios are described below in terms of the primary persona, Emily Expert.

**Primary Scenario**
It is one o’clock in the afternoon on a Wednesday. Emily Expert just got out of her class in 202 IST on the second floor. She has an hour of free time before her next class in 110 IST to work on an IST project. All she needs is a computer. She does not need any specialized software. The more time she spends looking for an open classroom, the less time she will have to work on her project. From memory, she knows that 110 IST is available any time after three o’clock in the afternoon, but that information does not help her right now. She starts checking each classroom one by one, but, then, suddenly, she remembers that she can just use ClassroomsUnlocked to find the perfect classroom. She finds her friend in the hallway. Her friend is working on a laptop, so she asks to borrow the laptop for a few seconds. Her friend obliges. She pulls up a web browser and launches ClassroomsUnlocked. The application displays a three-dimensional model of the IST Building highlighting in green all the classrooms that are available within the next half hour. She becomes excited when the cybertorium, 113 IST, is highlighted in green. She thinks the cybertorium is one of the nicest rooms at Penn State. She clicks on the time slider and extends the range from half an hour to a full hour. To her further joy, the cybertorium is still glowing green. Finally, she clicks on the cybertorium and pulls up the classroom’s schedule for Wednesday. She finds out that the classroom is reserved from 11:15 am to 12:05 pm and from 2:30 pm to 3:45 pm. She wishes she could take advantage of the full hour and a half of availability. However, class is really important, and she absolutely can’t be late for her next class. An hour in a comfy cybertorium chair will have to be enough. She returns the laptop to her friend and heads over to the cybertorium, anxious to get started working on her project as soon as possible.

Secondary Scenario

It is two o’clock in the afternoon on a Friday. Emily Expert has just finished exchanging contact information with her new teammates for a group project. The class is quickly coming to an end, and the group only has ten minutes left to decide where and when to hold meetings outside of class. None of the group members can afford to stay late after class. After five minutes of discussions, Emily figures out that all group members are available from seven o’clock to eight o’clock every Tuesday night. Group members prefer meeting in an IST classroom because the location is very convenient, the software needed for the project comes pre-installed, and whiteboards are available to sketch out ideas. Emily now has five minutes to figure out which classroom to use for weekly meetings. She could walk around the IST Building checking the sheets of paper, but that would take at least ten minutes of walking and note taking. She could also just pick a random classroom and hope that everything works out. Suddenly, she remembers that she can just use ClassroomsUnlocked to find the perfect classroom. She launches a web browser and launches ClassroomsUnlocked. The application displays a three-dimensional model of the IST Building highlighting in green all the classrooms that are available within the next half hour. She disregards this because she is not interested in the present. Instead, she is interested in planning for the future. She clicks on the day of week slider and slides it to Tuesday. Then, she clicks on the time slider and slides it to the range between seven o’clock and eight o’clock in the evening. To her delight, 110 IST, 113 IST, 205 IST, 206 IST, and 210 IST are all highlighted in green. She tells her teammates. They are amazed that she was able to provide this list of classrooms so quickly. After a quick vote, the team agrees that 110 IST is the best location for their weekly meetings because it has an especially nice whiteboard and is very quiet due to its isolation from all the other classrooms. Class is dismissed. Everyone looks forward to the very first meeting.
**Tertiary Scenario**

It is four o’clock in the afternoon on a Thursday. Emily Expert has just finished talking to her friend in the hallway right outside of 202 IST. Emily would like to use 202 IST to check her email inbox. However, this task is not absolutely critical. If 202 IST is not available, she is just going to walk back to her apartment and read her emails there on her laptop. She is not at all interested in any other classrooms. She is convinced that ClassroomsUnlocked is always better than the paper-based system, so she ignores the sheet of paper next to 202 IST and asks her friend to launch ClassroomsUnlocked on her laptop instead. Her friend obliges. The application displays a three-dimensional model of the IST Building highlighting in green all the classrooms that are available within the next half hour. Emily ignores everything except 202 IST. 202 IST is highlighted in green. She clicks on 202 IST and pulls up the classroom’s schedule for Thursday to confirm. Yes, 202 IST is available for email reading. She thanks her friend, heads into 202 IST, and begins checking her emails. Later, she wonders whether it would have been more efficient and less of a hassle for her friend to simply glance at the sheet of paper hanging next to 202 IST.

**User Stories**

These scenarios can be further broken down and distilled into simple user stories. The sentences below represent important user stories captured from the primary, secondary, and tertiary scenarios.

- As an IST/SRA undergraduate student, I want a three-dimensional model of the IST Building highlighting in green all the classrooms that are available within the next half hour so that I instantly recognize which classrooms are currently available.

- As an IST/SRA undergraduate student, I want a three-dimensional model of the IST Building highlighting in red all the classrooms that are occupied within the next half hour so that I instantly recognize which classrooms are currently occupied.

- As an IST/SRA undergraduate student, I want to be able to toggle between the first and second floors of the IST Building so that 110 IST Building is not always hidden from view.
• As an IST/SRA undergraduate student, I want to be able to click on the time slider and extend the time range so that I can find classrooms that are available for a longer period of time.

• As an IST/SRA undergraduate student, I want to be able to click on the day of the week slider and slide it to different days of the week so that I can plan for the future.

• As an IST/SRA undergraduate student, I want to be able to click on the time slider and slide it to different time spans so that I can plan for the future.

• As an IST/SRA undergraduate student, I want to be able to click on a classroom and pull up its schedule for the current day of the week so that I can analyze and confirm the raw information for myself.

• As an IST/STA undergraduate student, I want to be able to launch ClassroomsUnlocked from a variety of devices, including smartphones and tablets, so that I can search classroom availability on the go.

User Interface Design

The second step of designing ClassroomsUnlocked was designing the user interface. All design decisions drew inspiration from the personas, scenarios, and user stories as well as Norman’s Seven Stage of Action and Nielson’s ten usability heuristics. The design process was iterative and involved the use of wireframes.

An early wireframe reserved the top of the user interface exclusively for spatial information and the bottom of the user interface exclusively for temporal information. The top view (i.e. space) and the bottom view (i.e. time) were considered inexorably linked such that changes made in one view would be instantly reflected in the other. Attempting a very minimalist approach with strong perceived affordances, this wireframe used a cross-section three-
The three-dimensional model was designed to afford the kind of three-dimensional spatial interaction that best matches the users’ typical mental model of such interaction. Given recent advances in three-dimensional technology and the importance of these advances to the field of information science, many students in the College of IST have probably been exposed to this type of interface (ex. Google Earth) and intuitively understand its fundamental operations. A three-dimensional interface was preferred over a two-dimensional one due to the needs of the primary persona. The primary persona, a veteran user of the IST Building, has built up a rich three-dimensional mental model of the spatial structure of the building after several semesters of walking through its halls and noticing the width, height, and depth between objects. As with any memory, this mental model is not an absolute replica but rather an ever-shifting cognitive network of objects in abstract three-dimensional space. It is cognitively challenging and rather limiting to map this powerful cognitive network to a simple two-dimensional interface. It is better to provide a three-dimensional interface where users instantly recognize the three-dimensional shape of building and naturally superimpose their mental model and vast experiences upon it.

Likewise, the weekly calendar was designed to afford the kind of temporal selection that best matches the users’ typical mental model for such selection. Inspired by Google Calendar, the idea was that users would naturally click and drag over the calendar just like a real calendar in order to select a day of the week, a start time, and an end time. In addition to providing a much more organic and minimalist experience than simple text entry, this interface was also designed to provide implicit information about the relative arrangement of time through the use of spatial cues. The days of the week were arranged horizontally from Sunday to Saturday, and the times
were arranged vertically from morning to night. The figure below is a screenshot of the early wireframe.

![Early Wireframe](image)

**Figure 17. Early Wireframe**

After several iterations, the final user interface emerged. The top view remained very similar with the addition of clickable colored panels representing each room’s availability and a simple floor selector. The color green, which is strongly associated with the word “go” in traffic light systems, was chosen as the color of availability. The color red, which is strongly associated with the word “stop” in traffic light systems, was chosen as the color of reservation. The floor selector allowed for the dynamic removal of the second floor to reveal the only classroom on the
first floor, 110 IST Building. This selector was designed to respond instantly without the need for clicking a confirmation button or another similar action.

The bottom view changed dramatically in the pursuit of greater minimalism and better perceived affordances. Instead of a calendar, a simple combination of a day of the week slider and a time range slider was designed to represent the temporal component. The design of this interface was intended to make it clear to the user that they must select one and only one day of the week and one and only one time span within that day of the week. Another goal was to clearly indicate that all schedules are cyclical and periodic with respect to the week and that the day of the week is much more important than the precise date. This interface begins with the current day of the week and a time range spanning half an hour past the current time. Half an hour was strategically chosen because the average IST and SRA endeavor requires at least half an hour to complete. The sliders have prominent handles that are believed to afford and encourage users to slide them and experiment with different temporal settings. The figure below is a screenshot of the final user interface.

Figure 18. Final User Interface
Data Model

The third step of designing ClassroomsUnlocked was defining a data model to hold the classroom availability information and support the sophisticated spatiotemporal relationship queries needed to answer the question: “During the limited time I have, where may I go in the IST Building to complete my assignments and study for my exams?” The data model assumes that the Schedule of Courses is the ultimate authority on classroom availability information. Therefore, instead of storing a separate record for each separate occurrence of a classroom reservation, the data model stores a single unified record for each weekly recurrence pattern (i.e. a combination of a day of the week, a start time, and an end time). A consequence of this assumption is that the data model does not account for extracurricular classroom reservations. However, these types of events are far less frequent and much easier for students to memorize. Also, students may be able to multitask and do school work at such events.

The data model was defined as a traditional relational model for the purposes of simplicity and standardization. Admittedly, recent so-called “no-SQL” data models databases offer several unique and modern benefits over the traditional relational model. However, they also tend to be more complex and do not follow unified standards. The Entity-Relationship (ER) diagram is a popular high-level method for modeling relational data. This paper will employ the ER diagram notation used by the book Fundamentals of Data Systems (Elmasri & Navathe, 2009). Below is an ER diagram of the data model for ClassroomsUnlocked.
SQLite was chosen as the relational database management system. The reasons, again, were simplicity and standardization. The goal of SQLite is to provide the simplest and most lightweight system on the market that follows the latest SQL standards. Fortunately, SQLite is also open-source, free-to-download, and automatically included in several well-known programming libraries (“When to use,” 2013). The ER diagram above was translated into the following SQLite code. All attributes were assigned the TEXT data type because SQLite will accept textual data without error even when the INTEGER data type is used. It was believed that this odd usage of the TEXT data type would warn potential users of the data model of this unexpected behavior.

```
-- a Penn State course
-- identified by its course abbreviation
CREATE TABLE Course(
    courseAbbreviation TEXT,
```

Figure 19. ER Diagram
courseName TEXT,
courseCredits TEXT,
courseDescription TEXT,
    PRIMARY KEY(courseAbbreviation)
); 

-- a section of a Penn State course
-- identified by its schedule number
-- associated with a Penn State course by course abbreviation
CREATE TABLE Section(
    scheduleNumber TEXT,
courseAbbreviation TEXT,
sectionNumber TEXT,
seatsOpen TEXT,
seatsMax TEXT,
instructorName TEXT,
    PRIMARY KEY(scheduleNumber),
FOREIGN KEY(courseAbbreviation) REFERENCES Course(courseAbbreviation)
); 

-- a temporal relationship between a section of a Penn State course and a room at Penn State
-- identified by its schedule number, room name, start time, and end time
-- all times are stored in 'yyyy-mm-dd hh:mm' format where 'hh:mm' is in military time and 'yyyy-mm-dd' actually represents a day of the week...
-- '2013-01-01' => 'Sunday', '2013-01-02' => 'Monday', '2013-01-03' => 'Tuesday', '2013-01-04' => 'Wednesday', '2013-01-05' => 'Thursday', '2013-01-06' => 'Friday', '2013-01-07' => 'Saturday'
-- associated with a section of a Penn State course by name
CREATE TABLE SectionRoom(
    scheduleNumber TEXT,
roomName TEXT,
startTime TEXT,
endTime TEXT,
    PRIMARY KEY(scheduleNumber, roomName, startTime, endTime),
FOREIGN KEY(scheduleNumber) REFERENCES Section(scheduleNumber)
); 

After executing the preceding code within a proper SQLite environment, the three relational tables that define the data model will be created, and the database will be ready for the insertion of records from the Schedule of Courses. Instead of storing the day of the week as a separate field within the SectionRoom table, the day of the week is implicitly included in both the start time and the end time by means of a coded date system where January 1, 2013 represents
Sunday and January 7, 2013 represent Saturday. This simple approach allows for powerful temporal queries based only upon the SQL equality and inequality operators.

**Implementation**

The final step of designing ClassroomsUnlocked was implementing the software. The software is divided into two separate command-line utilities called DownloadSchedule and ScheduleToSqlite and one web-based client-server system called ClassroomsUnlocked.

*DownloadSchedule (Source Code in Appendix A)*

The purpose of the DownloadSchedule utility is to automate the process of downloading large volumes of HTML documents from the Schedule of Courses website. This utility is specifically designed to find all the course subject abbreviations for the current academic semester and then download exactly 10 HTML documents from each course subject. Based on several tests of the Schedule of Courses website, it is assumed that the total number of HTML documents for each course subject will never exceed 10. Although the Schedule of Courses is a public website, it is challenging to use traditional web crawling methods to obtain its HTML documents. The Schedule of Courses website generates all of its pages dynamically and requires the use of cookies. The website may employ these techniques to avoid becoming indexed by Google’s search engine. Most HTTP APIs for most programming languages do not have features for saving cookies. One notable exception is an open-source, free-to-download Apache Foundation project called HttpClient. HttpClient is a Java library that can save cookies and behave like a normal web browser (“HttpClient,” 2013). Thus, the DownloadSchedule utility became “DownloadSchedule.java,” a small single-class Java program completely wrapped around the functionality of HttpClient. Below is an example of an HTML document downloaded from the Schedule of Courses website. Note that the pictures and other non-textual elements were not downloaded.
This output can be replicated for the current academic semester. After downloading HttpClient and properly adding its JAR file to the Java classpath, compile and run “DownloadSchedule.java,” which is found in Appendix A.

*ScheduleToSqlite (Source Code in Appendix B)*

Afterwards, these HTML documents need to be parsed into a set of SQLite INSERT statements used to fill the SQLite database with meaningful information. The utility that performs this job is named ScheduleToSqlite. To parse records out of an HTML document, specific patterns in the document must be recognized using a string-matching process known as regular expressions. Perl was one of the first and most successful programming languages to incorporate regular expressions into its syntax. Many modern languages with regular expression capabilities model their regular expression syntax and libraries after Perl (“Regular expressions,” 2013). Therefore, Perl was chosen as the lingua franca of ScheduleToSqlite. I was previously unfamiliar with Perl, so this decision resulted in a tremendous learning experience. The design of
“ScheduleToSqlite.pl” was straightforward. The Perl script accepts a path to an HTML document, creates a matching output document with “.sql” appended to its name, and fills the output document with SQLite INSERT statements for every course, section, and section-room combination it finds. The output of ScheduleToSqlite is in the following format.

```sql
INSERT OR IGNORE INTO Course(courseAbbreviation, courseName, courseCredits, courseDescription)
VALUES('IST 110', 'Information, People and Technology', '3.0', 'The use, analysis and design of information systems and technologies to organize, coordinate, and inform human enterprises.');

INSERT OR IGNORE INTO Section(scheduleNumber, courseAbbreviation, sectionNumber, seatsOpen, seatsMax, instructorName)
VALUES('929923', 'IST 110', '001', '4', '150', 'Petrick, Irene Johnston');

INSERT OR IGNORE INTO SectionRoom(scheduleNumber, roomName, startTime, endTime)
VALUES('929923', '113 IST Building', '2013-01-02 16:15', '2013-01-02 17:30');

INSERT OR IGNORE INTO SectionRoom(scheduleNumber, roomName, startTime, endTime)
VALUES('929923', '113 IST Building', '2013-01-04 16:15', '2013-01-04 17:30');

INSERT OR IGNORE INTO Course(courseAbbreviation, courseName, courseCredits, courseDescription)
VALUES('IST 110S', 'Information, People and Technology', '3.0', 'The use, analysis and design of information systems and technologies to organize, coordinate, and inform human enterprises.');

This output can be replicated for the current academic semester. After configuring your Perl environment, run “DownloadSchedule.java,” which is found in Appendix B, against an HTML document downloaded from the Schedule of Courses website and open the output file. In
a Windows environment, the following batch script may help automate the processing of multiple HTML documents. This script will find all the HTML documents in the “Downloads” folder, run “ScheduleToSqlite.pl” against them, and then merge all the output files in the “Downloads” folder into a single file named “Dump.sql.”

```
for %%f in (Downloads\*.html) do {
    echo "Downloads\%%~nf.html"
    perl ScheduleToSqlite.pl "Downloads\%%~nf.html"
}

copy Downloads\*.sql Dump.sql /Y
```

**ClassroomsUnlocked (Source Code in Appendix C)**

While the DownloadSchedule and ScheduleToSqlite utilities are intended to be run once each new semester to prime the system, ClassroomsUnlocked runs constantly, serving the immediate needs of the users. It is a web-based client-server application using PHP and SQLite on the server side and HTML5 and JavaScript on the client side. By following universal standards from the World Wide Web Consortium, ClassroomsUnlocked can be run from a wide variety of computing devices, including smartphones and tablets. As technologists, IST and SRA students are very likely to be using smartphones and tablets. HTML5 was chosen to take advantage of the latest advances in client-side web browsing technologies. Although this decision limits the backwards compatibility of ClassroomsUnlocked, it also keeps the design clean, modern, and up-to-date with the latest standards from the World Wide Web Consortium. As long as students are running a modern, HTML5-compatible browser, they should be able to use ClassroomsUnlocked successfully. For the purposes of this prototype, success within the latest version of the Google Chrome web browser is assumed to equate to success within any other HTML5-compatible browser. Empirically, this assumption may not always hold, but it effectively prevents the design from becoming needlessly plagued by cross-browser compatibility concerns.
PHP and SQLite were chosen because Penn State University hosts a free public-facing PHP server for all students, which supports SQLite, the preferred relational database management system. JavaScript was chosen for two reasons: Google Earth and jQuery. The three-dimensional model of the IST Building provided to me by Guorary Cai is pre-optimized for Google Earth, and Google Earth offers many built-in features that satisfied the spatial needs of Classrooms Unlocked. The only API for the Google Earth Plug-in is written in JavaScript (“Google Earth API,” 2013). Also, a lightweight cross-browser-compatible JavaScript library named jQuery provided many useful HTML Document Object Model (DOM) manipulation features that make it easy to quickly prototype the user interface. For example, all the sliders were created using a powerful user interface library called jQuery UI (“jQuery,” 2013). Although JavaScript is not necessarily an object-oriented language, elegant solutions exist that allow for the modeling of classes and objects (Stefanov, 2006).

The server side is composed of five different PHP files and a SQLite file. The first PHP file is for phpLiteAdmin, an open-source, free-to-download project on Google Code which provides a SQLite database management interface similar to MySQL’s phpMyAdmin (“What is phpLiteAdmin,” 2013). This file is used to execute all the CREATE and INSERT statements needed to create the single “psu.db” SQLite file on the server containing all of the classroom availability information for Penn State University. The second PHP file “simplejson.php” was adapted from yet another Google Code project called SimpleJSON and is used to add JSON parsing capabilities to PHP (“SimpleJSON,” 2013). The next two PHP files mediate all client-side requests to the SQLite file, ensuring secure transactions without the threat of SQL injection. The first file, called “occupied.php,” handles an important Asynchronous JavaScript and XML (AJAX) request that fires the instant the user moves the time sliders. Given an HTTP POST request containing the current time span, this file queries the SQLite file and returns all the occupied classrooms in JavaScript Object Notation (JSON) for ease of use. The second file is
called “room.php.” Given an HTTP GET request containing the name of a classroom, this file returns a simple HTML document containing the classroom schedule for the current day of the week. The goal of this file is to provide an interface similar to the familiar paper-based system in case the user is interested in only one classroom instead of multiple classrooms. The last PHP file is “index.php.” This file is the main entry point of the application. Users start the application by asking their web browser to retrieve “index.php.” Despite being a PHP file, this file has no actual PHP code in it. The PHP suffix is used to make sure that it can safely communicate with “occupied.php” and “room.php.” The figure below outlines the name and purpose of each file.

<table>
<thead>
<tr>
<th>File Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>phpliteadmin.php</td>
<td>SQLite database management interface similar to MySQL’s phpMyAdmin</td>
</tr>
<tr>
<td>(not in Appendix C)</td>
<td></td>
</tr>
<tr>
<td>psu.db</td>
<td>SQLite file containing all of the classroom availability information</td>
</tr>
<tr>
<td>(not in Appendix C)</td>
<td></td>
</tr>
<tr>
<td>simplejson.php</td>
<td>adds JSON parsing capabilities to PHP</td>
</tr>
<tr>
<td>(not in Appendix C)</td>
<td></td>
</tr>
<tr>
<td>occupied.php</td>
<td>given an HTTP POST containing a time span, returns a JSON list of all the occupied classrooms</td>
</tr>
<tr>
<td>(in Appendix C)</td>
<td></td>
</tr>
<tr>
<td>room.php</td>
<td>given an HTTP GET containing a classroom name, returns a simple HTML document containing the classroom schedule for the selected day of the week</td>
</tr>
<tr>
<td>(in Appendix C)</td>
<td></td>
</tr>
</tbody>
</table>
The client side is much more complex because any logic not handled by the SQLite database must be handled by client-side JavaScript code in order to ensure a smooth AJAX experience without any visually jarring page refreshes. The figure below outlines the name and purpose of each client-side file.

<table>
<thead>
<tr>
<th>File Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>IstBuildingFloor1.kmz</strong></td>
<td>This compressed (i.e. zipped) Keyhole Markup Language (KML) file contains a three-dimensional model of the IST Building without the roof, the third floor, and the second floor. Whenever the first floor is selected, this file is loaded and displayed by the Google Earth Plug-in.</td>
</tr>
<tr>
<td><strong>IstBuildingFloor2.kmz</strong></td>
<td>This compressed (i.e. zipped) Keyhole Markup Language (KML) file contains a three-dimensional model of the IST Building without the roof and the third floor. Whenever the second floor is selected, this file is loaded and displayed by the Google Earth Plug-in.</td>
</tr>
<tr>
<td><strong>SaptiotemporalUi.css</strong></td>
<td>This Cascading Style Sheet (CSS) defines the</td>
</tr>
<tr>
<td><strong>Main.js</strong>  (in Appendix C)</td>
<td>layout and style for “index.php.” The layout, defined in terms of percentages instead of pixels, is considered “fluid” because it gracefully adjusts to changes in window size (Knight, 2009).</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>--------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>IstBuilding.js</strong> (in Appendix C)</td>
<td>This JavaScript file declares a class called IstBuilding. An IstBuilding object contains a great deal of information about the IST Building, including the precise geographic coordinates of each classroom and a collection of Room objects. An IstBuilding object also manages how the IST Building is displayed within the Google Earth Plug-in.</td>
</tr>
<tr>
<td><strong>Room.js</strong> (in Appendix C)</td>
<td>This JavaScript file declares a class called Room. A Room object manages how a single classroom is displayed within the Google Earth Plug-in.</td>
</tr>
</tbody>
</table>
| **Time.js**  
| **(in Appendix C)** | This JavaScript file declares several utility functions used to translate between several different time formats such as a date, a day of the week, an am/pm time, or a military time. |
| **TypeCheck.js**  
| **(in Appendix C)** | This JavaScript file declares several utility functions used to check the type of JavaScript variables. JavaScript has a dynamic typing system without type checking. Many of the other JavaScript files call this file’s functions in order to guard against bad input parameters by throwing exceptions. |
| **DayOfWeekSlider.js**  
| **(in Appendix C)** | This JavaScript file declares a class called DayOfWeekSlider. A DayOfWeekSlider object wraps around a normal jQuery UI slider, providing additional functionality unique to days of the week. This object also displays helpful text alongside the slider. |
| **AmPmTimeSlider**  
<p>| <strong>(in Appendix C)</strong> | This JavaScript file declares a class called AmPmTimeSlider. An AmPmTimeSlider object wraps around a normal jQuery UI range slider, providing additional functionality unique to am/pm time. This object ranges a |</p>
<table>
<thead>
<tr>
<th>SlidingParagraph.js (in Appendix C)</th>
<th>full 24 hours from 12:00 am to 12:00 am in 5-minute increments. This object also displays helpful text alongside the slider.</th>
</tr>
</thead>
<tbody>
<tr>
<td>This JavaScript file declares the SlidingParagraph class. The SlidingParagraph object helps the DayOfWeekSlider and AmPmTimeSlider objects place text strategically in proportion to slider values. This object makes it easy to create scales for sliders and to have a piece of text float above the slider handle no matter where the handle moves.</td>
<td></td>
</tr>
</tbody>
</table>

**Figure 22. Client-side Files**

**Screenshots**

The figures below demonstrate the first and second floors of the IST Building as displayed within the Google Earth Plug-in. Green classrooms are available, and red classrooms are occupied. By clicking on a classroom, the classroom’s name and status appear in a speech bubble. By clicking on the status, the room schedule for that classroom on the selected day of the week is opened in a new tab.
Figure 23. Screenshot of First Floor of IST Building

Figure 24. Screenshot of Second Floor of IST Building
Figure 25. Screenshot of Clicking on Status of 110 IST Building on a Tuesday

The figure below demonstrates the day of the week slider and the am/pm time slider. The day of the week slider spans from Sunday to Saturday, and the am/pm time slider spans 24 hours from 12:00 am to 12:00 am in 5-minute increments. The day of the week slider has only a single handle and a single value. The am/pm time slider has two handles and a range of values. The text above the handles follows the position of the handles and change in value accordingly.

Figure 26. Screenshot of Day of Week Slider and Time Slider

This final figure demonstrates the entire user interface as a whole.
Figure 27. Screenshot of Entire User Interface
Chapter 4
Analysis

Keystroke-level Model

The keystroke-level model (KLM) estimates the average efficiency of tasks by breaking them down into discrete, measurable, keystroke-level steps called operations. First, each step is codified into an operation. Then, simple heuristics are used to estimate the average time needed to complete each operation. Once all the operations have been estimated, all of the average times are summed together to yield the average time needed to complete the entire task. This model assumes that tasks can be decomposed into operations, that the task completion time depends solely upon keystrokes, device operations, and mental operations of the same order of magnitude, and that the estimate should be based upon an expert user choosing one and only path through the system and making no errors. The figure below outlines the typical KLM operation taxonomy (Card, Moran, & Newell, 1980).

<table>
<thead>
<tr>
<th>Code</th>
<th>Operation</th>
<th>Average Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>K</td>
<td>key press and release</td>
<td></td>
</tr>
<tr>
<td></td>
<td>best typist (135 wpm)</td>
<td>0.08 seconds</td>
</tr>
<tr>
<td></td>
<td>good typist (90 wpm)</td>
<td>0.12 seconds</td>
</tr>
<tr>
<td></td>
<td>poor typist (40 wpm)</td>
<td>0.28 seconds</td>
</tr>
<tr>
<td></td>
<td>average skilled typist (55 wpm)</td>
<td>0.20 seconds</td>
</tr>
<tr>
<td></td>
<td>average non-secretary typist (40 wpm)</td>
<td>0.28 seconds</td>
</tr>
<tr>
<td></td>
<td>typing random letters</td>
<td>0.50 seconds</td>
</tr>
<tr>
<td></td>
<td>typing complex codes</td>
<td>0.75 seconds</td>
</tr>
</tbody>
</table>
For users over the age of 30, graduated time adjustment multipliers must be used to account for the gradual decrease in reaction time that naturally occurs as people age. An IST or SRA student over the age of 30 would be an outlier rather than the norm, so any time adjustment multipliers can safely be ignored (Card, Moran, & Newell, 1980).

Efficiency of the Primary Scenario

The primary scenario can be decomposed into the following operations, starting immediately after Emily Expert has launched ClassroomsUnlocked.

1. Visually perceive and comprehend the user interface. (M)
2. Point the mouse at 113 IST. (P)
3. Understand that the time span must be extended to a full hour. (M)
4. Point the mouse at the second handle of the time slider. (P)
5. Press the left mouse button. (B)
6. Point the mouse at two o’clock in the afternoon. (P)
7. Release the left mouse button. (B)
8. Wait for the asynchronous database call to complete and the user interface to
refresh itself. (W)

9. Visually perceive and comprehend the changes to the user interface. (M)

10. Point the mouse at 113 IST. (P)

11. Press the left mouse button. (B)

12. Release the left mouse button. (B)

13. Point the mouse at the word “AVAILABLE.” (P)

14. Press the left mouse button. (B)

15. Release the left mouse button. (B)

16. Visually perceive and comprehend the new user interface. (M)

Thus, the primary scenario can be expressed as “MPMPBPBWMPBBPBBM.” Based on
10 measurements of the prototype system, the average performance of the asynchronous database
call is 0.2 seconds on average. The “W” should, therefore, be replaced with “W(0.2).” The sum
of “MPMPBPBW(0.2)MPBBPBBM.” is 11.1 seconds. This implies that the average time to
complete is the primary scenario is 11.1 seconds.

How efficient is the paper-based system in the same scenario? This can be evaluated by
decomposing the paper-based system into operations. This system does not have any key-stroke
level operations besides mental preparation. However, it does have a wealth of physical
operations. Walking to the nearest classroom will be codified as “W.” Based on 25
measurements taken throughout the IST Building by me walking at my normal pace, the average
time for “W” is approximately 15.3 seconds. Reading the classroom schedule will be codified as
“R.” Based on 25 measurements taken throughout the IST Building by me reading at my normal
pace, the average time for “R” is approximately 2.7 seconds. Assuming a truly random
distribution of classroom availability, on average, the user will find the perfect classroom, if it
exists, after randomly checking exactly half of the classrooms (i.e. 4 classrooms). The following operations are based on the assumption that the student must visit exactly 4 classrooms.

1. Mentally prepare for the journey. (M)
2. Walk to the first classroom. (W)
3. Read the first classroom schedule. (R)
4. Walk to the second classroom. (W)
5. Read the second classroom schedule. (R)
6. Walk to the third classroom. (W)
7. Read the third classroom schedule. (R)
8. Walk to the fourth classroom. (W)
9. Read the fourth classroom schedule. (R)

The sum of “MWRWRWRWR” is 73.2 seconds or 1 minute and 13.2 seconds. This implies that the average time to complete the same scenario using the paper-based system is 1 minute and 13.2 seconds.

Efficiency of the Secondary Scenario

The secondary scenario is a simple permutation of the primary scenario. In the secondary scenario operations 3 through 9 must be repeated until the user is satisfied with the result. Thus, to find the average completion time of the secondary scenario, first, find the average completion time of the primary scenario (i.e. 11.1 seconds). Then, find the average completion time of operations 3 through 9, which is 5 seconds. Multiply 5 seconds times the number of repetitions minus one repetition. Finally, add the result of the multiplication to the average completion time of the primary scenario to find the answer. In other words, “secondary scenario = primary scenario – operations 3 through 9 * (repetitions – 1).”
For the paper-based system, the secondary scenario requires the repetition of all 9 operations. Instead of multiplying 5 seconds times the number of repetitions minus one repetition, 73.2 seconds must be multiplied times the number of repetitions minus one repetition.

**Efficiency of the Tertiary Scenario**

The tertiary scenario is the simplest of all the scenarios and it is expected that ClassroomsUnlocked will perform no more efficiently than the paper-based system. The tertiary scenario can be decomposed into the following operations.

1. Visually perceive and comprehend the user interface. (M)
2. Point the mouse at 202 IST. (P)
3. Press the left mouse button. (B)
4. Release the left mouse button. (B)
5. Point the mouse at the word “AVAILABLE.” (P)
6. Press the left mouse button. (B)
7. Release the left mouse button. (B)
8. Visually perceive and comprehend the new user interface. (M)

The sum of “MPBBPBBM” is 5 seconds. This implies that the average time to complete this scenario is 5 seconds. Meanwhile, the paper-based system’s approach to the same scenario can be broken down into an even smaller set of operations.

1. Recall the classroom schedule from memory or mentally prepare to read the classroom schedule in your immediate proximity. (M)
2. Read the classroom schedule. (R)

The sum of “MR” is 3.9 seconds. This implies that the average time it take the paper-based system to complete the same scenario is 3.9 seconds.
Chapter 5
Discussion

Results

The assumptions used for the paper-based system were relatively optimistic. 110 IST Building and 113 IST Building are very far away from the other classrooms, and walking times to these classrooms would radically depart from the 15.3-second average. Also, the average reading pace is most likely much faster than average because I am experienced in analyzing classroom availability information. Finally, classroom availability is not randomly distributed at all because the University decides classroom availability intentionally. It is by no means guaranteed that a student will find the perfect classroom after visiting exactly four classrooms.

Nevertheless, even with these optimistic assumptions on the side of the paper-based system, ClassroomsUnlocked appeared to be significantly more efficient than the paper-based system. For the primary scenario, ClassroomsUnlocked had an average time of 11.1 seconds. Under the same circumstances, the paper-based system had an average time of 1 minute and 13.2 seconds. That is a difference in efficiency of 1 minute and 2.1 seconds. In other words, ClassroomsUnlocked demonstrated an 85% increase in efficiency. For the secondary scenario, the estimate was dependent on the number of repetitions. For ClassroomsUnlocked, each repetition added only 5 seconds to the average time. For the paper-based system, each repetition added 73.2 seconds to the average time. In other words, as the number of repetitions increase, the average time for the paper-based system grows at a rate just over 14.5 times higher than the average time for ClassroomsUnlocked.

The tertiary scenario was expected to demonstrate that the paper-based system has already achieved maximum efficiency. It appears that this expectation was indeed met. For the
tertiary scenario, ClassroomsUnlocked had an average time of 5 seconds. Meanwhile, the paper-based system had an average time of 3.9 seconds. Thus, it seems that ClassroomsUnlocked was unable to break through the efficiency barrier set by the paper-based system. This seems to be a clear reversal of the trend from the previous scenarios.

**Conclusion**

The results of my analysis support my hypothesis that the geo-centric integration and presentation of classroom availability information in the IST Building will lead to more effective information seeking and sense making in a range of common student activities. ClassroomsUnlocked actively sought the best requirements modeling practices and the best design principles and was rewarded with a significant increase in efficiency. The existing paper-based system remains the most efficient at answering one-room queries under perfect conditions. However, the personas, scenarios, and user stories generated for this prototype all support the notion that such perfect queries are rare and can be easily answered without a formal system. Students in the College of IST demand a system that can answer their everyday complex queries about the availability of multiple classrooms, and ClassroomsUnlocked is a first leap in the right direction.

These results may have limited generalizability because the design was custom-tailored to the IST Building. However, ClassroomsUnlocked contains information about the availability of classrooms throughout Penn State’s campus. Thus, it would be both reasonable and practical to consider generalizing these results to other academic buildings at Penn State. To a further extent, these results could also be generalized to similar universities whose students face similar issues when searching for classroom availability information. Nevertheless, one potential limitation of these generalizations would be the unexplored motivations that drive students to be interested in classrooms without computers. Classroom availability can be considered a model for room availability in general. In the right circumstances, effective design concepts from this
software prototype may generalize well to other room availability systems in the fields of business, hospitality, and medicine.

**Recommendations for Future Research**

This paper was limited in scope to the design and preliminary analysis of a software prototype. As a prototype and not a formal system, it was challenging to exhaustively evaluate the efficiency of ClassroomsUnlocked. Why precisely is ClassroomsUnlocked efficient? Can better design decisions increase the efficiency still further? Are there factors outside of requirements modeling and design that increase the efficiency, such as aesthetics and processing power? Is efficiency the only benefit? This paper addressed the research question mostly qualitatively with an estimated and heuristically measured quantitative component. I recommend that future studies address these questions by means of quantitative user studies. Specifically, I recommend that the College of IST adopts a version of ClassroomsUnlocked as a formal system on a trial basis. This would provide researchers with the perfect opportunity to conduct meaningful quantitative user studies that measure exactly how this system behaves “in the wild.”

Accessibility is another limitation of this paper. For the sake of simplicity, this paper did not consider the needs of students who might require special accommodations in order to be able to use ClassroomsUnlocked. Given the visual nature of the design, visually impaired students may struggle to operate the user interface. Future studies should find good design practices for supporting visually impaired students. ClassroomsUnlocked should take advantage of the new accessibility features found in HTML5. Also, the choice of the colors red and green should be more carefully evaluated in terms of color blindness. The current symbols for available and occupied classrooms would be invisible to red-green colorblind students.
Appendix A

Source Code for DownloadSchedule.java

/ *-------------------------\
 "DownloadSchedule.java"
 \*-------------------------*/
// download all the HTML files from schedule.psu.edu?
// challenge accepted
// requires Apache's HttpClient library
import java.io.*;
import java.util.*;

// we'll also need Apache's HttpClient library
import org.apache.http.*;
import org.apache.http.client.*;
import org.apache.http.client.entity.*;
import org.apache.http.client.methods.*;
import org.apache.http.client.params.*;
import org.apache.http.cookie.*;
import org.apache.http.impl.client.*;
import org.apache.http.message.*;
import org.apache.http.util.*;

// class it up
public class DownloadSchedule {
    // insert current semester here
    public static final String SEMESTER = "Spring 2013";
    // insert number of pages per course abbreviation here
    public static final int PAGES_PER_ABBREV = 10;

    // we'll use this method to download a certain HTML page for a course abbreviation
    // if the course abbreviation were "IST" and the page number were 0, the name of the
downloaded page would be "downloads\IST.0.html"
    // if anything goes wrong, don't worry about it
    // we'll just let the caller handle that
    public static void loadPage(HttpClient httpClient, String abbrev, int page) throws
        Exception {
        // go postal on the HTTP protocol
        HttpPost httpPost = new
            HttpPost("http://schedule.psu.edu/act_search.cfm?pageIndex=" + page);

        // prepare the endless litany of parameters
        ArrayList<NameValuePair> parameters = new ArrayList<NameValuePair>();
        parameters.add(new BasicNameValuePair("Semester", SEMESTER));
        parameters.add(new BasicNameValuePair("CrseLoc", "UP::University Park Campus"));
        parameters.add(new BasicNameValuePair("CECrseLoc", "All"));
        parameters.add(new BasicNameValuePair("course_abbrev", abbrev));
        parameters.add(new BasicNameValuePair("course_num", ""));
        parameters.add(new BasicNameValuePair("college", ""));
        parameters.add(new BasicNameValuePair("prof_last", ""));
        parameters.add(new BasicNameValuePair("course_keywords", ""));
        parameters.add(new BasicNameValuePair("credits1", ""));
        parameters.add(new BasicNameValuePair("credits2", ""));
        parameters.add(new BasicNameValuePair("matchdays", "E"));
        parameters.add(new BasicNameValuePair("start_time", ""));
        parameters.add(new BasicNameValuePair("start_am_pm", "AM"));
        parameters.add(new BasicNameValuePair("end_time", ""));
        parameters.add(new BasicNameValuePair("end_am_pm", "PM"));

        // now let's rake in the parameters
        httpClient.execute(httpPost, parameters);
    }
parameters.add(new BasicNameValuePair("start_date", ""));
parameters.add(new BasicNameValuePair("end_date", ""));
httpPost.setEntity(new UrlEncodedFormEntity(parameters));

// download the HTML page
HttpResponse response = httpClient.execute(httpPost);
InputStream inputStream = response.getEntity().getContent();
(new File("downloads")).mkdirs();
BufferedWriter writer = new BufferedWriter(new FileWriter("." + File.separator + "downloads" + File.separator + abbrev + "." + page + ".html"));
BufferedReader reader = new BufferedReader(new InputStreamReader(inputStream));
String line = reader.readLine();
while (line != null) {
    writer.write(line);
    writer.newLine();
    line = reader.readLine();
}
reader.close();
writer.close();

// enter here if you dare

public static void main(String[] args) {
   // this is a crazy huge task
   // if anything goes wrong, let's just quit while we're ahead and report exactly
   what went wrong
   try {
      // HttpClient, please place any cookies you recieve in your cookie jar
      // also, please try your best to look like a normal browser
      DefaultHttpClient httpclient = new DefaultHttpClient();
      HttpClientParams.setCookiePolicy(httpclient.getParams(),
      CookiePolicy.BROWSER_COMPATIBILITY);

      // prime the pump by performing a simple HTTP get
      HttpGet httpGet = new HttpGet("http://schedule.psu.edu";
      HttpResponse response = httpclient.execute(httpGet);
      InputStream inputStream = response.getEntity().getCont(610);
BufferedReader reader = new BufferedReader(new InputStreamReader(inputStream));
String line = reader.readLine();

// find all the course abbreviations
ArrayList<String> abbrevs = new ArrayList<String>();
boolean criticalSection = false;
int start = 0;
int end = 0;
while (line != null) {
    if (line.contains("All Course Subjects")) {
        criticalSection = true;
    } else if (line.contains("/select")) {
        criticalSection = false;
    } else if (criticalSection) {
        start = line.indexOf("value=");
        end = line.indexOf(">");
        if (start >= 0 && end > 0) {
            abbrevs.add(line.substring(start + "value=".length(), end));
        }
    }
    line = reader.readLine();
}
reader.close();

// for each course abbreviation, download as many HTML pages as specified by PAGES_PER_ABBREV
for (int a = 0; a < abbrevs.size(); a++) {
    System.out.println(abbrevs.get(a));
    for (int p = 0; p < PAGES_PER_ABBREV; p++) {
        loadPage(httpClient, abbrevs.get(a), p);
    }
}
} catch (Exception e) {
    e.printStackTrace();
}
Appendix B

Source Code for ScheduleToSqlite.pl

#!/usr/bin/perl

# ScheduleToSqlite.pl
# need to convert an HTML file downloaded from schedule.psu.edu into a SQL file that can
# be loaded into a SQLite database?
# boy, have I got the Perl script for you

# how to use
#-----------------
# to convert "schedule.html" into "schedule.html.sql", execute "perl
# schedule.to.sqlite.pl schedule.html" from the command-line

# let's get dangerous
use strict;
use warnings;

# no command-line arguments, Seymour?
if (@ARGV < 1) {
  die "Feed me command-line arguments, Seymour! \n"
}

# keep feeding me those delicious command-line arguments, Seymour
while (@ARGV) {
  # feed me a tasty filename, Seymour
  my $filename = shift @ARGV;

  # open INPUTFILE
  # z.B. "schedule.html"
  open INPUTFILE, $filename or die $!

  # open OUTPUTFILE
  # z.B. "schedule.html.sql"
  open OUTPUTFILE, "$filename.sql" or die $!

  # slurp up the entire contents of INPUTFILE
  # store everything in $slurp
  my $slurp = do { local $/; <INPUTFILE> }

  # hey, yo, Reggie the T-Rex, could you do me a solid?
  # find all the course abbreviations in $slurp
  while ($slurp =~ m/(<p class="course_abbrev">)(.+)(</p>)/g) {
    # you da best, Reggie
    my $courseAbbreviation = $2;

    # Reggie, while you're at it, could you find the course name?
    # it should be nearby
    # by the way, Mr. SQLite doesn't like that "'" stuff, so you gotta escape that
    with "\'"
    $slurp =~ m/(<p class="course_longname">)(.+)(</p>)/g;
    my $courseName = $2;
    $courseName =~ s//\'/g;

    # I need the course credits too
    $slurp =~ m/(<p class="course_credits">Credits: )(.+)(</p>)/g;
    my $courseCredits = $2;

    # do something with $courseAbbreviation, $courseName, $courseCredits
  }

  # write output
  print OUTPUTFILE $slurp

  # close OUTPUTFILE
  close OUTPUTFILE;

  # I've done my duty, Seymour. You can now load the SQL file into your SQLite database.
}

# let's get dangerous
use strict;
use warnings;

# no command-line arguments, Seymour?
if (@ARGV < 1) {
  die "Feed me command-line arguments, Seymour! \n"
}

# keep feeding me those delicious command-line arguments, Seymour
while (@ARGV) {
  # feed me a tasty filename, Seymour
  my $filename = shift @ARGV;

  # open INPUTFILE
  # z.B. "schedule.html"
  open INPUTFILE, $filename or die $!

  # open OUTPUTFILE
  # z.B. "schedule.html.sql"
  open OUTPUTFILE, "$filename.sql" or die $!

  # slurp up the entire contents of INPUTFILE
  # store everything in $slurp
  my $slurp = do { local $/; <INPUTFILE> }

  # hey, yo, Reggie the T-Rex, could you do me a solid?
  # find all the course abbreviations in $slurp
  while ($slurp =~ m/(<p class="course_abbrev">)(.+)(</p>)/g) {
    # you da best, Reggie
    my $courseAbbreviation = $2;

    # Reggie, while you're at it, could you find the course name?
    # it should be nearby
    # by the way, Mr. SQLite doesn't like that "'" stuff, so you gotta escape that
    with "\'"
    $slurp =~ m/(<p class="course_longname">)(.+)(</p>)/g;
    my $courseName = $2;
    $courseName =~ s//\'/g;

    # I need the course credits too
    $slurp =~ m/(<p class="course_credits">Credits: )(.+)(</p>)/g;
    my $courseCredits = $2;

    # do something with $courseAbbreviation, $courseName, $courseCredits
  }

  # write output
  print OUTPUTFILE $slurp

  # close OUTPUTFILE
  close OUTPUTFILE;

  # I've done my duty, Seymour. You can now load the SQL file into your SQLite database.
}
# and... the course description
# which has like a bazillion unneccessary spaces in it
# you can fix that, right?
# remember, Mr. SQLite doesn't like that "'" stuff, so you gotta escape that with

```
$slurp =~ m/(<p class="course_description">)(.+)( </a>)/g;
my $courseDescription =~ s/ (2,)/ /g;
$courseDescription =~ s/''/"/g;
```

# alright, Mr. SQLite, here's a new Course record for ya
print OUTPUTFILE "INSERT OR IGNORE INTO Course(courseAbbreviation, courseName, courseCredits, courseDescription) VALUES('${$courseAbbreviation}', '${$courseName}', '${$courseCredits}', '${$courseDescription}');\n"

# Reggie, I like $slurp
# I like $slurp a lot
# but, I've got an even better variable for ya called $sections
# could you quick fill $sections with all the sections associated with the course
abbreviation you just found?
$slurp =~ m/<tr class="course_header">
     s+/g;
my $sectionsStart = pos $slurp;
$slurp =~ m/</table>/g;
my $sectionsEnd = pos $slurp;
my $sections = substr $slurp, $sectionsStart, ($sectionsEnd - $sectionsStart);

# nice work, Reggie
# now, find all the sections in $sections
while ($sections =~ m/<tr class="course_details">
    s+/g)
{
    # Reggie, I have many needs
    # first, I need the schedule number
    my $scheduleNumber = $2;
    # second, I need the section number
    $sections =~ m/<td><p>)(.+)(<
        s+/g;
    my $sectionNumber = $2;
    # third, I need the number of seats open
    # whenever the number is zero, there's this pesky <span> tag
    # please get rid of that
    # after that, please get rid of any spaces
    $sections =~ m/<td><p><span id="outputdiv$scheduleNumber" class="outputdiv">
        s+/g;
    my $seatsOpen = $2;
    $seatsOpen =~ s/<span class='red'>//g;
    $seatsOpen =~ s/< /g;
    # finally, I need the maximum number of seats
    $sections =~ m/(<p>(.+?)<
        s+/g;
    my $seatsMax = $2;

    # alright, Reggie, it's time for $times
    # please fill $times with all the times associated with the schedule number
    you just found
    $sections =~ m/<td>
        s+/g;
    my $timesStart = pos $sections;
    $sections =~ m/</td>/g;
    my $timesEnd = pos $sections;
    my $times = substr $sections, $timesStart, ($timesEnd - $timesStart);
    # we'll just push each time we find into @timesArray
    my @timesArray = ();

    # Reggie, please find all the times in $times
    while ($times =~ m/(<p>)(.+?)<
        s+/sg)
    {
# I apologize, Reggie
# you would think that a time record would only take up a single line
# nope
# also, a pesky <span> tag rears its ugly head whenever a section meets
online or by appointment
# best of luck
my $time = $2;
$time =~ s/^s+//g;
$time =~ s/s+//g;
$time =~ s/<span.+?>/g;
$time =~ s/<[^/]/g;

# push it to the limit
push @timesArray, $time;
}

# Reggie, welcome back to the wonderful world of $sections
# now, say goodbye to $sections and say hello to $rooms
# please fill $rooms with all the rooms associated with the schedule number
you just found
$sections =~ m/<td>s+/g;
my $roomsStart = pos $sections;
$sections =~ m/<\/td>/g;
my $roomsEnd = pos $sections;
my $rooms = substr $sections, $roomsStart, ($roomsEnd - $roomsStart);

# we'll just push each room we find into @roomsArray
my @roomsArray = ();

# Reggie, please find all the rooms in $rooms
# sometimes, the ending </p> tag gets dropped
# if that happens, used the ending </a> tag instead
while ($rooms =~ m/(<p>)(.+?)((<\/p>)|(<\/a>)))/sg) {
  # all you need to do, Reggie, is remove the silly <a> stuff and the silly
extra spaces
  # if you encounter, "&nbsp;", just replace it with "n/a"
  # remember, Mr. SQLite doesn't like that "" stuff, so you gotta escape
  my $room = $2;
  $room =~ s/<a.+?>//g;
  $room =~ s/<[^/]/g;
  $room =~ s/\s+/$\n/a/g;
  $room =~ s/'/'/''/g;

  # push it to the limit
  push @roomsArray, $room;
}

# rooms are stupid, Reggie
# if the number of rooms does not match the number of times, we're just going
to keep pushing "n/a" into @roomsArray until it does
while (@timesArray - @roomsArray) {
  push @roomsArray, "n/a";
}

# Reggie, say hello to $sections
# Reggie, say goodbye to $sections
# Reggie, say hello to $instructor
# please fill $instructor with the instructor associated with the schedule
number you just found
$sections =~ m/<td>s+/g;
my $instructorStart = pos $sections;
$sections =~ m/<\/td>/g;
my $instructorEnd = pos $sections;
my $instructor = substr $sections, $instructorStart, ($instructorEnd - $instructorStart);
# Reggie, please find the instructor name in $instructor
my $instructorName;
if ($instructor =~ m/{<p><a href="http://">(.+)(</p><td>)/g) {
    # Reggie, you did it
    # remember, Mr. SQLite doesn't like that "'" stuff, so you gotta escape
    $instructorName =~ s/'/'/g;
} else {
    # Reggie, I forgive you
    # some weird sections don't have instructors
    # we'll just use "n/a"
    $instructorName = "n/a";
}

# alright, Mr. SQLite, here's a new Section record for ya
print OUTPUTFILE "INSERT OR IGNORE INTO Section(scheduleNumber, courseAbbreviation, sectionNumber, seatsOpen, seatsMax, instructorName)
VALUES('$scheduleNumber', '$courseAbbreviation', '$sectionNumber', '$seatsOpen', '$seatsMax', '$instructorName');"

# for each time in @timesArray...
while (@timesArray) {
    # shift out a time
    my $time = shift @timesArray;
    my $room = shift @roomsArray;
    # Reggie, work your magic
    # I know there's at least one meeting day, but there could be as many as five meeting days
    # after that, the start hours, start minutes, start am/pm, end hours, end minutes, and end am/pm shouldn't be too hard to figure out
    # ten-hut, make sure to convert all times to military time
    if ($time =~ m/(Y|M|T|W|R|F|S)?\d?d:\d?d( ?)((A|P)M)-\d?d:\d?d( ?)((A|P)M)/g) {
        # Reggie, you did it
        # make sure to report all the new SectionRoom records to Mr. SQLite
        my $day1 = $1;
        my $day2 = $3 || "n/a";
        my $day3 = $6 || "n/a";
        my $day4 = $9 || "n/a";
        my $day5 = $12 || "n/a";
        my $startHours = $15;
        my $startMinutes = $19;
        my $startAmPm = "$endAmPm";

        if ($startAmPm eq "FM") {
            if ($startHours < 12) {
                $startHours += 12;
            }
        }
        else {
            if ($starthours == 12) {
                $starthours = 0;
            }
        }
        if (length("$starthours") < 2) {
            $starthours = "0" . $starthours;
        }
        my $endHours = $22;
        my $endMinutes = $24;
        my $endAmPm = "$endAmPm";
        if ($endAmPm eq "FM") {
if ($endHours < 12) {
    $endHours += 12;
} else {
    if ($endHours == 12) {
        $endHours = 0;
    }
}

if (length("$endHours") < 2) {
    $endHours = "0".$endHours;
}

$dayOfWeekToDate = ( "Y" => "1", "M" => "2", "W" => "3", "D" => "4", "R" => "5", "P" => "6", "S" => "7" );

my $day1StartTime = "2013-01-0".$dayOfWeekToDate{$day1}.
$day1EndTime = "2013-01-0".$dayOfWeekToDate{$day1}.

print OUTPUTFILE "INSERT OR IGNORE INTO SectionRoom(scheduleNumber, roomName, startTime, endTime) VALUES(''$scheduleNumber', '$room', '
$day1StartTime', '$day1EndTime');";

if ($day2 ne "n/a") {
    my $day2StartTime = "2013-01-0".$dayOfWeekToDate{$day2}.
    $day2EndTime = "2013-01-0".$dayOfWeekToDate{$day2}.

    print OUTPUTFILE "INSERT OR IGNORE INTO SectionRoom(scheduleNumber, roomName, startTime, endTime) VALUES(''$scheduleNumber', 'S'$room', '$day2StartTime', '
$day2EndTime');";

    if ($day3 ne "n/a") {
        my $day3StartTime = "2013-01-0".$dayOfWeekToDate{$day3}.
        $day3EndTime = "2013-01-0".$dayOfWeekToDate{$day3}.

        print OUTPUTFILE "INSERT OR IGNORE INTO SectionRoom(scheduleNumber, roomName, startTime, endTime) VALUES(''$scheduleNumber', 'R'$room', '$day3StartTime', '
$day3EndTime');";

        if ($day4 ne "n/a") {
            my $day4StartTime = "2013-01-0".$dayOfWeekToDate{$day4}.
            $day4EndTime = "2013-01-0".$dayOfWeekToDate{$day4}.

            print OUTPUTFILE "INSERT OR IGNORE INTO SectionRoom(scheduleNumber, roomName, startTime, endTime) VALUES(''$scheduleNumber', 'APPT'$room', '$day4StartTime', '
$day4EndTime');";

            if ($day5 ne "n/a") {
                my $day5StartTime = "2013-01-0".$dayOfWeekToDate{$day5}.
                $day5EndTime = "2013-01-0".$dayOfWeekToDate{$day5}.

                print OUTPUTFILE "INSERT OR IGNORE INTO SectionRoom(scheduleNumber, roomName, startTime, endTime) VALUES(''$scheduleNumber', 'WEB'$room', '$day5StartTime', '
$day5EndTime');";
            } else {
                # Reggie, I forgive you
                # sometimes you get times like "WEB", "AND WEB", and "APPT"
                # just go ahead and report a new SectionRoom record to Mr. SQLite
                print OUTPUTFILE "INSERT OR IGNORE INTO SectionRoom(scheduleNumber, roomName, startTime, endTime) VALUES(''$scheduleNumber', '$room', '$time', '$time');";
            }
        } else {
            print OUTPUTFILE "INSERT OR IGNORE INTO SectionRoom(scheduleNumber, roomName, startTime, endTime) VALUES(''$scheduleNumber', '$room', '$time', '$time');";
        }
    } else {
        print OUTPUTFILE "INSERT OR IGNORE INTO SectionRoom(scheduleNumber, roomName, startTime, endTime) VALUES(''$scheduleNumber', '$room', '$time', '$time');";
    }
} else {
    if ($endMinutes < 12) {
        $endMinutes += 12;
    } else {
        if ($endMinutes == 12) {
            $endMinutes = 0;
        }
    }

    if (length("$endMinutes") < 2) {
        $endMinutes = "0".$endMinutes;
    }

    $startHours = $endHours;
    my $day1EndTime = "2013-01-0".$dayOfWeekToDate{$day1}.
    $day2StartTime = "2013-01-0".$dayOfWeekToDate{$day2}.
    $day3StartTime = "2013-01-0".$dayOfWeekToDate{$day3}.
    $day4StartTime = "2013-01-0".$dayOfWeekToDate{$day4}.
    $day5StartTime = "2013-01-0".$dayOfWeekToDate{$day5}.

    print OUTPUTFILE "INSERT OR IGNORE INTO SectionRoom(scheduleNumber, roomName, startTime, endTime) VALUES(''$scheduleNumber', 'W'$room', '$day1EndTime', '
$day2StartTime');";

    if ($day3 ne "n/a") {
        my $day3EndTime = "2013-01-0".$dayOfWeekToDate{$day3}.
        $day4StartTime = "2013-01-0".$dayOfWeekToDate{$day4}.

        print OUTPUTFILE "INSERT OR IGNORE INTO SectionRoom(scheduleNumber, roomName, startTime, endTime) VALUES(''$scheduleNumber', 'WEB'$room', '$day3EndTime', '
$day4StartTime');";

        if ($day5 ne "n/a") {
            my $day5EndTime = "2013-01-0".$dayOfWeekToDate{$day5}.
            $day6StartTime = "2013-01-0".$dayOfWeekToDate{$day6}.

            print OUTPUTFILE "INSERT OR IGNORE INTO SectionRoom(scheduleNumber, roomName, startTime, endTime) VALUES(''$scheduleNumber', 'APPT'$room', '$day5EndTime', '
$day6StartTime');";
        } else {
            print OUTPUTFILE "INSERT OR IGNORE INTO SectionRoom(scheduleNumber, roomName, startTime, endTime) VALUES(''$scheduleNumber', 'WEB'$room', '$day5EndTime', '$time');";
        }
    } else {
        print OUTPUTFILE "INSERT OR IGNORE INTO SectionRoom(scheduleNumber, roomName, startTime, endTime) VALUES(''$scheduleNumber', 'WEB'$room', '$time', '$time');";
    }
} else {
    print OUTPUTFILE "INSERT OR IGNORE INTO SectionRoom(scheduleNumber, roomName, startTime, endTime) VALUES(''$scheduleNumber', 'WEB'$room', '$time', '$time');";
}
print OUTPUTFILE "\n";
}
}

# close OUTPUTFILE
# z.B. "schedule.html.sql"
close OUTPUTFILE or die $!;

# close INPUTFILE
# z.B. "schedule.html"
close INPUTFILE or die $!;
Appendix C

Source Code for ClassroomsUnlocked

occupied.php

```php
<?php
require "simplejson.php";

$get = false;
$post = true;
$db = "psu.db";
$sql;
$startTime;
$endTime;

if ($get && isset($_GET["startTime"]) && isset($_GET["endTime"])) {
    $startTime = $_GET["startTime"];
    $endTime = $_GET["endTime"];
}

if ($post && isset($_POST["startTime"]) && isset($_POST["endTime"])) {
    $startTime = $_POST["startTime"];
    $endTime = $_POST["endTime"];
}

if (isset($startTime) && isset($endTime)) {
    $json = array();
    $dbName = $db;
    $sqlError;
    $db = sqlite_open($dbName, 0666, $sqlError);
    if ($db) {
        $sql = "SELECT DISTINCT SectionRoom.roomName FROM SectionRoom WHERE SectionRoom.roomName IN ('110 IST Building', '113 IST Building', '202 IST Building', '203 IST Building', '205 IST Building', '206 IST Building', '208 IST Building', '210 IST Building') AND SectionRoom.startTime <= " . $endTime . " AND SectionRoom.endTime >= " . $startTime . " ORDER BY SectionRoom.roomName;";
        $result = sqlite_query($db, $sql, $sqlError);
        if ($result) {
            $rowNum = 0;
            while ($row = sqlite_fetch_array($result, SQLITE_ASSOC)) {
                $json[$rowNum] = $row;
                $rowNum++;
            }
        }
    }
    echo toJson($json);
    sqlite_close($db);
}
?>
```
<?php

$db = "psu.db";
$sql;
$roomName;
$dayOfWeek;

if ($get && isset($_GET["roomName"]))
    $roomName = $_GET["roomName"]; $dayOfWeek = $_GET["dayOfWeek"]; }

if ($post && isset($_POST["roomName"]))
    $roomName = $_POST["roomName"]; $dayOfWeek = $_POST["dayOfWeek"]; }

if (isset($roomName) && isset($dayOfWeek))
    echo "<center>", "<h1>" . $roomName . "</h1>" . $dayOfWeek . "</h1>";
$dbFile = $db;
$sqlError;
$db = sqlite_open($dbFile, 0666, $sqlError);
if ($db) {
    $sql = "SELECT * FROM SectionRoom, Section, Course WHERE SectionRoom.roomName = ". $roomName . " AND SectionRoom.startTime LIKE " . $dayOfWeek . "%' AND SectionRoom.endTime LIKE " . $dayOfWeek . "%' AND SectionRoom.scheduleNumber = Section.scheduleNumber AND Course.courseAbbreviation = Section.courseAbbreviation ORDER BY SectionRoom.startTime;";
    $result = sqlite_query($db, $sql, $sqlError);
    if ($result) {
        "<table cellpadding="10" cellspacing="0">
        <tr><th>Start</th><th>End</th><th>Course</th><th>Instructor</th><th>Section</th></tr>
        while ($row = sqlite_fetch_array($result, SQLITE_ASSOC)) {
            "<tr<td>" . militaryTimeToAmPmTime(substr($row["SectionRoom.startTime"], -5)) .. "</td>" . militaryTimeToAmPmTime(substr($row["SectionRoom.endTime"], -5)) .. "</td>" . "<td>" . $row["Course.courseName"] .. "</td>" . "<td>" . $row["Section.instructorName"] .. "</td>" . "<td>" . $row["Section[sectionNumber"] .. "</td>";"
```php
function militaryTimeToAmPmTime($time) {
    $parts = explode(":" , $time);
    $hours = intval($parts[0]) % 24;
    $minutes = intval($parts[1]) % 60;
    $amPm;
    if ($hours < 12) {
        $amPm = "am";
    } else {
        $amPm = "pm";
    }
    if ($hours <= 0 && $minutes == 0) {
        return "Midnight";
    } else if ($hours == 12 && $minutes == 0) {
        return "Noon";
    } else {
        // 0 hours => 12 hours
        // subtract hours by twelve when "necesse est"
        if ($hours == 0) {
            $hours = 12;
        } else if ($hours > 12) {
            $hours = $hours - 12;
        }
        // "1" hour => "01" hour
        $hoursString = "" . $hours;
        if (strlen($hoursString) < 2) {
            $hoursString = "0" . $hoursString;
        }
        // "1" minute => "01" minute
        $minutesString = "" . $minutes;
        if (strlen($minutesString) < 2) {
            $minutesString = "0" . $minutesString;
        }
        // return "hh:mm (a/p)m"
        return $hoursString . ":" . $minutesString . " " . $amPm;
    }
}

?>
```

```
<doctyple html>
<html lang="en">
<head>
    <meta charset="utf-8" />
    <!-- external resources -->
    <link rel="stylesheet" href="http://code.jquery.com/ui/1.10.1/themes/start/jquery-ui.min.css" />
    <script src="http://code.jquery.com/jquery-latest.min.js"></script>
    <script src="http://code.jquery.com/ui/1.10.1/jquery-ui.min.js"></script>
    <script src="https://www.google.com/jsapi?key=AIzaSyCtzD2opMseg_w8Xc51R4htpmEHNgItBoKA"></script>
    <!-- internal resources -->
    <link rel="stylesheet" href="SpatiotemporalUi.css" />
    <script src="Room.js"></script>
    <script src="IstBuilding.js"></script>
```
<script src="SlidingParagraph.js"></script>
<script src="DayOfWeekSlider.js"></script>
<script src="AmPmTimeSlider.js"></script>
<script src="Time.js"></script>
<script src="TypeCheck.js"></script>
<script src="Main.js"></script>
<title>Classrooms Unlocked</title>
</head>
<body>
<div id="main">
  <div id="spatio">
    <form id="floor-form">
      <label>Select a Floor:</label>
      <select id="floor-select">
        <option value="2" selected="true">Level 2</option>
        <option value="1">Level 1</option>
      </select>
    </form>
  </div>
  <div id="google-earth"></div>
</div>
<div id="temporal">
  <div id="day-of-week"></div>
  <div id="am-pm-time"></div>
</div>
</body>
</html>

SpatiotemporalUi.css

#main {
  font-family: arial, sans-serif;
  color: black;
  font-size: 12pt;
}

#spatio {
  width: 94%;
  height: 48.5%;
  margin-left: 3%;
}

#floor-form {
  height: 10%;
  margin-left: auto;
  margin-right: auto;
  text-align: center;
}

#google-earth {
  width: 40%;
  height: 85%;
  margin-left: auto;
  margin-right: auto;
}

#temporal {
  width: 94%;
  height: 48.5%;
  margin-left: 3%;
  margin-top: 3%;
}
// Main.js

// define constants
--------------------
// the golden ratio
var GOLDEN_RATIO = 1.61;

// define globals
----------------
// hey, yo, Google, hook me up with the latest stable version of Google Earth 1.0
google.load("earth", "1");

// let's create a global variable to hold our instance of Google Earth 1.0
var ge = null;

// let's create global variables to hold important values
var istBuilding = null;
var dayOfWeekSlider = null;
var amPmTimeSlider = null;

// on load
--------
$(function() {
    // load the floor selector
    $('#floor-select').change(function(e) {
        istBuilding.setFloor(parseInt($('#floor-select').val()));
    });

    // size the width of Google Earth according to the Golden Ratio
    $('#google-earth').css('height', $('#google-earth').width() / GOLDEN_RATIO);

    // auto-resize Google Earth according to the Golden Ratio
    $(window).resize(function() {
        $('#google-earth').css('height', $('#google-earth').width() / GOLDEN_RATIO);
    });

    // Google Earth 1.0, "google-earth" is where I want you
    google.earth.createInstance("google-earth", function(instance) {
        ge = instance;
        // just our luck, the window and the navigation controls are invisible by default
        // also, the terrain is always getting in the way
        // we gotta change that
        ge.getNavigationControl().setVisibility(true);
        ge.getLayerRoot().enableLayerById(ge.LAYER_TERRAIN, false);
        // load everything else
        istBuilding = new IstBuilding(ge, parseInt($('#floor-select').val()));
    });

    // create the day of week and am/pm time sliders
    var now = new Date();
    var dayOfWeek = dateToDayOfWeek("2013-01-0" + (now.getDay() + 1));
    var totalMinutes = (now.getHours() * 60) + (now.getMinutes() % 60);
    var hours = Math.floor(totalMinutes / 60);
    var minutes = totalMinutes - (hours * 60);
    var hours2 = Math.floor(totalMinutes2 / 60);
    var minutes2 = totalMinutes2 - (hours2 * 60);
    dayOfWeekSlider = new DayOfWeekSlider("#day-of-week", dayOfWeek, dayOfWeekAmPmTimeChanged);
    amPmTimeSlider = new AmPmTimeSlider("#am-pm-time", hours, minutes, hours2, minutes2, dayOfWeekAmPmTimeChanged);
});
dayOfWeekSlider.set(dayOfWeek);
amPmTimeSlider.set(hours, minutes, minutes2);
}
);  
// do nothing
});

// handles temporal changes
function dayOfWeekAmPmTimeChanged() {
  // convert to military time
  var dateMilitaryTime = dayOfWeekToDate(dayOfWeekSlider.dayOfWeek) + " " +
    militaryTime(amPmTimeSlider.hours, amPmTimeSlider.minutes);
  var dateMilitaryTime2 = dayOfWeekToDate(dayOfWeekSlider.dayOfWeek) + " " +
    militaryTime(amPmTimeSlider.hours, amPmTimeSlider.minutes);
  // ajax post
  $.ajax({
    url: "occupied.php",
    type: "post",
    data: "startTime=" + dateMilitaryTime +
      "&endTime=" + dateMilitaryTime2,
    dataType: "json",
    cache: false
  }).done(function (json) {
    // tell the IST Building about the occupied rooms
    var occupiedRoomNames = [];
    for (var i = 0; i < json.length; i++) {
      occupiedRoomNames.push([json[i]["SectionRoom.roomName"]]);
    }
    istBuilding.setOccupiedRoomNames(occupiedRoomNames, dayOfWeekSlider.dayOfWeek);
  });
}

IstBuilding.js

function IstBuilding(ge, floor) {
  //------------------------
  // define constants
  //------------------------

  // wanna change the absolute URLs or the KML IDs for the IST Building 3D models?
  // go ahead and modify this constant
  this.BUILDING_MODELS = {
    1: "http://www.personal.psu.edu/btd5030/thesis/IstBuildingFloor1.kmz",
    2: "http://www.personal.psu.edu/btd5030/thesis/IstBuildingFloor2.kmz"
  }

  // wanna change the absolute coordinates of the rooms?
  // go ahead and modify this constant
  this.ROOM_COORDS = {
    "110 IST Building": "77.86795522177843,40.7936026720291,5",
    "113 IST Building": "77.869011787469734,10 - 77.8687724699734,10 - 77.8687624511762,40.7935984411265,10 - 77.8687866997974,10 - 77.8687666864794,10 - 77.86888565681736,10 - 77.869004261052,40.79373983421439,10 - 77.8690014166356,40.79357158986993,10　
    "202 IST Building": "77.86795522177843,40.7936026720291,5 - 77.8686724699734,10 - 77.8687624511762,40.7935984411265,10 - 77.8687866997974,10 - 77.8687666864794,10 - 77.86888565681736,10 - 77.869004261052,40.79373983421439,10 - 77.8690014166356,40.79357158986993,10　
  }
}
"206 IST Building": "-77.867645678213419, 40.7938141266904710 -, 77.86756859744982, 40.79380594898834, 10 -, 77.86757290095968, 40.79372490172975, 10 -, 77.86767446998512, 40.79381353076484, 10 -, 77.86777364956818, 40.79373557673745, 10 -", "208 IST Building": "-77.86777364956818, 40.79373557673745, 10 -, 77.86767446998512, 40.79372490172975, 10 -, 77.86767446998512, 40.79381353076484, 10 -, 77.86777364956818, 40.79380594898834, 10 -", "210 IST Building": "-77.86769678055163, 40.79367748838544, 10 -, 77.86779384626887, 40.79360810564207, 10 -, 77.86767446998512, 40.79373557673745, 10 -", "212 IST Building": "-77.86769678055163, 40.79373557673745, 10 -, 77.86779384626887, 40.79360810564207, 10 -, 77.86767446998512, 40.79373557673745, 10 -", "214 IST Building": "-77.86777364956818, 40.79373557673745, 10 -, 77.86767446998512, 40.79372490172975, 10 -, 77.86767446998512, 40.79381353076484, 10 -, 77.86777364956818, 40.79380594898834, 10 -", "216 IST Building": "-77.86777364956818, 40.79372490172975, 10 -, 77.86767446998512, 40.79372490172975, 10 -, 77.86767446998512, 40.79381353076484, 10 -, 77.86777364956818, 40.79380594898834, 10 -"};

// wanna change the room colors?
// go ahead and modify this constant
this.ROOM_COLORS = {
  available: "ff00ff00",
  occupied: "ff0000ff"
};

// guard against bad parameters
// also, save all parameters
//-------------------------------
if (!ge) {
  throw "Google Earth must exist";
}
// save Google Earth
this.ge = ge;

// floor must be a positive integer in the closed interval [1,2], else throw exception
if (!isUint(floor) || floor < 1 || floor > 2) {
  throw "floor must be a positive integer in the closed interval [1,2]";
}
// save floor
this.floor = floor;

// more variables
//---------------------

// we'll need an object to keep track of the building models
this.buildingModels = {
  1: null,
  2: null
};

// we'll need an object to keep track of all rooms by floor
this.floors = {
  1: {
    "110 IST Building": {
      kml: null,
      available: true
    }
  },
  2: {
    "113 IST Building": {
      kml: null,
      available: true
    },
    "202 IST Building": {
      kml: null,
      available: true
    },
    "203 IST Building": {
      kml: null,
      available: true
    }
  }
};
"205 IST Building" : {
    kml : null,
    available : true
},
"206 IST Building" : {
    kml : null,
    available : true
},
"208 IST Building" : {
    kml : null,
    available : true
},
"210 IST Building" : {
    kml : null,
    available : true
}

// we'll need to keep track of the names of the occupied rooms
this.occupiedRoomNames = [];

// we'll need to keep track of the current start time and end time
this.startTime = null;
this.endTime = null;

// create a temporary "me" pointer
var me = this;

//------------
// initialize
//------------

// load the rooms
for (var x in me.ROOM_COORDS) {
    // load the room
    var room = new Room(ge, x, me.ROOM_COORDS[x], me.ROOM_COLORS["available"], true);

    // insert the room into floor 1 and floor 2 when necessary
    for (var i = 1; i < 3; i++) {
        for (var y in me.floors[i]) {
            if (x == y) {
                me.floors[i][y]["kml"] = room;
                me.floors[i][y]["kml"].setVisible(i == me.floor);
                break;
            }
        }
    }
}

// load the first floor
google.earth.fetchKml(ge, this.BUILDING_MODELS[1], function(kml) {
    // modify and append the first floor
    me.buildingModels[1] = kml;
    ge.getFeatures().appendChild(me.buildingModels[1]);
    me.buildingModels[1].setVisibility(1 == me.floor);
}

// load the second floor
google.earth.fetchKml(ge, me.BUILDING_MODELS[2], function(kml) {
    // modify and append the second floor
    me.buildingModels[2] = kml;
    ge.getFeatures().appendChild(me.buildingModels[2]);
    me.buildingModels[2].setVisibility(2 == me.floor);
}

// fly
var lookAt = ge.createLookAt(""");
lookAt.setHeading(-160.0);
lookAt.setTilt(45.0);
lookAt.setLatitude(40.79381897249);
lookAt.setLongitude(-77.86817842992);
lookAt.setRange(200.0);
ge.getView().setAbstractView(lookAt);
});
});

// set occupied room names
IstBuilding.prototype.setOccupiedRoomNames = function(occupiedRoomNames, dayOfWeek) {
  // guard against bad parameters
  // also, save all parameters
  // ------------------------------
  // occupied room names must be an object, else throw an exception
  if (!isObject(occupiedRoomNames)) {
    throw "occupied room names must be an object";
  }
  // save occupied room names
  this.occupiedRoomNames = occupiedRoomNames;

  // day of week must be a string, else throw exception
  if (!isString(dayOfWeek)) {
    throw "day of week must be a string";
  }
  // save day of week
  this.dayOfWeek = dayOfWeek;

  // ----------------------
  // re-adjust
  // ----------------------
  for (var i = 1; i < 3; i++) {
    for (var x in this.floors[i]) {
      var occupied = false;
      for (var y = 0; y < occupiedRoomNames.length; y++) {
        if (x == occupiedRoomNames[y]) {
          occupied = true;
          this.floors[i][x]["kml"].setName(x + " - <a target="_blank"
href="room.php?roomName=" + encodeURIComponent(x) + "&dayOfWeek=" +
encodeURIComponent(dayOfWeek) + "&"+OCCUPIED+</a>");
        }
      }
      if (this.floors[i][x]["available"]) {
        this.floors[i][x]["available"] = false;
        if (this.prevSelectedRoom != this.floors[i][x]) {
          this.floors[i][x]["kml"].setColor(this.ROOM_COLORS["occupied"]);
        }
      }
      break;
    }
  }
  if (!occupied) {
    this.floors[i][x]["kml"].setName(x + " - <a target="_blank"
href="room.php?roomName=" + encodeURIComponent(x) + "&dayOfWeek=" +
encodeURIComponent(dayOfWeek) + "&"+AVAILABLE</a>");
    if (!this.floors[i][x]["available"]) {
      this.floors[i][x]["available"] = true;
      if (this.prevSelectedRoom != this.floors[i][x]) {
        this.floors[i][x]["kml"].setColor(this.ROOM_COLORS["available"]);
      }
    }
  }
}
// set floor
IstBuilding.prototype.setFloor = function(floor) {
  //----------------------------------------
  // guard against bad parameters
  // also, save all parameters
  //----------------------------------------

  // floor must be a positive integer in the closed interval [1,2], else throw exception
  if (!isUint(floor) || floor < 1 || floor > 2) {
    throw "floor must be a positive integer in the closed interval [1,2]";
  }
  // save floor
  this.floor = floor;

  //----------------------------------------
  // re-adjust
  //----------------------------------------

  // reset floor visibility
  this.buildingModels[1].setVisibility(1 == floor);
  this.buildingModels[2].setVisibility(2 == floor);

  // reset room visibility
  for (var i = 1; i < 3; i++) {
    for (var x in this.floors[i]) {
      this.floors[i][x]["kml"].setVisible(i == floor);
    }
  }
}

Room.js

// requires "https://www.google.com/jsapi"
// requires "TypeCheck.js"

function Room(ge, name, coords, color, visible) {
  //----------------------------------------
  // guard against bad parameters
  // also, save all parameters
  //----------------------------------------

  // Google Earth must exist, else throw exception
  if (!ge) {
    throw "Google Earth must exist";
  }
  // save Google Earth
  this.ge = ge;

  // name must be a string, else throw exception
  if (!isString(name)) {
    throw "name must be a string";
  }
  // save name
  this.name = name;

  // coords must be a string, else throw exception
  if (!isString(coords)) {
    throw "coords must be a string";
  }
  // save coords
  this.coords = coords;

  // color must be a string, else throw exception
  if (!isString(color)) {
    throw "color must be a string";
  }
}
// save color
this.color = color;

// visible must be a boolean, else throw an exception
if (!isBoolean(visible)) { throw "visible must be a boolean";
}
// save visible
this.visible = visible;

// set aside a variable to hold the KML object representation of this room
this.kml = null;

// create a temporary "me" pointer
var me = this;

// initialize room

// create the KML object representation of this room
this.kml = ge.parseKml("<Placemark id=" + name + "">
  "<name>" + name + "</name>
  "<visibility>" + (visible ? "1" : "0") + "</visibility>
  "<Polygon>
    "<tessellate>1</tessellate>
    "<altitudeMode>absolute</altitudeMode>
    "<outerBoundaryIs>
      "<LinearRing>
        "<coordinates>" +
        "coords" +
        "</coordinates>
      "</LinearRing>
    "</outerBoundaryIs>
  "</Polygon>
</Placemark>"
);
// style the KML object
this.kml.setStyleSelector(ge.createStyle(""));
this.kml.getStyleSelector().getLineStyle().getColor().set(color);
this.kml.getStyleSelector().getPolyStyle().getColor().set(color);
// append the KML object
google.earth.getFeatures().appendChild(this.kml);
// listen for clicks
google.earth.addEventListener(ge.getWindow(), "click", function(e) {
  // what's the target of this event
  var target = e.getTarget();
  // if we're the target...
  if (target.getType() == "KmlPlacemark" && target.getId() == me.name) {
    // if clicked callback is a function, pass this event to it
    if (isFunction(clickedCallback)) {
      clickedCallback(e);
    }
  }
});

// set name
Room.prototype.setName = function(name) {
  //-------------------------------
  // guard against bad parameters
  // also, save all parameters
  //-------------------------------
// name must be a string, else throw exception
if (!isString(name)) {
  throw "name must be a string";
}
// save name
this.name = name;

// reset the KML object's name
this.kml.setName(name);

// set color
Room.prototype.setColor = function(color) {
  // guard against bad parameters
  // also, save all parameters
  //-------------------------------
  // color must be a string, else throw exception
  if (!isString(color)) {
    throw "color must be a string";
  }
  // save color
  this.color = color;

  // reset the KML object's styling
  this.kml.getStyleSelector().getLineStyle().getColor().set(color);
  this.kml.getStyleSelector().getPolyStyle().getColor().set(color);
}

// set visible
Room.prototype.setVisible = function(visible) {
  // guard against bad parameters
  // also, save all parameters
  //------------------------
  // visible must be a boolean, else throw an exception
  if (!isBoolean(visible)) {
    throw "visible must be a boolean";
  }
  // save visible
  this.visible = visible;

  // reset the KML object's visibility
  this.kml.setVisibility(visible);
}

Time.js

// requires "type.check.js"

// day of week to date
function dayOfWeekToDate(dayOfWeek) {
  // do the conversion
var conversion = {
    "Sunday" : "2013-01-01",
    "Monday" : "2013-01-02",
    "Tuesday" : "2013-01-03",
    "Wednesday" : "2013-01-04",
    "Thursday" : "2013-01-05",
    "Friday" : "2013-01-06",
    "Saturday" : "2013-01-07"
};

var result = conversion[dayOfWeek]

// if the conversion failed, the day of week must be wrong
// throw an exception
if (isUndefined(result)) {
    throw "day of week must be Sunday, Monday, Tuesday, Wednesday, Thursday, Friday, or Saturday";
}

// return the conversion
return result;

// date to day of week
function dateToDayOfWeek(date) {
    // do the conversion
    var conversion = {
        "2013-01-01" : "Sunday",
        "2013-01-02" : "Monday",
        "2013-01-03" : "Tuesday",
        "2013-01-04" : "Wednesday",
        "2013-01-05" : "Thursday",
        "2013-01-06" : "Friday",
        "2013-01-07" : "Saturday"
    };
    var result = conversion[date]

    // if the conversion failed, the date must be wrong
    // throw an exception
    if (isUndefined(result)) {
        throw "date must be in the closed interval \["2013-01-01", "2013-01-07"]\";
    }

    // return the conversion
    return result;
}

// am/pm time
function amPmTime(hours, minutes) {
    // hours must be a positive integer, else throw an exception
    // modulo-force hours into the closed interval [0, 23]
    if (!isUint(hours)) {
        throw "hours must be a positive integer";
    }
    hours = hours % 24;

    // minutes must be a positive integer, else throw an exception
    // modulo-force minutes into the closed interval [0, 59]
    if (!isUint(minutes)) {
        throw "minutes must be a positive integer";
    }
    minutes = minutes % 60;

    // am or pm? that is the question
    var amPm = "";
    if (hours < 12) {
        amPm = "am";
    } else {
        amPm = "pm";
    }

    // 0 hours and 0 minutes -> return "midnight"
    // 12 hours and 0 minutes -> return "noon"
    if (hours == 0 && minutes == 0) {
        return "Midnight";
    } else if (hours == 12 && minutes == 0) {
        return "Noon";
    } else {  // hours and minutes
        return hours + " " + minutes + " " + amPm;
    }
}
return "Noon";
} else {
  // 0 hours -> 12 hours
  // subtract hours by twelve when "necesse est"
  if (hours == 0) {
    hours = 12;
  } else if (hours > 12) {
    hours = hours - 12;
  }

  // "1" hour -> "01" hour
  var hoursString = hours.toString();
  if (hoursString.length < 2) {
    hoursString = "0" + hoursString;
  }

  // "1" minute -> "01" minute
  var minutesString = minutes.toString();
  if (minutesString.length < 2) {
    minutesString = "0" + minutesString;
  }

  // return "hh:mm (a/p)m"
  return hoursString + ":" + minutesString + " " + amPm;
}

// military time
function militaryTime(hours, minutes) {
  // hours must be a positive integer, else throw an exception
  if (!isUint(hours)) {
    throw "hours must be a positive integer";
  }
  // minutes must be a positive integer, else throw an exception
  // modulo-force minutes into the closed interval [0, 59]
  if (!isUint(minutes)) {
    throw "minutes must be a positive integer";
  }
  minutes = minutes % 60;

  // "1" hour -> "01" hour
  var hoursString = hours.toString();
  if (hoursString.length < 2) {
    hoursString = "0" + hoursString;
  }

  // "1" minute -> "01" minute
  var minutesString = minutes.toString();
  if (minutesString.length < 2) {
    minutesString = "0" + minutesString;
  }

  // return "hh:mm"
  return hoursString + ":" + minutesString;
}

TypeCheck.js

// requires nothing

// undefined
function isUndefined(u) {
  return typeof u === "undefined";
}

// boolean
function isBoolean(b) {
  return typeof b === "boolean";
}

// -9007199254740990 to 9007199254740990
function isInt(n) {
    return +n === n && !(n % 1);
}

// -128 to 127
function isInt8(n) {
    return +n === n && !(n % 1) && n < 0x80 && n >= -0x80;
}

// -32768 to 32767
function isInt16(n) {
    return +n === n && !(n % 1) && n < 0x8000 && n >= -0x8000;
}

// 0 to 9007199254740990
function isUint(n) {
    return +n === n && !(n % 1) && n >= 0;
}

// 0 to 255
function isUint8(n) {
    return +n === n && !(n % 1) && n < 0x100 && n >= 0;
}

// 0 to 65535
function isUint16(n) {
    return +n === n && !(n % 1) && n < 0x10000 && n >= 0;
}

// 0 to 4294967295
function isUint32(n) {
    return +n === n && !(n % 1) && n < 0x10000000 && n >= 0;
}

// including Infinity and -Infinity but not NaN (Number.MAX_VALUE = 1.7976931348623157e+308)
function isFloat(n) {
    return +n === n;
}

// -3.4028234e+38 to 3.4028234e+38 (single precision)
function isFloat32(n) {
    return +n === n && Math.abs(n) <= 3.4028234e+38;
}

// excluding Infinity, -Infinity, and NaN (Number.MAX_VALUE = 1.7976931348623157e+308)
function isFloat64(n) {
    return +n === n && Math.abs(n) <= 1.7976931348623157e+308;
}

// string
function isString(s) {
    return typeof s === "string";
}

// xml
function isXml(x) {
    return typeof x === "xml";
}

// object
function isObject(o) {
    return typeof o === "object";
}
function isFunction(f) {
  return typeof f === "function";
}

DayOfWeekSlider.js

// requires "http://code.jquery.com/jquery-latest.min.js"
// requires "http://code.jquery.com/ui/1.10.1/jquery-ui.min.js"
// requires "http://code.jquery.com/ui/1.10.1/themes/start/jquery-ui.min.css"
// requires "SlidingParagraph"
// requires "TypeCheck.js"
// day of week slider
function DayOfWeekSlider(jQuerySelector, dayOfWeek, valueChangedCallback) {
  // guard against bad parameters
  // also, save all parameters
  //-------------------------------
  // jQuery selector must be a string, else throw an exception
  if (!isString(jQuerySelector)) {
    throw "jQuery selector must be a string";
  }
  // jQuery selector must select an element, else throw an exception
  if (!$(jQuerySelector).length < 1) {
    throw "jQuery selector must select an element";
  }
  // save jQuery selector
  this.jQuerySelector = jQuerySelector;
  // day of week must be date-convertible, else throw an exception
  dayOfWeekToDate(dayOfWeek);
  // save day of week
  this.dayOfWeek = dayOfWeek;
  // save value changed callback
  this.valueChangedCallback = valueChangedCallback;
  //----------------
  // more variables
  //----------------
  // set aside a variable to hold the sliding paragraph
  this.slidingParagraph = null;
  // create a temporary "me" pointer
  var me = this;
  //------------------
  // initialize slider
  //------------------
  // create and append the sliding paragraph element
  $(jQuerySelector).append($("<div></div>")).addClass("day-of-week-slider-sliding-paragraph");
  // create the sliding paragraph
  var percentFromLeft = (parseInt(dayOfWeekToDate(dayOfWeek).slice(-1)) - 1) / 6;
  this.slidingParagraph = new SlidingParagraph(jQuerySelector + ".day-of-week-slider-sliding-paragraph", dayOfWeek, percentFromLeft);
  // create and append the slider element
  $(jQuerySelector).append($("<div></div>")).addClass("day-of-week-slider");
// create a function to handle any slider event
var handleAnySliderEvent = function(e, ui) {
    // possibly, a new value
    var dayOfWeek = dateToDayOfWeek("2013-01-0" + ui.value);
    // reset the sliding paragraph
    var percentFromLeft = (ui.value - 1) / 6;
    me.slidingParagraph.reset(dayOfWeek, percentFromLeft);
    // new value?
    if (me.dayOfWeek != dayOfWeek) {
        // save the new value
        me.dayOfWeek = dayOfWeek;
    }
}

// create the slider
$(jQuerySelector + " .day-of-week-slider").slider({
    animate: true,
    min: 1,
    max: 7,
    step: 1,
    value: parseInt(dayOfWeekToDate(dayOfWeek).slice(-1)),
    slide: handleAnySliderEvent,
    change: function(e, ui) {
        // handle the slider event
        handleAnySliderEvent(e, ui);
        // if we have a value changed callback function, pass me to it
        if (isFunction(valueChangedCallback)) {
            valueChangedCallback(me);
        }
    }
});

// create and append the ruler element and the anchor element
$(jQuerySelector).append($("<div></div>")).addClass("day-of-week-slider-ruler");
$(jQuerySelector + " .day-of-week-slider-ruler").append($("<p></p>")).addClass("day-of-week-slider-ruler-anchor");

// set up an element for each day of the week
for (var i = 1; i < 8; i++) {
    // convert the index into a day of week
    currentDayOfWeek = dateToDayOfWeek("2013-01-0" + i);
    // create and append
    $(jQuerySelector + " .day-of-week-slider-ruler-
        anchor").append($("<span></span>")).addClass("day-of-week-slider-ruler=" +
        currentDayOfWeek.text(currentDayOfWeek.slice(0, 2)));
    // calculate the desired left offset of the element and also the current width of the element
    var leftOffset = ((i - 1) / 6) * $($(this).jQuerySelector + " .day-of-week-slider-uler-anchor").width;
    \n    var width = $($(this).jQuerySelector + " .day-of-week-slider-ruler=uler-anchor").width();
    \n    $(this).jQuerySelector + " .day-of-week-slider-ruler=" +
    currentDayOfWeek.width();
    \n    // attempt to position relative to the anchor element
    $(this).jQuerySelector + " .day-of-week-slider-ruler=" +
    currentDayOfWeek.position({
        my: "left",
        at: "left",
        of: $(this).jQuerySelector + " .day-of-week-slider-ruler-anchor",
        within: $(this).jQuerySelector + " .day-of-week-slider-ruler-anchor",
        collision: "fit"
    });
}

// whenever the window is resized...
$(window).resize(function() {
    // pretend that the slider has moved
    handleAnySliderEvent(null, { value: $(jQuerySelector + " .day-of-week-uler-slide").slider("option", "value") });
});
for (var i = 1; i < 8; i++) {
    // convert the index into a day of week
    currentDayOfWeek = dateToDayOfWeek("2013-01-0" + i);
    // calculate the desired left offset of the element and also the current
    // width of the element
    var leftOffset = ((i - 1) / 6) * $(me.jQuerySelector + ".day-of-week-slider-ruler-anchor").width();
    var width = $(me.jQuerySelector + ".day-of-week-slider-ruler-" + currentDayOfWeek).width();
    // attempt to position relative to the anchor element
    // if any problems occur, just make sure they fit inside the anchor element
    $(me.jQuerySelector + ".day-of-week-slider-ruler-" + currentDayOfWeek).position({
        my: "left" + (leftOffset - (width / 2)),
        at: "left",
        of: me.jQuerySelector + ".day-of-week-slider-ruler-anchor",
        within: me.jQuerySelector + ".day-of-week-slider-ruler-anchor",
        collision: "fit"
    });
}

DayOfWeekSlider.prototype.set = function (dayOfWeek) {
    //-------------------------------
    // guard against bad parameters
    // also, save all parameters
    //-------------------------------

    // day of week must be date-convertible, else throw an exception
    dayOfWeekToDate(dayOfWeek);
    // save day of week
    this.dayOfWeek = dayOfWeek;

    //-------------------
    // re-adjust
    //-------------------

    // reset the sliding paragraph
    var percentFromLeft = ((parseInt(dayOfWeekToDate(dayOfWeek).slice(-1)) - 1) / 6);
    this.slidingParagraph.reset({dayOfWeek, percentFromLeft});

    // reset the slider
    $(this.jQuerySelector + ".day-of-week-slider").slider("value",
        parseInt(dayOfWeekToDate(dayOfWeek).slice(-1)));
}

AmPmTimeSlider.js

// requires "http://code.jquery.com/jquery-latest.min.js"
// requires "http://code.jquery.com/ui/1.10.1/jquery-ui.min.js"
// requires "http://code.jquery.com/ui/1.10.1/themes/start/jquery-ui.min.css"
// requires "SlidingParagraph"
// requires "Time.js"
// requires "TypeCheck.js"

// am/pm time slider
function AmPmTimeSlider(jQuerySelector, hours, minutes, hours2, minutes2, valueChangedCallback) {
    //-------------------------------
    // define constants
    //-------------------------------

    // wanna change the range of the slider?
    // go ahead and modify these two constants
this.MIN_VALUE_IN_MINUTES = 0;
this.MAX_VALUE_IN_MINUTES = 1440;

// wanna change the step-size of the slider?
// go ahead and modify this constant
this.STEP_SIZE_IN_MINUTES = 5;

// wanna change the ruler text for morning, midnight, am, noon, pm, and evening?
// go ahead and modify these three constants
this.MORNING_MIDNIGHT = "Midnight";
this.PM = "PM";
this.NOON = "Noon";
this.AM = "AM";
this.EVENING_MIDNIGHT = "Midnight";

// guard against bad parameters
// also, save all parameters

// jQuery selector must be a string, else throw an exception
if (!isString(jQuerySelector)) {
    throw "jQuery selector must be a string";
}
// jQuery selector must select an element, else throw an exception
if (!$(jQuerySelector).length < 1) {
    throw "jQuery selector must select an element";
}
// save jQuery selector
this.$jQuerySelector = jQuerySelector;

// hours must be a positive integer in the closed interval [0, 24], else throw an exception
if (!isUint(hours) || hours < 0 || hours > 24) {
    throw "hours must be a positive integer in the closed interval [0, 24]";
}
// save hours
this.hours = hours;

// minutes must be a positive integer divisible by 5 and in the closed interval [0, 59], else throw an exception
if (!isUint(minutes) || minutes % 5 != 0 || minutes < 0 || minutes > 59) {
    throw "minutes must be a positive integer divisible by 5 and in the closed interval [0, 59]";
}
// save minutes
this.minutes = minutes;

// hours 2 must be a positive integer greater than hours and in the closed interval [0, 24], else throw an exception
if (!isUint(hours2) || hours2 < hours || hours2 < 0 || hours2 > 24) {
    throw "hours 2 must be a positive integer greater than hours and in the closed interval [0, 24]";
}
// save hours 2
this.hours2 = hours2;

// minutes 2 must be a positive integer divisible by 5 and in the closed interval [0, 59], else throw an exception
if (!isUint(minutes2) || minutes2 % 5 != 0 || minutes2 < 0 || minutes2 > 59) {
    throw "minutes 2 must be a positive integer divisible by 5 and in the closed interval [0, 59]";
}
// save minutes 2
this.minutes2 = minutes2;

// save value changed callback
```javascript
this.valueChangedCallback = valueChangedCallback;

// more variables

// set aside a variable to hold the sliding paragraph
this.slidingParagraph = null;

// create a temporary "me" pointer
var me = this;

// initialize slider

// create and append the sliding paragraph element
$(jQuerySelector).append($('<div></div>').addClass('am-pm-time-slider-sliding-paragraph'));

// create the sliding paragraph
var percentFromLeft = ((hours * 60) + minutes) / this.MAX_VALUE_IN_MINUTES;
this.slidingParagraph = new SlidingParagraph(jQuerySelector + " .am-pm-time-slider-sliding-paragraph", amPmTime(hours, minutes) + " to " + amPmTime(hours2, minutes2), percentFromLeft);

// create and append the slider element
$(jQuerySelector).append($('<div></div>').addClass('am-pm-time-slider'));

// create a function to handle any slider event
var handleAnySliderEvent = function(e, ui) {
  // possibly, new values?
  var hours = Math.floor(ui.values[0] / 60);
  var minutes = ui.values[0] - (hours * 60);
  var hours2 = Math.floor(ui.values[1] / 60);
  var minutes2 = ui.values[1] - (hours2 * 60);
  // reset the sliding paragraph
  var percentFromLeft = ((ui.values[0] + ui.values[1]) / 2) / me.MAX_VALUE_IN_MINUTES;
  me.slidingParagraph.reset(amPmTime(hours, minutes) + " to " + amPmTime(hours2, minutes2), percentFromLeft);
  // new values?
  if (me.hours != hours || me.minutes != minutes || me.hours2 != hours2 || me.minutes2 != minutes2) {
    // save the new values
    me.hours = hours;
    me.minutes = minutes;
    me.hours2 = hours2;
    me.minutes2 = minutes2;
  }
}

// create the slider
$(jQuerySelector + " .am-pm-time-slider").slider({
  animate: true,
  min: this.MIN_VALUE_IN_MINUTES,
  max: this.MAX_VALUE_IN_MINUTES,
  step: this.STEP_SIZE_IN_MINUTES,
  range: true,
  values: [(hours * 60) + minutes, (hours2 * 60) + minutes2],
  slide: handleAnySliderEvent,
  change: function(e, ui) {
    // handle the slider event
    handleAnySliderEvent(e, ui);
    // if we have a value changed callback function, pass me to it
    if (isFunction(valueChangedCallback)) {
      valueChangedCallback(me);
    }
  }
});
```
// create and append the ruler element, the first anchor element, and the second anchor element

// attempt to position the morning midnight element, the noon element, and the evening midnight element

// if any problems occur, just make sure they fit inside the anchor element

// skip noon

// calculate the desired left offset of the element and also the current width of the element

// attempt to position the morning midnight element, the noon element, and the evening midnight element relative to the anchor element

// if any problems occur, just make sure they fit inside the anchor element
within: jQuerySelector + " .am-pm-time-slider-ruler-second-anchor",
collision: "fit"
});

$(jQuerySelector + " .am-pm-time-slider-ruler-noon").position({
  my: "left-" + ($(jQuerySelector + " .am-pm-time-slider-ruler-second-anchor").width() / 2 - ($(this).jQuerySelector + " .am-pm-time-slider-ruler-noon").width() / 2),
  at: "left",
  of: jQuerySelector + " .am-pm-time-slider-ruler-second-anchor",
  within: jQuerySelector + " .am-pm-time-slider-ruler-second-anchor",
  collision: "fit"
});

$(jQuerySelector + " .am-pm-time-slider-ruler-pm").position({
  my: "left-" + ($(jQuerySelector + " .am-pm-time-slider-ruler-second-anchor").width() / 4 - ($(jQuerySelector + " .am-pm-time-slider-ruler-pm").width() / 2),
  at: "left",
  of: jQuerySelector + " .am-pm-time-slider-ruler-second-anchor",
  within: jQuerySelector + " .am-pm-time-slider-ruler-second-anchor",
  collision: "fit"
});

$(jQuerySelector + " .am-pm-time-slider-ruler-evening-midnight").position({
  my: "left-" + ($(jQuerySelector + " .am-pm-time-slider-ruler-second-anchor").width() - ($(this).jQuerySelector + " .am-pm-time-slider-ruler-evening-midnight").width() / 2),
  at: "left",
  of: jQuerySelector + " .am-pm-time-slider-ruler-second-anchor",
  within: jQuerySelector + " .am-pm-time-slider-ruler-second-anchor",
  collision: "fit"
});

// whenever the window is resized...
$(window).resize(function() {
  // pretend that the slider has moved
  handleAnySliderEvent(null, { values: $(jQuerySelector + " .am-pm-time-slider").slider("option", "values") });
  // reposition the ruler
  for (var i = 1; i < 24; i++) {
    // skip noon
    if (i >= 12) {
      // convert to am/pm hours
      var amPmHours = (i <= 12) ? i : (i - 12);
      // calculate the desired left offset of the element and also the current width of the element
      var leftOffset = ((i * 60) / me.MAX_VALUE_IN_MINUTES) * $(me jQuerySelector + " .am-pm-time-slider-ruler-first-anchor").width();
      var width = $(me jQuerySelector + " .am-pm-time-slider-ruler-pm").width();
      $(me jQuerySelector + " .am-pm-time-slider-ruler-" + i).position({
        my: "left-" + (leftOffset - (width / 2)),
        at: "left",
        of: me jQuerySelector + " .am-pm-time-slider-ruler-first-anchor",
        within: me jQuerySelector + " .am-pm-time-slider-ruler-first-anchor",
        collision: "fit"
      });
    }
  }

  // attempt to position relative to the anchor element
  if any problems occur, just make sure they fit inside the anchor element
  $(me jQuerySelector + " .am-pm-time-slider-ruler-evening-midnight").position({
    my: "left-" + ($(me jQuerySelector + " .am-pm-time-slider-ruler-evening-midnight").width() / 2),
    at: "left",
    of: me jQuerySelector + " .am-pm-time-slider-ruler-second-anchor",
  });
});
within: me.jQuerySelector + " .am-pm-time-slider-ruler-second-anchor",
collision: "fit"
});

$(jQuerySelector + " .am-pm-time-slider-ruler-am").position({
my: "left" + (($(jQuerySelector + " .am-pm-time-slider-ruler-second-anchor").width() / 4) - ($(jQuerySelector + " .am-pm-time-slider-ruler-am").width() / 2)),
at: "left",
of: jQuerySelector + " .am-pm-time-slider-ruler-second-anchor",
within: jQuerySelector + " .am-pm-time-slider-ruler-second-anchor",
collision: "fit"
});

$(jQuerySelector + " .am-pm-time-slider-ruler-noon").position({
my: "left" + ($(me.jQuerySelector + " .am-pm-time-slider-ruler-second-anchor").width() / 2) - ($(jQuerySelector + " .am-pm-time-slider-ruler-noon").width() / 2)),
at: "left",
of: me.jQuerySelector + " .am-pm-time-slider-ruler-second-anchor",
within: me.jQuerySelector + " .am-pm-time-slider-ruler-second-anchor",
collision: "fit"
});

$(jQuerySelector + " .am-pm-time-slider-ruler-pm").position({
my: "left" + (($(jQuerySelector + " .am-pm-time-slider-second-anchor").width() / 4) - ($(jQuerySelector + " .am-pm-time-slider-ruler-pm").width() / 2)),
at: "left",
of: jQuerySelector + " .am-pm-time-slider-ruler-second-anchor",
within: jQuerySelector + " .am-pm-time-slider-ruler-second-anchor",
collision: "fit"
});

$(me.jQuerySelector + " .am-pm-time-slider-ruler-evening-midnight").position({
my: "left" + ($(me.jQuerySelector + " .am-pm-time-slider-ruler-second-anchor").width() / 2) - ($(jQuerySelector + " .am-pm-time-slider-ruler-evening-midnight").width() / 2)),
at: "left",
of: me.jQuerySelector + " .am-pm-time-slider-ruler-second-anchor",
within: me.jQuerySelector + " .am-pm-time-slider-ruler-second-anchor",
collision: "fit"
});
});
}

// set
AmPmTimeSlider.prototype.set = function(hours, minutes, hours2, minutes2) {

//-----------------------------

// guard against bad parameters
// also, save all parameters
//-----------------------------

// hours must be a positive integer in the closed interval [0, 24], else throw an exception
if (!isUint(hours) || hours < 0 || hours > 24) {
    throw "hours must be a positive integer in the closed interval [0, 24]";
}

// save hours
this.hours = hours;

// minutes must be a positive integer divisible by 5 and in the closed interval [0, 59], else throw an exception
if (!isUint(minutes) || minutes % 5 !== 0 || minutes < 0 || minutes > 59) {
    throw "minutes must be a positive integer divisible by 5 and in the closed interval [0, 59]";
}

// save minutes
this.minutes = minutes;

// hours 2 must be a positive integer greater than hours and in the closed interval [0, 24], else throw an exception
if (!isUint(hours2) || hours2 < hours || hours2 < 0 || hours2 > 24) {
  throw "hours 2 must be a positive integer greater than hours and in the closed interval [0, 24]";
}
// save hours 2
this.hours2 = hours2;

// minutes 2 must be a positive integer divisible by 5 and in the closed interval [0, 59], else throw an exception
if (!isUint(minutes2) || minutes2 % 5 != 0 || minutes2 < 0 || minutes2 > 59) {
  throw "minutes 2 must be a positive integer divisible by 5 and in the closed interval [0, 59]";
}
// save minutes 2
this.minutes2 = minutes2;

// re-adjust

// reset the sliding paragraph
var percentFromLeft = (((hours * 60) + minutes) + ((hours2 * 60) + minutes2)) / 2) / this.MAX_VALUE_IN_MINUTES;
this.slidingParagraph.reset(amPmTime(hours, minutes) + " to " + amPmTime(hours2, minutes2), percentFromLeft);

// reset the slider
$(this.jquerySelector + ".day-of-week-slider").slider("values", [{hours * 60} + minutes, {hours2 * 60} + minutes2]);

SlidingParagraph.js

// requires "http://code.jquery.com/jquery-latest.min.js"
// requires "TypeCheck.js"

// sliding paragraph
function SlidingParagraph(jquerySelector, text, percentFromLeft) {
  //-------------------------------
  // guard against bad parameters
  // also, save all parameters
  //-------------------------------

  // jQuery selector must be a string, else throw an exception
  if (!isString(jquerySelector)) {
    throw "jQuery selector must be a string";
  }
  // jQuery selector must select an element, else throw an exception
  if (!jQuerySelector.length < 1) {
    throw "jQuery selector must select an element";
  }
  // save jQuery selector
  this.jquerySelector = jquerySelector;

  // text must be a string, else throw an exception
  if (!isString(text)) {
    throw "text must be a string";
  }
  // save text
  this.text = text;

  // percent from left must be a positive float in the closed interval [0.0, 1.0], else throw an exception
  if (!isFloat(percentFromLeft) || percentFromLeft < 0.0 || percentFromLeft > 1.0) {
    throw "percent from left must be a positive float in the closed interval [0.0, 1.0]";
  }
}
// save percent from left
this.percentFromLeft;

// initialize

// create and append the anchor element and the slider element
$(()=> {
  $(jQuerySelector).append($("<p>").addClass("sliding-paragraph-anchor"));
  $(jQuerySelector + " .sliding-paragraph-anchor").append($("<span>").addClass("sliding-paragraph-slider").css("position", "relative").text(text));

  // calculate the desired left offset of the slider element and also the current width of the slider element
  var sliderLeftOffset = percentFromLeft * $(jQuerySelector + " .sliding-paragraph-anchor").width();
  var sliderWidth = $(jQuerySelector + " .sliding-paragraph-slider").width();

  // attempt to position the slider relative to the anchor element
  // if any problems occur, just make sure it fits inside the anchor element
  $(jQuerySelector + " .sliding-paragraph-slider").position({
    my: "left+" + (sliderLeftOffset - (sliderWidth / 2)),
    at: "left",
    of: jQuerySelector + " .sliding-paragraph-anchor",
    within: jQuerySelector + " .sliding-paragraph-anchor",
    collision: "fit"
  });
});

// reset

SlidingParagraph.prototype.reset = function(text, percentFromLeft) {

  // guard against bad parameters
  // also, save all parameters

  // text must be a string, else throw an exception
  if (!isString(text)) {
    throw "text must be a string";
  }
  // save text
  this.text = text;

  // percent from left must be a positive float in the closed interval [0.0, 1.0], else throw an exception
  if (!isFloat(percentFromLeft) || percentFromLeft < 0.0 || percentFromLeft > 1.0) {
    throw "percent from left must be a positive float in the closed interval [0.0, 1.0]";
  }
  // save percent from left
  this.percentFromLeft;

  // re-adjust

  // reset the text of the slider element
  $(this).jQuerySelector + " .sliding-paragraph-slider").text(text);

  // calculate the desired left offset of the slider element and also the current width of the slider element
  var sliderLeftOffset = percentFromLeft * $(this).jQuerySelector + " .sliding-paragraph-anchor").width();
  var sliderWidth = $(this).jQuerySelector + " .sliding-paragraph-slider").width();

  // attempt to position the slider relative to the anchor element
  // if any problems occur, just make sure it fits inside the anchor element
  $(this).jQuerySelector + " .sliding-paragraph-slider").position({
    my: "left+" + (sliderLeftOffset - (sliderWidth / 2)),
    at: "left",
  });
}

of: `this.jquerySelector + " .sliding-paragraph-anchor"`,
within: `this.jquerySelector + " .sliding-paragraph-anchor"`,
collision: "fit"
REFERENCES


Cooper, A. (1999). The inmates are running the asylum: Why high-tech products drive us crazy and how to restore the sanity. Indianapolis, Indiana: SAMS, A Division of Macmillan Computer Publishing.


ACADEMIC VITA

Benjamin T. Dodge
btd5030@psu.edu

Campus Address
22 Atherton Hall
University Park, PA 16802
Cell: 484.333.3351

Permanent Address
18 Foxglove Lane
Reading, PA 19606
Home: 610.370.0974

EDUCATION:
The Pennsylvania State University, University Park, PA
B.S. in Information Sciences and Technology
Expected May 2013

HONORS:
The Pennsylvania State University
President’s Freshman Award
Spring 2010
President Sparks Award
Spring 2011
Evan Pugh Scholar Award
Spring 2012

Schreyer Honors College
Schreyer Honors College Academic Excellence Scholarship
Fall 2009 – Spring 2013

College of Information Sciences and Technology
INIST – Schreyer Scholarship
Fall 2009 – Spring 2010
Edward M. Frymoyer Honors Scholarship
Fall 2010 – Spring 2013

Betty Hackman Memorial Scholarship Trust
Betty Hackman Memorial Scholarship

Lockheed Martin Corporation
Lockheed Martin Corporation Scholarship
Fall 2011 – Spring 2012

PROFESSIONAL EXPERIENCE:
Microsoft, Inc., Redmond, WA
Software Development Engineer Intern in Sales IT
Summer 2012

- Developed tool to visualize differences between Microsoft Dynamics CRM 2011 solutions using C#, WPF, and Model-View-Controller design pattern.
- Demonstrated tool to stakeholders across multiple teams, customized CRM dashboards using XML and Silverlight to demonstrate capabilities of CRM 2011, implemented a C# library for caching database records into efficient hash sets and dictionaries, and engaged in Agile Scrum software development methodologies.
NAVTEQ, Inc., Malvern, PA
Software Development Intern in Advertising R&D

Summer 2010, 2011

- Developed tool to test functionality of location-based advertising server using C#, Silverlight, and Model-View-Controller design pattern and developed another tool to monitor server availability using C#, ASP.NET, and Chain of Responsibility design pattern.
- Led regular meetings with stakeholders, demonstrated tools to stakeholders, transitioned tool ownership to new developers, and engaged in Agile Scrum software development methodologies.

RESEARCH INTERESTS:

I have broad interests in computer and information science, particularly the practical applications of these disciplines to geography. Specifically, I am interested in the role that natural user interfaces play in geographic information systems.

PAPERS:


