An Illustrated Primer

by

Kendra Leigh Pugh

B.S., Massachusetts Institute of Technology (2009)

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of
Master of Engineering in Electrical Engineering and Computer Science
at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2012

© Massachusetts Institute of Technology 2012. All rights reserved.

Author

Department of Electrical Engineering and Computer Science
August 7th, 2012

Certified by

Dennis M. Freeman
Professor of Electrical Engineering
Thesis Supervisor

Accepted by

Dennis M. Freeman
Chairman, Masters of Engineering Thesis Committee
An Illustrated Primer

by

Kendra Leigh Pugh

Submitted to the Department of Electrical Engineering and Computer Science on August 7th, 2012, in partial fulfillment of the requirements for the degree of Master of Engineering in Electrical Engineering and Computer Science

Abstract

Educational technology has received unprecedented attention recently, as the efficacy of traditional education methods are coming into question. This thesis introduces the Primer - an e-textbook that initially customizes the presentation of a given course's content to the preferences of the reader, then gradually scales those customizations back as the reader progresses through the e-textbook. The Primer is designed to assist in the “two sigma” problem [1] of bringing the educational experience of a student closer to that of receiving individual instruction from a competent tutor that uses mastery learning techniques.

The Primer initially improves the accessibility of a new course to the student by customizing the representation of the course’s content to the student’s needs; additional instructional videos, dereferencing of definitions, and hyperlinks back to the defining information originally introducing a concept are provided given the students’ preferences. These customizations are based both on the strengths and weaknesses in the student’s background in the course’s prerequisites, and the information presentation styles preferred by the student. The Primer reduces these customizations over the course of the textbook so that the e-textbook gradually and eventually reflects the delivery style intended by the e-textbook’s author, and to give consistency across users’ e-textbook experience. The goal is a method of information delivery that is personalized, yet standardized.

The Primer allows students to ease into an e-textbook without becoming discouraged with the presentation style native to the e-textbook’s field. It can prevent the student from becoming discouraged by providing more support for areas of the e-textbook’s content in which the student has unstable grounding. The Primer was implemented using readings from 6.01 and additional content from OCW Scholar, and feedback was received from students as well as professionals in the field of educational technology.

Thesis Supervisor: Dennis M. Freeman
Title: Professor of Electrical Engineering
Acknowledgments

I’d like to thank Dennis Freeman; primarily for guidance and support, ultimately for a large amount of patience, and firstly for allowing me the opportunity to work on the Primer.

I’d like to thank my parents, who simultaneously insisted on constraint satisfaction to meet the qualifications of a Real Person™, and insisted that I could do whatever I wanted. I’m not sure I’m quite there yet with either, but I wouldn’t have thought it possible without your persistence.

Thank you to my friends, who put up with me gracefully. In particular, to Caitlin Johnson, Sarina Canelake, Adam Hartz, and Alejandro Sedeno; Matt Malchano, Clara Rhee, Danielle Magrogan, Coral Ash, Andrew Rowe, John Innes. This wouldn’t have been possible without you.

Thank you to Sal Khan, who helped me understand 18.06, and whose innovation legitimized my interest in “TA stuff”
## Contents

1 Introduction .......................... 11
   1.1 Educational Technology .............. 11
   1.2 The Primer .......................... 12
   1.3 Evaluation .......................... 12
   1.4 Contents ........................... 13

2 Previous Work .......................... 15
   2.1 Machines and the Study of Education ........................................................................ 15
   2.2 WWW and Online Education .............. 17
   2.3 Electronic and Interactive Textbooks ................................................................. 18

3 The Primer ............................... 21
   3.1 User Profiling .......................... 22
   3.2 Approaching Standardization .............. 22
   3.3 Disclaimer ............................... 22

4 Primer Interface .......................... 25
   4.1 Admin ................................. 25
      4.1.1 User Profile ......................... 25
      4.1.2 Chapters ........................... 26
      4.1.3 Definition Elements ................. 27
   4.2 User ................................. 28
      4.2.1 Login and Logout ..................... 28
# List of Figures

4-1  Administration Home ......................................... 26  
4-2  Admin User Profile Page ..................................... 27  
4-3  Example User Profile ......................................... 28  
4-4  Chapter Admin Page ........................................... 29  
4-5  Detailed view of Chapter Admin Page ....................... 30  
4-6  Definition Elements Admin Page ............................. 31  
4-7  Detailed view of Definition Element Admin Page ........... 32  
4-8  Primer Masthead ................................................ 32  
4-9  Primer Login Page ............................................. 33  
4-10 Primer User Profile ........................................... 33  
4-11 Primer Excerpt ................................................ 34  
4-12 A Triggered Definition Element .............................. 34  
4-13 Primer Chapters ............................................... 34  
4-14 Primer Definition Elements ................................. 35  
4-15 Primer Definition Element .................................... 36  

5-1  Reference to the ‘voltage’ Definition Element ............... 43  
5-2  The Primer Template Hierarchy ............................... 44
Chapter 1

Introduction

This thesis explores the current state of educational technology, and attempts to address an outstanding problem in educational technology with a new combination of solutions. The outstanding problem in educational technology this thesis attempts to address is known as the two sigma problem [1]. The new combination of solutions is an e-textbook that initially adapts its presentation to the student’s preferences, then gradually scales back the customizations over the course of the e-textbook such that by the end of the student’s experience, the e-textbook is a more uniform resource across students. This customized content delivery system will be referred to throughout as The Primer.

1.1 Educational Technology

The Primer is introduced during an exciting time for the field of educational technology. Unprecedented interest in educational reform and technology-enabled learning sets the stage for a multitude of new approaches to a student’s education, both in the classroom and at home [10] [6]. Simultaneously, the size of a given class is no longer necessarily limited by the size of a room, the number of desks or even the number of instructors available; web publications [7] and Massive Open Online Courses (MOOCs) [3] expand educational offerings to anyone willing to learn. The proliferation of tablets and popular adoption of e-books has lead to a race for domi-
nation among various e-textbook publishing formats. The Primer is introduced at a
time where adaptation of its ideas could be beneficial to a great number of learners
worldwide.

1.2 The Primer

In the midst of all this innovation, the Primer is an attempt to carefully improve a
classic aspect of the higher education experience, the textbook, without sacrificing
any of its inherent benefits. An e-textbook device has the benefit of being lighter and
easier to carry than a traditional textbook, and an e-textbook can theoretically be
transferred to a new device when normal wear and tear affects the previous device.
A textbook with presentation style personalized to the reader can improve the rate
and quality of information transfer between the textbook and the reader.

But what of standardization? What if every economics student read a slightly
different version of *The Wealth of Nations* [9]? The downside of MOOCs is that
professional oversight of a student’s experience may be diluted to the extent that it
is not guaranteed to be sufficiently rigorous or comparable to that of other students.

The Primer attempts to address all these issues. Too much customization and an
e-textbook experience isn’t relatable to any other student in the course. Too little and
the experience provides little advantage over a traditional textbook. The Primer aims
to provide improvement over the traditional textbook experience without sacrificing
the benefit that comes from a common source of information.

1.3 Evaluation

The Primer was evaluated by several experts both familiar with the course content
upon which the Primer was based and expressing interest in the field of educational
technology. These experts are participants in educational technology efforts such as
OCW, the iCampus project, and edX. The experts were asked to read a sample of
the Primer under several different profiles, create a dynamic element for use within
the Primer, edit the Primer to include this dynamic element, and give feedback on the experience.

1.4 Contents

Chapter 2 discusses both the historical and current related work relevant to the Primer.

Chapter 3 is an overview of the goals of the Primer and the specifics of which problems it does and does not solve.

Chapter 4 provides a detailed description of the Primer's user interface. It establishes both the educator and student's interface configurations, and indicates the decisions that went into establishing the user interface as provided.

Chapter 5 presents the architecture of the Primer.

Chapter 6 reviews the evaluations of the Primer.

Chapter 7 concludes the thesis and indicates desirable possible extensions to the Primer.
Chapter 2

Previous Work

Educational technology has a rich history; traditional technologies include those items classically associated with a traditional instructional setting, such as blackboards, chalk, textbooks, and lecture halls. Although these elements aren’t typically indicated by the contemporary use of the term ‘Educational Technology’, they are the standard against which emerging technologies are measured. The past century, and in particular the past ten years, have seen incredible advances and a myriad of options proposed to improve upon these traditional technologies. The Primer is comparable to or influenced by these advances.

2.1 Machines and the Study of Education

The first significant advancements in educational technology in the 20th century were the introduction of training films and teaching machines. Thomas Edison’s 1908 film *Flypaper* represents the first educational film in the United States, and Sidney Pressey’s teaching machine represents the first in a long effort towards an automated tutor [8]. The Primer represents an element of educational technology that derives both from educational film and machines that respond to student input; it would be impossible without these two pieces of groundbreaking technology. Edison was inspired by the blossoming industry of film; ‘Pressey by the goal of providing teachers with more individual time with students. Both goals are relevant to the Primer; it
intends to make use of the ubiquity of recording technologies and the ability of anyone
with a computer, a webcam and the Internet to make an instructional video, and to
improve personal learning experience so that more time with instructors can be spent
refining understanding and clarifying those difficulties a student encounters that can’t
be approached by automated means.

Training film use boomed during World War II as soldiers needed basic instruction
before entering the field. This legitimized the presence of video in the instructional
setting. Simultaneously, the advances in behavior psychology by Skinner and others
legitimized the quantitative study of human experience in relation to learning. These
historical developments are relevant to the Primer in that the relevance of video to
the learner was not solidified until World War II, and that a quantitative approach
to measuring and changing students’ experiences, rather than exclusively test scores
to measure performance, had not been legitimized. These advances solidified those
concepts.

The Primer makes heavy use of hyperlinks and hypermedia. Precursing the use
of hypertext and hypermedia, Vannevar Bush wrote of the utility of a device that
tracked the history of a person through research materials, called a memex [2], in
1945. Much later, Ted Nelson wrote and researched hypertext, hypermedia, and
transclusion in an effort to manifest a system similar to the World Wide Web.

Though the behavioral psychology movement focused on human learning, little
effort to relate the results of such experiments to the system of education was re-
cognized before the ideas promoted by Edward Bloom. Bloom’s famous “two sigma”
problem highlights the basics of the disparity between individual professional tutor-
ing and the experience inside a typical classroom [1]. The Primer attempts to bridge
the two sigma gap by personalizing the presentation of textbook information to the
student. This personalization is expected to improve retention of new information
and automatically reinforce information with which the student has difficulty.
2.2 WWW and Online Education

The World Wide Web contributes greatly to the idea of the Primer; the Primer is designed to be accessed from a web server, and viewed on a personal computer, laptop, mobile device, or tablet device. The advances in educational technology following the proliferation of the World Wide Web provide a backdrop against which the Primer can be evaluated.

One of the most ambitious efforts to introduce educational technologies that take advantage of the web was the Microsoft iCampus Project [4]. Innovation such as Technology Enhanced Active Learning and online tutors used in MIT courses came out of the iCampus project. These technologies center around human-computer interaction, and much of the coursework associated with the technologies could be completed at any time from any computer with a connection to the internet. The iCampus project represents a step in the direction towards technologies such as the Primer.

Another step in the right direction is MIT’s OpenCourseWare (OCW) [7]. OCW is an archive of materials used during previous terms of MIT courses for instruction and learning. The Primer’s content is derived in part from OCW offerings; the OCW offerings in turn were developed in response to student input on information not readily absorbed through a traditional textbook. In addition, the Primer would be an excellent resource for OCW Scholar. OCW Scholar are traditional OCW offerings augmented with additional content and designed for independent learners to use in absence of additional resources for learning, such as an instructor or course staff.

Similar to OCW scholar is The Khan Academy [6]. Started by Salman Khan as an effort to educate his cousins over the internet, Khan Academy features over 3,200 instructional videos as well as exercises designed to teach anyone with an internet connection the basics in a wide variety of disciplines. The success of Khan Academy has led to experimentation in California and Colorado K-12 schools with the concept of an “Inverted” or “flipped” classroom. In such a classroom, students watch videos and absorb instructional materials at home, then practice skills with one another and under the instructional guidance of teachers or tutors. The Primer is an example of
an educational technology that is both influenced by Khan Academy - educational videos are available and practice sessions are tailored to the student - and would augment the Khan Academy experience by providing a textbook.

Yet another emerging educational technology that makes use of the web is the Massive Open Online Course, or MOOC. Similarly to sites such as OCW and The Khan Academy, the Primer is an educational technology that could see exemplary use in a MOOC. MOOCs are different from OCW in that they are designed to replace the classroom experience, not archive it; a greater variety of materials are available, typically including contact with an instructor for the course. MOOCs are a relatively new concept; some MOOCs under development include MIT and Harvard’s edX, Udacity, and Stanford’s Coursera.

2.3 Electronic and Interactive Textbooks

Though previously mentioned technologies have contributed to the emergence of The Primer, The Primer is more related to electronic textbooks than any previously discussed technology. Electronic textbooks were originally developed in the 1960’s by Doug Engelbart at Stanford Research Institute. The advent of the web allowed massive distribution of electronic textbooks, and the development of electronic textbook readers made electronic textbooks a viable alternative to paper media.

Today, competition for the electronic textbook market is still strong, and several formats for electronic textbooks exist, including SCORM, HTML5, ePub and KF8. The different formats are a result of different companies developing for the electronic book market at the same time, as well as an independent effort to come up with an open standard. The Primer is published in HTML5 to ensure the greatest compatibility with the largest number of devices. Closest in technological achievement to The Primer is Nature Publishing’s Principles of Biology, an interactive, modular textbook that is customizable by the instructor for a course. Additional companies are also introducing interactive electronic textbooks [5]. The difference between the Primer and these technologies is that the Primer’s presentation is customizable at the user
level, by the user. In this way, the Primer represents a new direction for e-textbooks.
Chapter 3

The Primer

The Primer is an electronic textbook that customizes the presentation of information to the stylistic preferences and educational strengths and weaknesses of the student reader. The Primer is designed to do this in attempt to reduce the “two sigma” gap - the performance difference between a student individually tutored using mastery techniques and a student receiving traditional instruction. In some ways, the Primer represents an inversion of Vannevar Bush’s Memex [2] - instead of recording the trail a student takes through a particular set of standard resources, the Primer tailors one of many paths through a set of standard resources for the student and provides the student with that path. The standard set of resources is always available.

The change the Primer induces is the number and depth of exposures to definitions and assistive videos the student automatically receives. The student, as well as any administrator, can change the student’s preferences at any time - if a student’s proficiency in a particular area increases, or if their preference for video lectures decreases, those changes can be reflected by changing their corresponding setting in the Primer.

The Primer is available anywhere there is an internet connection and a web browser. The Primer can be viewed on a personal computer or a mobile or tablet device. The profile a student develops with the Primer persists between a student’s sessions with the Primer.
3.1 User Profiling

The Primer engages in user profiling through explicit references to data stored in a user profile. User profiles keep track of student proficiency in a number of sub-fields relevant to the textbook’s domain, as well as general student preferences for definition availability.

3.2 Approaching Standardization

The Primer gradually scales back the customizations it makes by dividing a textbook into chapters. Each chapter can have a different level of allowed customizations and the level of customization can be adjusted across users. The Primer’s customizations can be adjusted by chapter; definitions and videos will be less or more likely to dereference on the page based on these adjustments. Since adjustments to chapters affect all users, the Primer can be standardized by manipulating chapter customizations. In particular, customization by chapter allows later chapters to have fewer dereferences, making those chapters more closely resemble the baseline layout of the textbook. In this way, a more uniform textbook can slowly emerge over the administration of a course, such that by the end of a course, all students’ Primers look the same.

3.3 Disclaimer

The Primer is not an electronic tutor; it does not record students’ interactions in an effort to establish their skill levels. The Primer simply adjusts to the decisions for information delivery that students make. The Primer is not a complete learning system. Learning management systems also handle evaluation and administrative tasks.

The Primer is not a Learning Content Management System (LCMS). An LCMS typically involves the ability to create and manipulate entire learning courses; the Primer is just an e-textbook. One similarity between LCMSes and the Primer is that the Primer’s definition elements could be construed as learning objects. The Primer
adjusts the learning path of a student to the students' preferences, but the Primer stands-in for a textbook in a course, not all course materials.

The Primer is different from existing e-textbook solutions in the following ways: although electronic textbooks are in existence today, none attempt to adjust presentation of information in the same ways and at the same level as The Primer. E-textbooks that customize their presentation to the reader exist, but these customizations are determined on the course level; all textbooks for students in a particular course look the same. The Khan Academy website most closely approximates the pursuit of experiences that differentiate The Primer from another e-textbook; however, Khan Academy does not publish e-textbooks.
Chapter 4

Primer Interface

The front end of the Primer is the most important aspect of its functionality; customized user experience means very little without discussion of user interface. This section walks through the user interface functionality of the Primer.

4.1 Admin

The Primer has a few essential administrative components. Administrative components in the Primer can be used to add, change, or delete users; add, edit, or delete a user’s profile preferences; add, change or delete chapter difficulty coefficients; and add, change, or delete definition elements. Additional changes to the Primer, such as changing chapter content, must be done by editing Django templates, discussed in Chapter 5. The Primer’s admin interface makes use of Django’s automatic admin interface, django.contrib.admin. The main administration page is shown in in figure 4-1.

4.1.1 User Profile

The user profiles pages allows administrators to add or make changes to a user’s user profile. The main page to access user profiles is shown in figure 4-2. An example of a user profile is shown in figure 4-3. Note that user profiles contain five values; four of
the values indicate a user's self-described proficiency in one of four areas that are the focus of 6.01, the course upon which this implementation of the Primer is based. The fifth value indicates a user's general preferences regarding definition dereferencing.

4.1.2 Chapters

The chapters page allows administrators to add, delete, and make changes to the difficulty coefficient associated with each chapter. The main page to access chapters is show in figure 4-4. An example of a chapter admin page is shown in figure 4-5. Chapter coefficients are used to modify the overall levels of definition dereferencing that happens in a given chapter; the lower the value, the less likely it is that a definition will be dereferenced. This enables the standardization of the presentation of the Primer as users progress to later chapters.
4.1.3 Definition Elements

The definition elements page allows administrators to add, edit, and delete definition elements. The main page to access definition elements is shown in figure 4-6, and an example definition element is shown in figure 4-7. Definition elements are the definitions and videos whose presentations are customized based on user preferences. In this implementation of the Primer, they are vocabulary word definitions and videos that are supplementary to the main e-textbook. Each definition element has a difficulty rating for each of the four main areas of focus in the e-textbook course. In addition, each definition element has an overall difficulty rating associated with definition elements. If these difficulty ratings, when multiplied by a chapter definition coefficient, exceed the ratings specified by the user in their preferences, then the element in the ‘triggered’ field follows the element in the ‘baseline’ field in-line within a chapter of the Primer. If not, then the ‘baseline’ field appears as a hyperlink to a page featuring both the ‘baseline’ and ‘triggered’ fields. This functionality is detailed and figures are available in the ‘Definitions’ section below.
4.2 User

The Primer’s user components are those aspects of the Primer meant for student interaction. The chapters of the Primer are the main focus of user interaction, but additional pages are required to make the Primer most useful. User pages feature a masthead with links to the login, profile, chapter, and definition aspects of the Primer, shown in figure 4-8. The user pages are described below.

4.2.1 Login and Logout

The Primer’s login page is shown in figure 4-9. Users log into the Primer in order to edit their profile preferences and allow the chapters of the Primer to adjust to their preferences. The login page will also display one of several error messages associated with logging in as appropriate. If a user is already logged in, the user’s name will appear in the error message and a link to the logout page will appear. If the user
login attempted is invalid or inactive, a message to that effect will also appear. If the user selects logout, a short logout message appears along with a link to log in again.

### 4.2.2 Profile

The user’s profile page is shown in figure 4-10. Users can view and edit their profile from the user profile page. The profile page allows a user to control the way the Primer appears based on their personal preferences; if their proficiency ratings exceed the difficulty rating of a given Primer definition element in a given chapter multiplied by the chapter’s standardization coefficient, than the baseline version of the definition element will appear. This calculation is performed for all areas of proficiency in the course book as well as for a general definition difficulty. If any of the difficulty ratings exceed the proficiency level of a user, than the triggered version of the definition element will be displayed. Users can update their profile preferences and the change
will immediately appear upon completion and submission of the HTML form.

4.2.3 Chapters

The chapter pages are the heart of the Primer. These pages display the basic e-textbook content as well as the definition elements in their appropriate state given the definition element and the logged-in user’s preferences. An example of the Primer is shown in figure 4-11. Note that images in the Primer are static and appear regardless of user preferences; the Primer could be easily adapted to make this a user preference, however. Figure 4-11 shows a definition element that has not been triggered; the hyperlink will direct the user to a definition element page. Figure 4-12 shows the same definition element triggered; the hyperlink is no longer available, and the definition appears immediately after the baseline appearance of the definition element.

Links to all chapters are available on a separate chapters page, shown in figure 4-13.
4.2.4 Definitions

The definition elements are what make the Primer different from a regular e-textbook. A list of all definition elements is available at the definitions webpage, shown in figure 4-14. Each definition element has its own page with both the baseline name of the definition and the triggered content of the definition displayed. An example of this page is shown in figure 4-15. Definition elements are also displayed within the chapter pages, and their appearance changes based on the settings for the definition element, the chapter, and the user coming together to create the particular definition reference.
### Change definition element

<table>
<thead>
<tr>
<th>Name:</th>
<th>state_machine</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chapter:</td>
<td>Chapter 1, Chapter 2, Chapter 3, Chapter 4</td>
</tr>
</tbody>
</table>

Hold down "Control", or "Command" on a Mac, to select more than one.

<table>
<thead>
<tr>
<th>ProgrammingThreshold</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>SystemsThreshold</td>
<td>0</td>
</tr>
<tr>
<td>CircuitsThreshold</td>
<td>0</td>
</tr>
<tr>
<td>ProbabilityThreshold</td>
<td>0</td>
</tr>
<tr>
<td>Baseline:</td>
<td>state machine</td>
</tr>
<tr>
<td>Triggered:</td>
<td>a method of modeling systems whose ou</td>
</tr>
<tr>
<td>DefinitionThreshold</td>
<td>2</td>
</tr>
</tbody>
</table>

Figure 4-7: Detailed view of Definition Element Admin Page

**Login Profile Chapters Definitions**

Figure 4-8: Primer Masthead
Login Page

kpugh is already logged in logout first chapter

Username: 
Password: 
Login

Welcome to the Primer

You are logged in as kpugh

Edit your profile below. 0 is a novice, 10 is an expert.
Definition Rating: 1
Programming Background: 1
Systems Background: 1
Circuits Background: 1
Probability Background: 1
Submit

logout here

here is the first chapter

Figure 4-9: Primer Login Page

Figure 4-10: Primer User Profile
The laws for the flow of electrical current are similar to those for the flow of an incompressible fluid. The net flow of electrical current into a node must be zero.

The following circuit has three elements, each represented with a box.

![Diagram](image)

Figure 4-11: Primer Excerpt

The laws for the flow of electrical current (the flow of electric charge through a path in a circuit) are similar to those for the flow of an incompressible fluid. The net flow of electrical current into a node must be zero.

The following circuit has three elements, each represented with a box.

![Diagram](image)

Figure 4-12: A Triggered Definition Element

### Login Profile Chapters Definitions

- Chapter 1
- Chapter 2
- Chapter 3
- Chapter 4
- Chapter 5
- Chapter 6
- Chapter 7
- Chapter 8

Figure 4-13: Primer Chapters
Login Profile Chapters Definitions

- state machine
- belief state
- State Estimation
- PCAP
- State Estimation Video
- interpreter
- expression
- variable
- mutation
- aliasing
- environment
- object-oriented programming
- class
- instance
- method
- inheritance
- recursion
- list comprehension
- lambda
- self
- state transition diagram
- transition table
- controller
- plant
- signal

Figure 4-14: Primer Definition Elements
mutation

the action of changing the values associated with a particular data structure without also overwriting that data structure. In particular, when a variable is mutated the variable name points to the changed structure.

Back to Chapter 2
Back to Chapter 3

Figure 4-15: Primer Definition Element
Chapter 5

Primer Architecture

5.1 Content

The innovative aspect of the Primer is its customizable interface, but in order to demonstrate the Primer, a suitable textbook had to be adapted to the Primer platform. The Primer is most useful when content suitable for definition elements are also incorporated; definitions and videos make the Primer different from a standard e-textbook. The Primer was made from a collection of source materials that I had access to; the 6.01 course readings and 6.01 OCW Scholar offerings.

5.1.1 Course Readings

I used the course readings for the MIT class 6.01: Introduction to EECS I from spring term 2012. This textbook provided the optimum platform to demonstrate the power of the Primer; the stylistic aspects of the authors in the 6.01 course readings are preserved, yet the definitions and videos designed to present repetition of information and information in a different way are readily accessible. The original \LaTeX files for the 6.01 course readings were carefully adapted to HTML5, including transformation of PDF figures into .jpg images. As a consequence of the adaptation, some small features of the textbook changed; figure references were changed from the standard \LaTeX method to simply “above” or “below” as the image reference appeared in the
source document. In addition, exercises in the 6.01 readings were visually distinct from regular text; this distinction was removed for ease of conversion to HTML5 and to ensure that the Primer rendered correctly on more browsers. These distinctions could be added back in using CSS styling.

5.1.2 Definitions

Once the base of the textbook was established, material for the definition elements in the Primer had to be established as well. While working for 6.01 OCW Scholar, I developed “Weekly Summaries” of 6.01 course content. The purpose of OCW Scholar over a traditional OCW offering is to increase the amount of support material available in order to enable independent learners. These summaries were based on my experiences as a teaching assistant in 6.01 during spring semester 2011; students had difficulty absorbing course material as presented, and these summaries attempted to ameliorate those difficulties.

In particular, the “Vocabulary” section of the summaries provide those terms a student would have great difficulty passing the course without understanding. These terms were used as the basis for definition elements in the Primer. These terms are exactly the kind of information that would be useful as definition elements - they are chosen based on student feedback on what information is essential to the course, but possibly missed on the individual efforts of a student through the course material. Definitions for the vocabulary words were taken from the 6.01 course readings.

5.1.3 Videos

In addition to weekly summaries, I developed a number of short video presentations for the 6.01 OCW Scholar website. These videos were also based on student feedback as to the more challenging or unintuitive aspects of the course. The videos were also organized to cover a short section of the 6.01 material at a time, breaking the material into conceptual chunks that are also meant to be viewed about once per week. The videos were distributed throughout the 6.01 readings in a manner that made sense
given the context of the chapters.

5.2 Django

The Primer was developed using the Django development framework. Django is an open source Web 2.0 application framework, written in Python, which follows the model-view-controller architectural pattern. This pattern is established through the use of files; models, views, and controllers all live in separate files. Additional files manage application settings. A single Django installation can run multiple websites, each with their own content and applications. This means that multiple Primers can be established using a single Django installation.

The "model" file, known as models.py in a Django application, allows a programmer to set up database tables automatically by writing Python classes. These classes are then accessible through a Python database-abstraction API that can be used to create, retrieve, update and delete objects.

The type of database can be specified in a separate settings file. Django has built-in support for PostgreSQL, MYSQL, SQLite, and Oracle. Django will also accept other database engines. I used SQLite for developing this application, but did not have to manipulate the database tables directly, as the Django database abstraction API made it easy to manage information in the database through Python.

The "controller" aspect of the model-view-controller architectural pattern is established, somewhat confusingly, by a file called views.py. views.py controls the use of database elements in webpages, and processes incoming HTML requests to create or change the model data.

The "view" aspect of the model-view-controller architectural pattern is implemented using Django templates. Django templates are HTML files containing a Django-specific template language that allows hierarchical organization and extensibility of different portions of HTML. Templates receive database and other information as passed on from views.py and renders it in HTML, completing the MVC architecture.
Two additional files complete the essentials in a Django website application. Each Django project has a settings.py file, which specifies a variety of settings for a given Django project. Settings include which database engine and database to use, the location of static and media files, which applications and middleware are included, logging configurations, and the location of the URL configuration.

The final file used in a given Django application is urls.py, the URL configuration itself. URLs are matched through regular expressions to specific controller logic in views.py, creating the final connection between website user and application.

The specific data models, view-rendering functions, and html templates used together describe the basic functionality of the Primer, and the specifics of three components are described in further detail in the following sections of this chapter. The code for the Django application is included in the appendicies.

5.2.1 Models

The Primer uses several model classes to store database information associated with its administration. Users, definition elements, and chapters all require database storage and transactions in order for the Primer to function.

The UserProfile model class is the model class used to represent a given user’s profile. A user’s profile contains his or her preferences regarding the appearance of the Primer. In particular, the UserProfile model class contains proficiency ratings in the four main areas of 6.01, as well as a general rating of proficiency with definitions. These values are used in calculations of the appearance of a particular Primer definition object in the chapters view.

Since the Primer uses Django’s default settings for administration of users, user profiles must be initiated separately from users. For users and groups, the Primer uses Django’s django.contrib.auth.models classes.

Since Django includes a Python database-abstraction API, both users and user profiles could be initiated in batches using a short Python script. Scaling the Primer up to deal with an MIT class’s worth of users is as simple as providing a file of usernames and passwords.
The Chapter model class represents a given Chapter of the Primer. Chapter models are necessary to retain the coefficient associated with the advanced nature of a given chapter; these coefficients are used to determine whether a given Primer chapter will have more or fewer dereferenced definitions.

The PrimerElement model class represents a Primer element, a learning object whose appearance changes based on the comparison of a given user's preferences with the PrimerElement's ratings. This class stores the ratings associated with a given element, which are multiplied by the chapter's advancement coefficient to determine the overall difficulty associated with a given PrimerElement. These ratings are compared to the preferences specified by a given user's UserProfile attributes when deciding whether to dereference the definition of a Primer definition object in the text of the Primer.

The DefinitionElement model class is a subclass of the PrimerElement model class. PrimerElement model objects have difficulty ratings in the four main areas of 6.01; DefinitionElements have an additional rating for general definition difficulty. DefinitionElements are subclassed to allow for extensibility to the Primer; additional UserProfile preferences could be added for comparison with different subclasses of PrimerElement. Different logic could be used to determine the rendering of different subclasses.

### 5.2.2 Views

Views are where the control logic for the Primer resides. Views query the database and receive user input, then perform any specified operations on that information, then pass the results on to templates for rendering.

The definitions view controls the display of individual definition elements on the Primer website. It queries the database for the term and definition or video associated with that term and sends them to a generic template that is used to display any individual definition.

The chapter view handles the majority of logic that goes into determining the custom appearance of the Primer. For a given chapter, a query of the definition
elements is done to determine which definition elements are relevant to that chapter. Then, for each definition element, a comparison is done of that definition element’s thresholds to the user’s background as specified by the user’s profile. The definition element’s thresholds are multiplied by a coefficient determined by the chapter; this is where the standardization through gradual regression to hidden definitions comes in. If the threshold exceeds the background, than the definition element is dereferenced and the definition or video will appear alongside the definition element’s baseline name in the text of the Primer website. If not, a hyperlink to the definition or video will appear in the text of the definition’s baseline name.

The `site.login` view handles users logging in. If a user attempts to log in while another user is logged in, or with an invalid or expired username, he will be brought back to the login screen with an error message.

The `logging.in` view handles the requests associated with user login. It queries the database with the given username and password, checks to see if the username is valid, checks to see if the user is active, and then logs the user in. Django uses the PBKDF2 algorithm with a SHA256 hash for passwords.

The `site.logout` view handles site logouts. If a user is logged in, it logs the current user out and provides a link back to the login page.

The `profile` view retrieves the user that is logged in’s profile values for display in a template. The difficulty rating preferences for the four main areas of 6.01 are retrieved, as well as the general definition difficulty rating.

The `submit.profile` handles requests to change a user profile. It receives an HTML POST request and updates the values associated with a user profile accordingly, then passes the new values to a template to render immediately.

### 5.2.3 Templates

Django’s template engine provides a powerful mini-language for defining the user-facing layer of an application. Templates are the “view” portion of the MVC architecture; they are the presentation logic. Unlike other parts of the Django development framework, templates can be maintained by anyone with an understanding of HTML;
no knowledge of Python is required. This means that the requirements for authoring a textbook using the Primer system is access to the admin pages, access to the templates, and knowledge of HTML. The Primer uses several templates in a hierarchical arrangement to determine its presentation.

Templates are the place where user interface design for the Primer occurs. Changing the Django templates associated with the Primer changes the appearance of the Primer itself. Stylistically, the Primer has been left with no CSS styling - its appearance adapts to the preferences of the browser and device and/or browser used to access the Primer.

Templates are also the place where Primer chapter content is changed. In order to add references to definition elements to the text of the Primer, a variable reference to the Primer DefinitionElement must be added to the appropriate chapter template. An example of this variable reference appears in figure 5-1; voltage is the variable.

```html
<p>Voltage is a difference in electrical potential between two different points in a circuit. We will often pick some point in a circuit and say that it is "ground" or has voltage 0. Now, every other point has a {{voltage}} defined with respect to ground. Because voltage is a relative concept, we could pick <em>any</em> point in the circuit and call it ground, and we would still get the same results.</p>
```

Figure 5-1: Reference to the ‘voltage’ Definition Element

`base.html` is the base of the Primer template hierarchy. It contains the basic layout of all templates that inherit from it. Django’s template language allows template inheritance - templates can inherit structure from other templates, and modify sections of templates called blocks in those inherited templates. The Primer’s template hierarchy is shown in figure 5-2.

`login.html` inherits from `base.html`. It determines the layout of the login page. It includes a `csrf_token`, a protection against Cross Site Request Forgeries. A CSRF attack occurs when a malicious Web site contains a link, a form button or some javascript that is intended to perform some action on a Web site, using the credentials of a logged-in user who visits the malicious site in their browser. In Django, for all incoming requests that are not using HTTP GET, HEAD, OPTIONS or TRACE, a CSRF cookie must be present. Using the `csrf_token` in `login.html` provides this
CSRF cookie. The csrf_token is used to provide secure login and must be used since login is conducted using an HTML POST request.

profile.html inherits from base.html. It simultaneously displays the user profile and provides the ability to edit it in an HTML form.

book_base.html inherits from base.html. It provides the template structure for the chapters and definition templates.

chapter_base.html inherits from book_base.html. It provides layout specific to the chapters page and individual chapters.

chapters.html inherits from chapter_base.html. It provides layout for the page used to view all chapters.

Individual chapter templates inherit from chapter_base.html. These templates control the layout of the Primer chapters. Definition elements are inserted into HTML as variables; these variables are populated by the chapter view before being passed to the template. These templates contain the 6.01 course readings, including images, adapted to HTML5.

definitions_base.html inherits from book_base.html. It provides layout specific to the definition templates.

definitions.html inherits from definitions_base.html. It controls the appearance of the page where all definitions and videos featured in the Primer are available.

definitions_global.html inherits from definitions_base.html. It specifies the

Figure 5-2: The Primer Template Hierarchy
appearance of a single definitions as viewed when selecting them from the definitions.html page or from a chapter page.

5.3 Server

The Primer is hosted at a static IP address on the MIT network; this makes the Primer accessible from any device with an internet connection. In particular, the Primer has a simple design to make it readily usable from mobile or tablet devices.

5.3.1 Apache

The site is served by Apache 2.2. Although Django delivers with its own Python development server, the amount of media demanded by an e-textbook requires a more robust HTTP server. Apache supports a variety of features, many implemented as compiled modules which extend the core functionality (mod_wsgi, discussed below, is one such module). Apache also features virtual hosting, which allows one Apache installation to serve many different websites. The Primer is hosted using a virtual host; multiple Primers with different addresses could be hosted from the same server using virtual hosting.

5.3.2 mod_wsgi

I used mod_wsgi, a popular mod for Apache, to host the Primer. mod_wsgi is an Apache mod that can host any Python application which supports the Python WSGI interface. mod_wsgi for the Primer is used in ‘daemon mode’, which means the application is being run in its own process. This reduces the impact on the normal Apache child processes used to serve up static files and host applications.
Chapter 6

Feedback

The Primer was reviewed by several professionals in the educational technology domain, both for specific feedback and to instigate a general discussion on the future of textbooks. The Primer was also reviewed by students that had and had not taken 6.01 in previous semesters to gauge the utility of the Primer in comparison to the standard 6.01 textbook. The feedback from resulting discussions is reviewed below.

6.1 Primer

Overall, the Primer is well-received. Experts approved of the basic idea of the Primer and of the implementation details associated with the 6.01 e-textbook. The admin user interface and template language was deemed intuitive enough to use to constitute an e-textbook authoring tool. Students regarded the Primer as an improvement upon the standard 6.01 textbook and expressed a desire for more courses, especially introductory courses, to have e-textbooks like the Primer.

Several comments on how to improve the Primer were received. The current implementation of the Primer provides no way to gradually decrease the rating associated with a definition element over the course of a given chapter. This means that if a user becomes familiar with a definition before the chapter ends, the only way to turn the triggered definition element off is to increase their proficiency rating in their user profile in the middle of a chapter. Associating a particular difficulty rating with a
given location of a given definition element is a good idea; the same definition element may have different difficulty ratings in different chapters due to different chapter coefficients. Associating a unique difficulty rating with every instance of a particular definition element in the Primer may constitute micromanaging; nevertheless, a granularity between every single instance and any instance within a chapter would present an improvement. A short comprehension quiz or profile dialog box to adjust profile settings within a chapter was suggested. On the other hand, some experts were happy with the persistence of definition dereferencing throughout a chapter. If in particular profile settings were adjusted automatically by some other extension, some persistence of definition dereferencing was deemed desirable.

Users suggested a way to see the difficulty ratings associated with a particular definition element. It would be beneficial to some users to understand why certain definitions were dereferencing with certain profile settings. In addition, the difficulty ratings could provide feedback to a user on his proficiency in the course relative to the difficulty ratings associated with concepts in the Primer. Users also wanted to be able to save progress through a given chapter, or the ability to use ‘bookmarks’ - this could be achieved using HTML anchorpoints. The ability to highlight or annotate chapters, and to toggle on and off highlights and annotations, was also deemed desirable.

In addition to lecture videos, users desired animation videos, such as of the progression of an environment as an object was instantiated or of state estimation. These short videos could be cropped from the longer lecture videos or created separately, increasing the frequency of video referencing within the Primer.

6.2 E-Textbooks

E-textbook discussion centered around the differences between the Primer and current e-textbooks, and what the optimal e-textbook would look like. In general, the customized presentation style of the Primer was deemed desirable in e-textbooks, and tailoring e-textbook presentation to a given login or username was a feature left to be desired in the state-of-the-art. E-textbooks for introductory courses would gain the
most from functionality like that of the Primer; e-textbooks for later courses would have an audience with a more uniform background and more familiarity with the language associated with a given domain.
Chapter 7

Contributions

The Primer represents a new advancement in educational technology. Both traditional learning environments and MOOCs could benefit from the ideas and implementation of the Primer. In this section I review the achievements of and possible extensions to the Primer.

7.1 The Primer

7.1.1 The Ideal

The Primer represents a step towards a more personalized web-delivered user experience in either a traditional or open classroom. The Primer functions in all popular web browsers and even from mobile and tablet devices. It incorporates the result of student feedback on the difficulties associated with using available resources into new resources that are designed to meet the needs of the individual student. It allows students to control the way information is presented to them, and to preserve that control in-between learning sessions. Unlike other e-textbooks, the Primer provides an e-textbook with customizations at the user level.
7.1.2 The Implementation

This implementation of the Primer, using 6.01 and 6.01 OCW Scholar materials, represents a usable alternative to the 6.01 course readings. The implementation of the Primer in this thesis merely needs user and user profile setup for students in order to be useful to the administration of 6.01. This implementation also has a simple enough interface to administrators to constitute an e-textbook authoring tool.

7.2 What's Next

Several small extensions to the Primer were conceived but not implemented before the completion of this thesis. A script to automate the creation of usernames and passwords would further prepare the thesis implementation of the Primer for use with a semester of 6.01. Primer elements that behave similar to definition elements but have different properties or difficulty coefficients could be created easily. CSS styling could be added to stylize the Primer or specific subsections of the Primer, such as the exercises, to visually distinguish them from the main content of the e-textbook. A self-serve account and profile creation page would reduce the amount of administrative hassle associated with the Primer. Giving users the opportunity to save their progress through the Primer could further reduce the setup required between learning sessions.

Larger extensions to the Primer could change the fundamental nature of the Primer, but are still worth considering. As pointed out during feedback collection, the ability to specify different difficulty ratings for a given definition element in a given chapter would allow changes to the dereferencing of a given term or video within a given chapter. This would represent an improvement as students are likely to become familiar with a definition element after a limited amount of exposures, and being able to decrease the amount of exposures in a given chapter without changing user profile settings is desirable. This could be accomplished by a quiz or reminder to adjust user profile settings.

Another extension worth consideration is the incorporation of communication between student and educator using the Primer. Primer administrators could leave
reading recommendations for struggling students, reading assignments for the entire class, and annotations regarding adjustments to students’ profiles. These features would allow a more personalized interaction surrounding the Primer without incorporating a separate communication medium.

One advantage a traditional textbook still harbors over the Primer is the ability to highlight and annotate text. The Primer would benefit from the inclusion of these features; however, they present a significant enough design challenge to constitute a large extension. Highlights and annotations could persist between sessions, and could be toggled on and off as a user profile setting.
Appendix A

Code

settings.py

```python
# Django settings for primer project.

DEBUG = False

TEMPLATE_DEBUG = DEBUG

ADMINS = (
    # ('Kendra Pugh', 'kpugh@mit.edu'),
)

MANAGERS = ADMINS

DATABASES = {

    'default': {
        'ENGINE': 'django.db.backends.sqlite3',  # Add 'postgresql_psycopg2', 'mysql', 'sqlite3' or 'oracle'.
        'NAME': '/srv/www/primer/sqlite3/primer.db',  # Or path to database file if using sqlite3.
        'USER': '',  # Not used with sqlite3.
        'PASSWORD': '',  # Not used with sqlite3.
        'HOST': '',  # Set to empty string for localhost. Not used with sqlite3.
    }
```

55
'PORT': '',
# Set to empty string for
default. Not used with sqlite3.
}
}

# Local time zone for this installation. Choices can be found here:
# http://en.wikipedia.org/wiki/List_of_tz_zones_by_name
# although not all choices may be available on all operating systems.
# On Unix systems, a value of None will cause Django to use the same
# timezone as the operating system.
# If running in a Windows environment this must be set to the same as
# your
# system time zone.
TIME_ZONE = 'America/New_York'

# Language code for this installation. All choices can be found here:
# http://www.i18nguy.com/unicode/language-identifiers.html
LANGUAGE_CODE = 'en-us'

SITE_ID = 1

# If you set this to False, Django will make some optimizations so as
# not
# to load the internationalization machinery.
USE_I18N = True

# If you set this to False, Django will not format dates, numbers and
# calendars according to the current locale.
USE_L10N = True

# If you set this to False, Django will not use timezone-ware datetimes
#.
USE_TZ = True

# Absolute filesystem path to the directory that will hold user-uploaded
# files.

MEDIA_ROOT = ''

MEDIA_URL = ''

STATIC_ROOT = ''

STATIC_URL = '/static/'

STATICFILES_DIRS = (  
    # Put strings here, like "'/home/html/static" or "C:/www/django/static".  
    # Always use forward slashes, even on Windows.  
    # Don't forget to use absolute paths, not relative paths.  
)

STATICFILES_FINDERS = (  
    'django.contrib.staticfiles.finders.FileSystemFinder',  
    'django.contrib.staticfiles.finders.AppDirectoriesFinder',  
    'django.contrib.staticfiles.finders.DefaultStorageFinder',  
)

# Make this unique, and don't share it with anybody.
SECRET_KEY = omitted

# List of callables that know how to import templates from various sources.
TEMPLATE_LOADERS = (
    'django.template.loaders.filesystem.Loader',
    'django.template.loaders.app_directories.Loader',
    # 'django.template.loaders.eggs.Loader',
)

MIDDLEWARE_CLASSES = (
    'django.middleware.common.CommonMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.middleware.csrf.CsrfViewMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',
    # Uncomment the next line for simple clickjacking protection:
    # 'django.middleware.clickjacking.XFrameOptionsMiddleware',
)

ROOT_URLCONF = 'primer.urls'

# Python dotted path to the WSGI application used by Django's runserver.
WSGI_APPLICATION = 'primer.wsgi.application'

TEMPLATE_DIRS = (
    # Put strings here, like "/home/html/django_templates" or "C:/www/
django/templates".
    # Always use forward slashes, even on Windows.
    # Don't forget to use absolute paths, not relative paths.
    '/srv/www/primer/templates'
)

INSTALLED_APPS = (
    'django.contrib.auth',
    'django.contrib.contenttypes',
)
'django.contrib.sessions',

#'django.contrib.sites',
'django.contrib.messages',
'django.contrib.staticfiles',

# Uncomment the next line to enable the admin:
'django.contrib.admin',

# Uncomment the next line to enable admin documentation:
# 'django.contrib.admindocs',
'book',

)

AUTH_PROFILE_MODULE = 'book.UserProfile'

# A sample logging configuration. The only tangible logging
# performed by this configuration is to send an email to
# the site admins on every HTTP 500 error when DEBUG=False.
# See http://docs.djangoproject.com/en/dev/topics/logging for
# more details on how to customize your logging configuration.
LOGGING = {
    'version': 1,
    'disable_existing_loggers': False,
    'filters': {
        'require_debug_false': {
            '()': 'django.utils.log.RequireDebugFalse'
        }
    },
    'handlers': {
        'mail_admins': {
            'level': 'ERROR',
            'filters': ['require_debug_false'],
            'class': 'django.utils.log.AdminEmailHandler'
        }
    },
    'loggers': {
        'django.request': {
            'handlers': ['mail_admins'],
        }
    }
}
models.py

```python
from django.db import models
from django.contrib.auth.models import User

class UserProfile(models.Model):
    user = models.OneToOneField(User)
    definitionRating = models.IntegerField()
    programmingBackground = models.IntegerField()
    systemsBackground = models.IntegerField()
    circuitsBackground = models.IntegerField()
    probabilityBackground = models.IntegerField()

    def __unicode__(self):
        return str(self.user) + "Profile"
```

urls.py

```python
from django.conf.urls import patterns, include, url

# Uncomment the next two lines to enable the admin:
from django.contrib import admin
admin.autodiscover()

urlpatterns = patterns('http://flahp.mit.edu/',
    url(r'^book/', include('book.urls')),
    url(r'^admin/', include(admin.site.urls)),
)
```
17
18 def create_user_profile(sender, instance, created, **kwargs):
19     if created:
20         UserProfile.objects.create(user=instance)
21
22         post_save.connect(create_user_profile, sender=User)
23
24 class Chapter(models.Model):
25     def __unicode__(self):
26         return "Chapter " + str(self.id)
27     advanceMultiplyer = models.DecimalField(max_digits=4, decimal_places=3)
28
29 class PrimerElement(models.Model):
30     name = models.CharField(max_length=100)
31     chapter = models.ManyToManyField(Chapter)
32     programmingThreshold = models.IntegerField()
33     systemsThreshold = models.IntegerField()
34     circuitsThreshold = models.IntegerField()
35     probabilityThreshold = models.IntegerField()
36     baseline = models.CharField(max_length=1000)
37     triggered = models.CharField(max_length=1000)
38
39     def __unicode__(self):
40         return self.name
41
42 class DefinitionElement(PrimerElement):
43     definitionThreshold = models.IntegerField()

views.py
1
2 from django.http import HttpResponseRedirect, HttpResponse
3 from django.core.urlresolvers import reverse
4 from book.models import Chapter, UserProfile, PrimerElement,
4 from django.shortcuts import get_object_or_404, render_to_response
61
from django.template import RequestContext
from django.contrib.auth import authenticate, login, logout

def index(request):
    return render_to_response('book/index.html')

def definitions(request, def_element):
    primerElement=DefinitionElement.objects.get(name=def_element)
    chapters = [x.pk for x in primerElement.chapter.all()]
    name = primerElement.baseline
    definition = primerElement_triggered
    return render_to_response('book/definitions/definitions_global.html',
        {'chapters': chapters,
        'name': name,
        'definition': definition})

def chapter(request, chapter_id):
    if not request.user.is_authenticated():
        return render_to_response('book/login.html',
            {'error_message': 'you must be logged in to use the Primer'},
            context_instance=(RequestContext(request)))
    else:

        primerDictionary = {}
        primerChapter = Chapter.objects.get(pk=chapter_id)
        currentProfile = UserProfile.objects.get(user=request.user)
        mul = primerChapter.advanceMultiplyer
        for element in DefinitionElement.objects.filter(chapter=primerChapter):
            if ((element.definitionThreshold * mul > \
            currentProfile.definitionRating) or
            (element.programmingThreshold * mul > \
            currentProfile.programmingBackground) or
(element.systemsThreshold * mul > \n    currentProfile.systemsBackground) or
(element.circuitsThreshold * mul > \n    currentProfile.circuitsBackground)
    or
(element.probabilityThreshold * mul > \n    currentProfile.probabilityBackground)
)

primerDictionary[element.name] = \n    element.baseline + " (" + \n    element.triggered + ") "
else:
    primerDictionary[element.name] = \n    '<a href="../../definitions/"+ \n    element.name +'/>' + element.baseline + \n    '</a>'

return render.to_response('book/chapter/+'+ \n    str(chapter.id)+'.'+html',
    primerDictionary)

def site_login(request):
    if request.user.is_authenticated():
        return render.to_response('book/login.html', {
            'error_message': str(request.user) + ' is already logged in <a href="../logout/"> logout </a> first chapter<a>'
        }, context_instance=RequestContext(request))
    else:
        return render.to_response('book/login.html',
            context_instance=RequestContext(request))

def logging_in(request):
    if request.user.is_authenticated():
        return render.to_response('book/login.html', {
            'error_message': str(request.user) + ' is already logged in <a href="../logout/"> logout </a>'
        }, context_instance=RequestContext(request))
username = request.POST['username']
password = request.POST['password']
user = authenticate(username=username, password=password)
if user is not None:
    if user.is_active:
        login(request, user)
        return HttpResponseRedirect(reverse('book.views.profile'))
    else:
        return render_to_response('book/login.html', {
            'error_message': 'inactive user',
        }, context_instance=RequestContext(request))
else:
    return render_to_response('book/login.html', {
        'error_message': 'Invalid Login',
    }, context_instance=RequestContext(request))
def site_logout(request):
    logout(request)
    return HttpResponse("Congratulations! You are logged out. <a href='../../login/'>login</a>")
def profile(request):
    user = request.user
    currentProfile = UserProfile.objects.get(user=request.user)
    return render_to_response('book/profile.html', {
        'user': user,
        'currentProfile': currentProfile,
    }, context_instance=RequestContext(request))
def submit_profile(request):
    user = request.user
    currentProfile = UserProfile.objects.get(user=user)
currentProfile.definitionRating = request.POST['definitionRating']
```python
currentProfile.programmingBackground = request.POST['
    programmingBackground']

currentProfile.systemsBackground = request.POST['
    systemsBackground']

currentProfile.circuitsBackground = request.POST['
    circuitsBackground']

currentProfile.probabilityBackground = request.POST['
    probabilityBackground']

currentProfile.save()

return render_to_response('book/profile.html', {
    'user': user,
    'currentProfile': currentProfile,
}, context_instance=RequestContext(request))
```
admin.py

```python
from book.models import Chapter, DefinitionElement, UserProfile
from django.contrib import admin

admin.site.register(Chapter)
admin.site.register(DefinitionElement)
admin.site.register(UserProfile)
```

base.html

```html
<!DOCTYPE html>
<html lang="en">
<head>
  <link rel="stylesheet" href="style.css" />
  <title>{% block title %}The Primer{% endblock title %}</title>
</head>
<body>
  <div id="masthead">
  </div>
</body>
</html>
```
</div>

<div id="content">

{% autoescape off %}
{% block content %}
{% endblock %}
{% endautoescape %}

</div>

</body>
</html>

book_base.html

{% extends "base.html" %}

login.html

{% extends "base.html" %}

{% block content %}
<h1>Login Page</h1>

{% if error-message %}<p><strong>{{ error-message }}</strong></p>{% endif %}

<form action="/book/logging-in/" method="post">
{% csrf_token %}
Username: <input type="text" name="username"><br />
Password: <input type="password" name="password"><br />
<input type="submit" value="Login" />
</form>
{% endblock %}

profile.html

{% extends "base.html" %}

{% block content %}

{% endblock %}
Welcome to the Primer

You are logged in as {{user.username}}

Edit your profile below. 0 is a novice, 10 is an expert.

<form action="/book/submit_profile/" method="post">
{% csrf-token %}

Definition Rating: <input type="text" name="definitionRating" value={{currentProfile.definitionRating}}>

Programming Background: <input type="text" name="programmingBackground" value={{currentProfile.programmingBackground}}>

Systems Background: <input type="text" name="systemsBackground" value={{currentProfile.systemsBackground}}>

Circuits Background: <input type="text" name="circuitsBackground" value={{currentProfile.circuitsBackground}}>

Probability Background: <input type="text" name="probabilityBackground" value={{currentProfile.probabilityBackground}}>

<input type="submit" value="Submit"/>
</form>

logout here

here is the first chapter

definitions_base.html

{% extends "book/book_base.html" %}

{% block title %}The Primer{% endblock %}

{% block content %}

Your Word goes here

Your definition goes here

{% for chapter in chapters %}

{% endblock %}
<a href='../../chapter/{{chapter}}/'>Back to Chapter {{chapter}}</a>

{% endfor %}

{% endblock %}

---

**definitions.html**

{% extends "book/book_base.html" %}

{% block title %}Primer Definitions{% endblock %}

{% block content %}

{% if definitions_list %}
<ul>
{% for definition in definitions_list %}
<li><a href="/book/definitions/{{definition.name}}/">{{ definition .baseline }}</a></li>
{% endfor %}

<p>No definitions are available.</p>
{% endif %}

{% endblock %}

---

**definitions_global.html**

{% extends "book/definitions/definitions_base.html" %}

{% block word %}{{name}}{% endblock %}

{% block definition %}{{ definition }}{% endblock %}

{% endblock %}

---

**chapter_base.html**

69
In the lab exercises of this course, we have implemented several brains for our robots. We used wall-following to navigate through the world and we used various linear controllers to drive down the hall and to control the robot head. In the first case, we just wrote a program that we hoped would do a good job. When we were studying...
linear controllers, we, as designers, made models of the controller's behavior in the world and tried to prove whether it would behave in the way we wanted it to, taking into account a longer-term pattern of behavior.

Often, we will want a system to generate complex long-term patterns of behavior, but we will not be able to write a simple control rule to generate those behavior patterns. In that case, we'd like the system to evaluate alternatives for itself, but instead of evaluating single actions, it will have to evaluate whole sequences of actions, deciding whether they're a good thing to do given the current state of the world.

Let's think of the problem of navigating through a city, given a road map, and knowing where we are. We can't usually decide whether to turn left at the next intersection without deciding on a whole path.

As always, the first step in the process will be to come up with a formal model of a real-world problem that abstracts away the irrelevant detail. So, what, exactly, is a path? The car we're driving will actually follow a trajectory through continuous space(time), but if we tried to plan at that level of detail we would fail miserably. Why? First, because the space of possible trajectories through two-dimensional space is just too enormous. Second, because when we're trying to decide which roads to take, we don't have the information about where the other cars will be on those roads, which will end up having a huge effect on the detailed trajectory we'll end up taking.

So, we can divide the problem into two levels: planning in advance which turns we'll make at which intersections, but deciding 'on-line', while we're driving, exactly how to control the steering wheel and the gas to best move from intersection to intersection, given the current circumstances (other cars, stop-lights, etc.).
We can make an abstraction of the driving problem to include road intersections and the way they're connected by roads. Then, given a start and a goal intersection, we could consider all possible paths between them, and choose the one that is best.

What criteria might we use to evaluate a path? There are all sorts of reasonable ones: distance, time, gas mileage, traffic-related frustration, scenery, etc. Generally speaking, the approach we'll outline below can be extended to any criterion that is additive: that is, your happiness with the whole path is the sum of your happiness with each of the segments. We'll start with the simple criterion of wanting to find a path with the fewest "steps"; in this case, it will be the path that traverses the fewest intersections. Then in the Uniform Cost section we will generalize our methods to handle problems where different actions have different costs.

One possible algorithm for deciding on the best path through a map of the road intersections in this (very small) world would be to enumerate all the paths, evaluate each one according to our criterion, and then return the best one. The problem is that there are lots of paths. Even in our little domain, with 9 intersections, there are 210 paths from the intersection labeled S to the one labeled G. We can get a much better handle on this problem, by formulating it as an instance of a graph search (or a "state-space search") problem, for which there are simple algorithms that perform well.
We'll model a state-space search problem formally as

- a (possibly infinite) set of states, the system can be in;
- a starting state, which is an element of the set of states;
- a goal test, which is a procedure that can be applied to any state, and returns True if that state can serve as a goal;
- a successor function, which takes a state and an action as input, and returns the new state that will result from taking the action in the state; and
- a legal action list, which is just a list of actions that can be legally executed in this domain.

The decision about what constitutes an action is a modeling decision. It could be to drive to the next intersection, or to drive a meter, or a variety of other things, depending on the domain. The only requirement is that it terminate in a well-defined next state (and that, when it is time to execute the plan, we will know how to execute the action.)

We can think of this model as specifying a labeled graph (in the computer scientist's sense), in which the states are the nodes, action specifies which of the arcs leading out of a node is to be selected, and the successor function specifies the node at the end of each of the arcs.

So, for the little world above, we might make a model in which
The set of states is the intersections \{'S', 'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H'\}.

The starting state is 'S'.

The {{goal}} test is something like:

```
lambda x: x == 'H'
```

The {{legal-action-list}} in this domain are the numbers 0, 1, ..., n-1, where n is the maximum number of successors in any of the states.

The map can be defined using a dictionary:

```
mapl = {'S': ['A', 'B'],
        'A': ['S', 'C', 'D'],
        'B': ['S', 'D', 'E'],
        'C': ['A', 'F'],
        'D': ['A', 'B', 'F', 'H'],
        'E': ['B', 'H'],
        'F': ['C', 'D', 'G'],
        'G': ['F', 'H']}
```

where each key is a state, and its value is a list of states that can be reached from it in one step.

Now we can define the {{successor-function}} as

```
def map1successors(s, a):
    return map1[s][a]
```

but with an additional test to be sure that if we attempt to take an action that doesn't exist in s, it just results in state s.
So, for example, the successor reached from state ‘A’ by taking action 1 is state ‘C’.

We can think of this structure as defining a search tree, like this:

It has the starting state, ‘S’, at the root node, and then each node has its successor states as children. Layer k of this tree contains all possible paths of length k through the graph.

Representing search trees

We will need a way to represent the tree as a Python data structure as we construct it during the search process. We will start by defining a class to represent a search node, which is one of the circles in the tree.

Each search node represents:

- The state of the node;
- the action that was taken to arrive at the node; and
- the search node from which this node can be reached.

We will call the node from which a node can be reached its parent node. So, for example, in the figure below we will represent the node with double circles around it with its state, ‘D’, the action that reached it, 1, and its parent node, which is the node labeled ‘B’ above it.

Note that states and nodes are not the same thing! In this...
tree, there are many nodes labeled by the same state; they represent
different paths to and through the state.</p>

<p>Here is a Python class representing a search {{node}}. It's pretty
straightforward.</p>

```python
class SearchNode:
    def __init__(self, action, state, parent):
        self.state = state
        self.action = action
        self.parent = parent

    def path(self):
        if self.parent == None:
            return [(self.action, self.state)]
        else:
            return self.parent.path() + [(self.action, self.state)]

    def inPath(self, s):
        if s == self.state:
```

The path corresponding to our double-circled node is <tt>((None, 'S'),
(1, 'B'), (1, 'D'))</tt>.</p>

Another helper method that we will find useful is the <tt>inPath</tt>
method, which takes a state, and returns <tt>True</tt> if the state
occurs anywhere in the path from the root to the node.

```python
def inPath(self, s):
    if s == self.state:
```
return True
eelif self.parent == None:
    return False
else:
    return self.parent.inPath(s)
</pre></p>
<p>
Basic search algorithm</p>

We'll describe a sequence of search algorithms of increasing sophistication and efficiency. An ideal algorithm will take a problem description as input and return a path from the start to a goal state, if one exists, and return None, if it does not. Some algorithms will not be capable of finding a path in all cases.

How can we systematically search for a path to the goal? There are two plausible strategies:

- Start down a path, keep trying to extend it until you get stuck, in which case, go back to the last choice you had, and go a different way. This is how kids often solve mazes. We'll call it *depth-first* search.
- Go layer by layer through the tree, first considering all paths of length 1, then all of length 2, etc. We'll call this *breadth-first* search.

Both of the search strategies described above can be implemented using a procedure with this basic structure:

```python
def search(initialState, goalTest, actions, successor):
    if goalTest(initialState):
        return [(None, initialState)]
    agenda = EmptyAgenda()
    add(SearchNode(None, initialState, None), agenda)
while not empty(agenda):
    parent = getElement(agenda)
    for a in actions:
        newS = successor(parent.state, a)
        newN = SearchNode(a, newS, parent)
        if goalTest(newS):
            return newN.path()
        else:
            add(newN, agenda)
    return None

We start by checking to see if the initial state is a \{goal\} state. If so, we just return a path consisting of the initial state.</p>

Otherwise, we have real work to do. We make the \emph{root} node of the tree. It has no parent, and there was no action leading to it, so all it needs to specify is its state, which is \tt{initialState}, so it is created with

SearchNode(None, initialState, None)

During the process of constructing the search tree, we will use a data structure, called an \emph{agenda}, to keep track of which \{node\}s in the partially-constructed tree are on the fringe, ready to be expanded, by adding their children to the tree.

We initialize the agenda to contain the root node. Now, we enter a loop that will run until the agenda is empty (we have no more paths to consider), but could stop sooner.</p>

Inside the loop, we select a node from the \{agenda\} (more on how we decide which one to take out in a bit) and \textit{expand it}. To expand
a node, we determine which actions can be taken from the state that is stored in the node, and visit the successor states that can be reached via the actions. </p>

When we visit a state, we make a new search node (newN, in the code) that has the node we are in the process of expanding as the parent, and that remembers the state being visited and the action that brought us here from the parent.</p>

Next, we check to see if the new state satisfies the goal test. If it does, we’re done! We return the path associated with the new node.</p>

If this state it doesn’t satisfy the goal test, then we add the new node to the agenda. We continue this process until we find a goal state or the agenda becomes empty. This is not quite yet an algorithm, though, because we haven’t said anything about what it means to add and extract nodes from the agenda. And, we’ll find, that it will do some very stupid things, in its current form.</p>

We’ll start by curing the stupidities, and then return to the question of how best to select nodes from the agenda.</p>

Basic pruning, or How not to be completely stupid</h3>

If you examine the full search tree, you can see that some of the paths it contains are completely ridiculous. It can never be reasonable, if we’re trying to find the shortest path between two states, to go back to a state we have previously visited on that same path. So, to avoid trivial infinite loops, we can adopt the following rule:
Don't consider any path that visits the same state twice.

If we can apply this rule, then we will be able to remove a number of branches from the tree, as shown here:

It is relatively straightforward to modify our code to implement this rule:

We've added code to our basic algorithm.
It just checks to see whether the current state already exists on the path to the node we're expanding and, if so, it doesn't do anything with it.
The next pruning rule doesn’t make a difference in the current domain,
but can have a big effect in other domains:

If there are multiple actions that lead from a state $r$ to a state $s$, consider only one of them.

To handle this in the code, we have to keep track of which new states we have reached in expanding this node, and if we find another way to reach one of those states, we just ignore it. The changes to the code for implementing this rule are shown:

```python
def search(initialState, goalTest, actions, successor):
    if goalTest(initialState):
        return [(None, initialState)]
    agenda = [SearchNode(None, initialState, None)]
    while agenda != []:
        parent = getElement(agenda)
        newChildStates = []
        for a in actions:
            newS = successor(parent.state, a)
            newN = SearchNode(a, newS, parent)
            if goalTest(newS):
                return newN.path()
            elif newS in newChildStates:
                pass
            elif parent.inPath(newS):
                pass
            else:
                newChildStates.append(newS)
                add(newN, agenda)
        return None
```

Each time we pick a new node to expand, we make a new empty list, and keep track of all of the new states we have
In designing algorithms, we frequently make use of two simple data structures: stacks and queues. You can think of them both as abstract data types that support two operations: \texttt{push} and \texttt{pop}. The \texttt{push} operation adds an element to the stack or queue, and the \texttt{pop} operation removes an element. The difference between a stack and a queue is what element you get back when you do a \texttt{pop}.

- **Stack**: When you \texttt{pop} a stack, you get back the element that you most recently put in. A stack is also called a LIFO, for \texttt{last in, first out}.
- **Queue**: When you \texttt{pop} a queue, you get back the element that you put in earliest. A queue is also called a FIFO, for \texttt{first in, first out}.

In Python, we can use lists to represent both stacks and queues. If \texttt{data} is a list, then \texttt{data.pop(0)} removes the first element from the list and returns it, and \texttt{data.pop()} removes the last element and returns it.

Here is a class representing stacks as lists. It always adds new elements to the end of the list, and pops items off of the same end, ensuring that the most recent items get popped off first.

```python
class Stack:
    def __init__(self):
```

```python
def __init__(self):
```
Here is a class representing stacks as lists. It always adds new elements to the end of the list, and pops items off of the front, ensuring that the oldest items get popped off first.

```python
class Queue:
    def __init__(self):
        self.data = []
    def push(self, item):
        self.data.append(item)
    def pop(self):
        return self.data.pop(0)
    def isEmpty(self):
        return self.data is []
</pre>
</p>
<p>We will use stacks and queues to implement our search algorithms.</p>
<h3>Depth-First Search</h3>
Now we can easily describe {depth_first_search} by saying that it's an instance of the generic search procedure described above, but in which the agenda is a <em>stack</em>: that is, we always expand the node we most recently put into the agenda.</p>

The code listing below shows our implementation of depth-first search.

```python
def depthFirstSearch(initialState, goalTest, actions, successor):
```
agenda = Stack()
if goalTest(initialState):
    return [(None, initialState)]
agenda.push(SearchNode(None, initialState, None))
while not agenda.isEmpty():
    parent = agenda.pop()
    newChildStates = []
    for a in actions:
        newS = successor(parent.state, a)
        newN = SearchNode(a, newS, parent)
        if goalTest(newS):
            return newN.path()
        elif newS in newChildStates:
            pass
        elif parent.inPath(newS):
            pass
        else:
            newChildStates.append(newS)
            agenda.push(newN)
return None
</pre>

You can see several operations on the {{agenda}}. We:

<ul>
<li>Create an empty <tt>Stack</tt> instance, and let that be the agenda.
<li>Push the initial node onto the agenda.
<li>Test to see if the agenda is empty.
<li>Pop the node to be expanded off of the agenda.
<li>Push newly visited nodes onto the agenda.
</ul>

Because the agenda is an instance of the <tt>Stack</tt> class, subsequent operations on the agenda ensure that it will act like a stack, and guarantee that children of the most recently expanded node will be chosen for expansion next.
So, let's see how this search method behaves on our city map, with start state \( S \) and goal state \( F \). Here is a trace of the algorithm (you can get this in the code we distribute by setting \( \texttt{verbose = True} \) before you run it.)

```python
depthFirst('S', lambda x: x == 'F', map1LegalActions, map1successors)
```

agenda: Stack([S])

expanding: S

agenda: Stack([S-0->A, S-1->B])

expanding: S-1->B

agenda: Stack([S-0->A, S-1->B-1->D, S-1->B-2->E])

expanding: S-1->B-2->E

agenda: Stack([S-0->A, S-1->B-1->D, S-1->B-2->E-1->H])

expanding: S-1->B-2->E-1->H


expanding: S-1->B-2->E-1->H-2->G

8 states visited

[(None, 'S'), (1, 'B'), (2, 'E'), (1, 'H'), (2, 'G'), (0, 'F')]

You can see that in this world, the search never needs to "backtrack", that is, to go back and try expanding an older path on its agenda. It is always able to push the current path forward until it reaches the goal.

Here is the search tree generated during the depth-first search process.

Here is another city map (it's a weird city, we know, but maybe a bit like trying to drive in Boston):
In this city, depth-first search behaves a bit differently (trying to go from \(<TT>S</TT>\) to \(<TT>D</TT>\) this time):

```python
depthFirst('S', lambda x: x == 'D', map2LegalActions, map2successors)
```

Here are some important properties of depth-first search:

- It will run forever if we don’t apply pruning rule 1, potentially going back and forth from one state to another, forever.
- It may run forever in an infinite domain (as long as the path it’s on has a new successor that hasn’t been previously visited, it can go down that path forever; we’ll see an example of this in the last section).
- It doesn’t necessarily find the shortest path (as we can see from the very first example).
<li> It is generally efficient in the amount of space it requires to store the agenda, which will be a constant factor times the depth of the path it is currently considering (we’ll explore this in more detail later).
</li></p>

<h3>Breadth-First Search</h3>

To change to `breadth.first.search`, we need to choose the oldest rather than the newest paths from the agenda to expand. All we have to do is change the agenda to be a queue instead of a stack, and everything else stays the same, in the code.

```python
def breadthFirstSearch(initialState, goalTest, actions, successor):
    agenda = Queue()
    if goalTest(initialState):
        return [(None, initialState)]
    agenda.push(SearchNode(None, initialState, None))
    while not agenda.isEmpty():
        parent = agenda.pop()
        newChildStates = []
        for a in actions:
            newS = successor(parent.state, a)
            newN = SearchNode(a, newS, parent)
            if goalTest(newS):
                return newN.path()
            elif newS in newChildStates:
                pass
            elif parent.inPath(newS):
                pass
            else:
                newChildStates.append(newS)
                agenda.push(newN)
    return None
```

</pre>
Here is how \texttt{breadth-first-search} works, looking for a path from \texttt{S} to \texttt{F} in our first city:

```plaintext
>>> breadthFirst('S', lambda x: x == 'F', map1LegalActions, map1successors)
agenda: Queue([S])
expanding: S
agenda: Queue([S-0->A, S-1->B])
expanding: S-0->A
expanding: S-1->B
expanding: S-0->A-1->C
7 states visited
[(None, 'S'), (0, 'A'), (1, 'C'), (1, 'F')]
```

We can see it proceeding systematically through paths of length two, then length three, finding the goal among the length-three paths.

Here are some important properties of \texttt{breadth-first search}:

- Always returns a shortest (least number of steps) path to a goal state, if a goal state exists in the set of states reachable from the start state.
- It may run forever if there is no solution and the domain is infinite.
- It requires more space than \texttt{depth-first search}.

Dynamic programming

Let's look at \texttt{breadth-first search} in the first city map example, but this time with goal state \texttt{G}:

88
breadthFirst('S', lambda x: x == 'G', mapLegalActions, 
  mapSuccessors)

agenda: Queue([S])
expanding: S

agenda: Queue([S-0->A, S-1->B])
expanding: S-0->A

expanding: S-1->B

expanding: S-0->A-1->C

expanding: S-0->A-2->D

expanding: S-1->B-1->D

expanding: S-1->B-2->E

expanding: S-0->A-1->C-1->F

16 states visited

[(None, 'S'), (0, 'A'), (1, 'C'), (1, 'F'), (2, 'G')]

The first thing that is notable about this trace is that it ends up visiting 16 states in a domain with 9 different states. The issue is that it is exploring multiple paths to the same state. For instance, it has both <tt>S-0->A-2->D</tt> and <tt>S-1->B-1->D</tt> in the agenda. Even worse, it has both <tt>S-0->A</tt> and <tt>S-1->B-1->D-0->A</tt> in the agenda! We really don’t need to consider all of these paths. We can make use of the following example of the dynamic programming principle:
The shortest path from \( \langle \text{TT} \rangle X \langle / \text{TT} \rangle \) to \( \langle \text{TT} \rangle Z \langle / \text{TT} \rangle \) that goes through \( \langle \text{TT} \rangle Y \langle / \text{TT} \rangle \) is made up of the shortest path from \( \langle \text{TT} \rangle X \langle / \text{TT} \rangle \) to \( \langle \text{TT} \rangle Y \langle / \text{TT} \rangle \) and the shortest path from \( \langle \text{TT} \rangle Y \langle / \text{TT} \rangle \) to \( \langle \text{TT} \rangle Z \langle / \text{TT} \rangle \). So, as long as we find the shortest path from the start state to some intermediate state, we don’t need to consider any other paths between those two states; there is no way that they can be part of the shortest path between the start and the goal. This insight is the basis of a new pruning principle:

Don’t consider any path that visits a state that you have already visited via some other path.

In \{breadth.first.search\}, because of the orderliness of the expansion of the layers of the search tree, we can guarantee that the first time we visit a state, we do so along the shortest path. So, we’ll keep track of the states that we have visited so far, by using a dictionary, called \langle tt \rangle visited \langle / tt \rangle \) that has an entry for every state we have visited. (An alternative representation would be just to keep a Python \langle tt \rangle set \langle / tt \rangle \) of visited \{node\}s. Then, if we are considering adding a new node to the tree that goes to a state we have already visited, we just ignore it. This test can take the place of the test we used to have for pruning rule 1; it’s clear that if the path we are considering already contains this state, then the state has been visited before. Finally, we have to remember, whenever we add a node to the agenda, to add the corresponding state to the visited list.

Here is our \{breadth.first.search\} code, modified to take advantage of dynamic programming.
```python
def breadthFirstDP(initialState, goalTest, actions, successor):
    agenda = Queue()
    if goalTest(initialState):
        return [(None, initialState)]
    agenda.push(SearchNode(None, initialState, None))
    visited = {initialState: True}
    while not agenda.isEmpty():
        parent = agenda.pop()
        for a in actions:
            newS = successor(parent.state, a)
            newN = SearchNode(a, newS, parent)
            if goalTest(newS):
                return newN.path()
            elif visited.has_key(newS):
                pass
            else:
                visited[newS] = True:
                agenda.push(newN)
    return None

So, let’s see how this performs on the task of going from `<TT>S</TT>` to `<TT>G</TT>` in the first city map:

```
expanding: S-0→A-2→D
agenda: Queue([S-1→B-2→E, S-0→A-1→C-1→F, S-0→A-2→D-3→H])
expanding: S-1→B-2→E
agenda: Queue([S-0→A-1→C-1→F, S-0→A-2→D-3→H])
expanding: S-0→A-1→C-1→F
8 states visited
[(None, 'S'), (0, 'A'), (1, 'C'), (1, 'F'), (2, 'G')]
As you can see, this results in visiting significantly fewer states.
Here is the tree generated by this process:

![tree3.jpg](/media/images/8/tree3.jpg)

In bigger problems, this effect will be amplified hugely, and will
make the difference between whether the algorithm can run in a
reasonable amount of time, and not.

We can make the same improvement to {{depth_first_search}}; we just
need
to use a stack instead of a queue in the algorithm above. It still will
not guarantee that the shortest path will be found, but will guarantee
that we never visit more paths than the actual number of states. The
only change to breadth-first search with dynamic programming is that
the new states are added to the beginning of the agenda.

Because all of our search algorithms (breadth-first and depth-first,
with and without dynamic programming) are all so similar, and we don't
like to repeat code, we provide (in file <tt>search.py</tt>) a single,
configurable search procedure. It also prints out some information
as it goes, if you have the <tt>verbose</tt> or <tt>somewhatVerbose</tt>
variables set to <tt>True</tt>, and has a limit on the maximum number of
nodes it will expand (to keep from going into an infinite loop).

def search(initialState, goalTest, actions, successor,
     depthFirst = False, DP = True, maxNodes = 10000):
    if depthFirst:
agenda = Stack()

else:
    agenda = Queue()

startNode = SearchNode(None, initialState, None)
if goalTest(initialState):
    return startNode.path()
agenda.push(startNode)
if DP: visited = {initialState: True}
count = 1
while not agenda.isEmpty() and maxNodes > count:
    n = agenda.pop()
    newStates = []
    for a in actions:
        newS = successor(n.state, a)
        newN = SearchNode(a, newS, n)
        if goalTest(newS):
            return newN.path()
        elif newS in newStates:
            pass
        elif ((not DP) and n.inPath(newS)) or 
              (DP and visited.has_key(newS)):
            pass
        else:
            count += 1
            if DP: visited[newS] = True
            newStates.append(newS)
            agenda.push(newN)
return None
</pre></p>

<p><h2>Connection to state machines</h2></p>

We can use state machines as a convenient representation of state-space search problems. Given a {{state-machine}}, in its initial state, what sequence
of inputs can we feed to it to get it to enter a done state? This is a search problem, analogous to determining the sequence of actions that can be taken to reach a goal state.

The `getNextValues` method of a state machine can serve as the successor function in a search (the inputs to the machine are the actions). Our standard machines do not have a notion of legal actions; but we will add an attribute called `legalInputs`, which is a list of values that are legal inputs to the machine (these are the actions, from the planning perspective) to machines that we want to use with a search.

The `startState` attribute can serve as the initial state in the search and the `done` method of the machine can serve as the goal test function.

Then, we can plan a sequence of actions to go from the start state to one of the done states using this function, where `smToSearch` is an instance of `sm.SM`.

```python
def smSearch(smToSearch, initialState = None, goalTest = None, maxNodes = 10000, depthFirst = False, DP = True):
    if initialState == None:
        initialState = smToSearch.startState
    if goalTest == None:
        goalTest = smToSearch.done
    return search(initialState, goalTest, smToSearch.legalInputs, # This returns the next state
                  lambda s, a: smToSearch.getNextValues(s, a)[0],
                  maxNodes = maxNodes,
                  depthFirst=depthFirst, DP=DP)
```

94
It is mostly clerical: it allows us to specify a different initial state or \{\text{goal}\} test if we want to, and it extracts the appropriate functions out of the \{\text{state-machine}\} and passes them into the search procedure. Also, because <tt>getNextValues</tt> returns both a state and an output, we have to wrap it inside a function that just selects out the next state and returns it.

<h2>Numeric search domain</h2>

Many different kinds of problems can be formulated in terms of finding the shortest path through a space of states. A famous one, which is very appealing to beginning calculus students, is to take a derivative of a complex equation by finding a sequence of operations that takes you from the starting expression to one that doesn’t contain any derivative operations. We’ll explore a different simple one here:

- The states are the integers.
- The initial state is some integer; let’s say 1.
- The legal actions are to apply the following operations: \{2n, n+1, n−1, n^2, −n\}.
- The goal test is \lambda x: x = 10.

So, the idea would be to find a short sequence of operations to move from 1 to 10.

Here it is, formalized as state machine in Python:

```python
class NumberTestSM(sm.SM):
  startState = 1
  legalInputs = ['x*2', 'x+1', 'x-1', 'x**2', '-x']
  def __init__(self, goal):
    self.goal = goal
  def nextState(self, state, action):
    if action == 'x*2':
      return state * 2
```
```python
def getNextValues(self, state, action):
    nextState = self.nextState(state, action)
    return (nextState, nextState)

def done(self, state):
    return state == self.goal
```

First of all, this is a bad domain for applying `depth_first_search`.

Why? Because it will go off on a gigantic chain of doubling the starting state, and never find the goal. We can run `breadth_first_search`, though. Without dynamic programming, here is what happens (we have set `<tt>verbose = False</tt>` and `<tt>somewhatVerbose = True</tt>` in the search file):

```python
>>> smSearch(NumberTestSM(10), initialState = 1, depthFirst = False, DP = False)
expanding: 1
expanding: 1--x+2--->2
expanding: 1--x--1--->0
expanding: 1--x--1--->0
expanding: 1--x--1--->0
expanding: 1--x--1--->0
expanding: 1--x--1--->0
expanding: 1--x--1--->0
expanding: 1--x--1--->0
```

96
expanding: $1-x \rightarrow -1-x+1 \rightarrow 0$

expanding: $1-x+1 \rightarrow 0$

expanding: $1-x \rightarrow -1$

expanding: $1-x-1 \rightarrow 0$

expanding: $1-x \rightarrow -1$

expanding: $1-x \rightarrow -1$

expanding: $1-x \rightarrow -1$

expanding: $1-x \rightarrow -1$

expanding: $1-x \rightarrow -1$

expanding: $1-x+1 \rightarrow 0$

expanding: $1-x+1 \rightarrow 0$

expanding: $1-x+1 \rightarrow 0$

expanding: $1-x+1 \rightarrow 0$

expanding: $1-x+1 \rightarrow 0$

expanding: $1-x+1 \rightarrow 0$

expanding: $1-x+1 \rightarrow 0$

expanding: $1-x+1 \rightarrow 0$

expanding: $1-x+1 \rightarrow 0$

We find a nice short path, but visit 33 states. Let's try it with DP:

```python
terminal_search(NumberTestSM(10), initialState=1, depthFirst=False, DP=True)
```

expanding: $1$

expanding: $1-x \rightarrow -2$

expanding: $1-x \rightarrow -2$

expanding: $1-x \rightarrow -2$

We find the same path, but visit noticeably fewer states. If we change the goal to 27, we find that we visit 564 states without DP and 119, with. If the goal is 1027, then we visit 12710 states without DP and 1150 with DP, which is getting to be a very big difference.

To experiment with `depth_first_search`, we can make a version of the problem where the state space is limited to the integers in some range. We do this by making a subclass of the `NumberTestSM`, which remembers the maximum legal value, and uses it to restrict the set of legal inputs for a state (any input that would cause the successor state to go out of bounds just results in staying at the same state, and it will be pruned.)
Here's what happens if we give it a range of -20 to +20 to work in:

```python
>>> smSearch(NumberTestFiniteSM(10, 20), initialState = 1, depthFirst = True,
     DP = False)
expanding: 1
expanding: 1--x-->-1
expanding: 1--x-->-1-x+1-->0
expanding: 1--x-->-1-x*2-->-2
expanding: 1--x-->-1-x*2-->-2--x-->2
expanding: 1--x-->-1-x*2-->-2--x-->2-x+1-->3
expanding: 1--x-->-1-x*2-->-2--x-->2-x+1-->3--x-->3--x**2-->9
20 states visited
[(None, 1), ('-x', -1), ('x*2', -2), ('-x', 2), ('x+1', 3), ('-x', -3),
 ('x**2', 9), ('x+1', 10)]
```

We generate a much longer path!

We can see from trying lots of different searches in this space that
(a) the DP makes the search much more efficient and (b) that the
difficulty of these search problems varies incredibly widely.

<h2>Computational complexity</h2>
To finish up this segment, let's consider the computational complexity of these algorithms. As we've already seen, there can be a huge variation in the difficulty of a problem that depends on the exact structure of the graph, and is very hard to quantify in advance. It can sometimes be possible to analyze the average case running time of an algorithm, if you know some kind of distribution over the problems you're likely to encounter. We'll just stick with the traditional worst-case analysis, which tries to characterize the approximate running time of the worst possible input for the algorithm.

First, we need to establish a bit of notation. Let

- \( b \) be the branching factor of the graph; that is, the number of successors a node can have. If we want to be truly worst-case in our analysis, this needs to be the maximum branching factor of the graph.
- \( d \) be the maximum depth of the graph; that is, the length of the longest path in the graph. In an infinite space, this could be infinite.
- \( l \) be the solution depth of the problem; that is, the length of the shortest path from the start state to the shallowest goal state.
- \( n \) be the state space size of the graph; that is the total number of states in the domain.

\{\text{depth-first-search}\}, in the worst case, will search the entire search tree. It has \( d \) levels, each of which has \( b \) times as many paths as
the previous one. So, there are $b^d$ paths on the $d^{th}$ level.

The algorithm might have to visit all of the paths at all of the levels, which is about $b^{d+1}$ states. But the amount of storage it needs for the agenda is only $bd$. 

The algorithm might have to visit all of the paths at all of the levels, which is about $b^{d+1}$ states. But the amount of storage it needs for the agenda is only $bd$. 

So, to be clear, consider the numeric search problem. The branching factor $b = 5$, in the worst case. So, if we have to find a sequence of 10 steps, breadth-first search could potentially require visiting as many as $5^{11} = 48828125$ nodes!

This is all pretty grim. What happens when we consider the DP version of {{breadth_first_search}}? We can promise that every state in the state space is visited at most once. So, it will visit at most $n$ states. Sometimes $n$ is much smaller than $b^l$ (for instance, in a road network). In other cases, it can be much larger (for instance, when you are solving an easy (short solution path) problem embedded in a very large space). Even so, the DP version of the search will visit fewer states, except in the very rare case in which there are never multiple paths to the same state (the graph is actually a tree). For example, in the numeric search problem, the shortest path from 1 to 91 is 9 steps long, but using DP it only requires visiting 1973 states, rather
than $5^{10} = 9765625$.

In many cases, the arcs in our graph will actually have different costs. In a road network, we would really like to find the shortest path in miles (or in time to traverse), and different road segments have different lengths and times. To handle a problem like this, we need to extend our representation of search problems, and add a new algorithm to our repertoire.

We will extend our notion of a successor function, so that it takes a state and an action, as before, but now it returns a pair $(\text{newS}, \text{cost})$, which represents the resulting state, as well as the cost that is incurred in traversing that arc. To guarantee that all of our algorithms are well behaved, we will require that all costs be positive (not zero or negative).

Here is our original city map, now with distances associated with the roads between the cities.

We can describe it in a dictionary, this time associating a cost with each resulting state, as follows:

```
map1dist = {'S': [('A', 2), ('B', 1)],
            'A': [('S', 2), ('C', 3), ('D', 2)],
            'B': [('S', 1), ('D', 2), ('E', 3)],
            'C': [('A', 3), ('F', 1)],
            'D': [('A', 2), ('B', 2), ('F', 4), ('H', 6)],
            'E': [('B', 3), ('H', 2)],
            'F': [('C', 1), ('D', 4), ('G', 1)],
            'H': [('D', 6), ('E', 2), ('G', 4)],
            'G': [('F', 1), ('H', 4)]}
```
When we studied {{{breadth_first_search}}}, we argued that it found the shortest path, in the sense of having the fewest nodes, by seeing that it investigate all of the length 1 paths, then all of the length 2 paths, etc. This orderly enumeration of the paths guaranteed that when we first encountered a goal state, it would be via a shortest path. The idea behind {{{uniform cost search}}} is basically the same: we are going to investigate paths through the graph, in the order of the sum of the costs on their arcs. If we do this, we guarantee that the first time we extract a {{{node}}} with a given state from the {{{agenda}}}, it will be via a shortest {{{path}}}, and so the first time we extract a node with a {{{goal}}} state from the agenda, it will be an optimal solution to our problem.  

<h4>Priority Queue</h4>  
Just as we used a stack to manage the agenda for depth-first search and a queue to manage the agenda for bread-first search, we will need to introduce a new data structure, called a {{{priority queue}}} to manage the agenda for {{{uniform_cost_search}}}. A priority queue is a data structure with the same basic operations as stacks and queues, with two differences:

<ul>
<li>Items are pushed into a priority queue with a numeric score, called a {{{cost}}}.</li>
<li>When it is time to pop an item, the item in the priority queue with the least {{{cost}}} is returned and removed from the priority queue.</li>
</ul>

There are many interesting ways to implement priority queues so that they are very computationally efficient. Here, we show a very simple implementation that simply walks down the entire contents of the priority queue to find the least-{{cost}} item for a pop operation. Its <tt>data</tt> attribute consists of a list of <tt>(cost, item)</tt> pairs. It calls the <tt>argmaxIndex</tt> procedure from our utility package,
which takes a list of items and a scoring function, and returns a pair consisting of the index of the list with the highest scoring item, and the score of that item. Note that, because \texttt{argmaxIndex()} finds the item with the highest score, and we want to extract the item with the least cost, our scoring function is the negative of the cost.

```python
class PQ:
    def __init__(self):
        self.data = []
    def push(self, item, cost):
        self.data.append((cost, item))
    def pop(self):
        (index, cost) = util.argmaxIndex(self.data, lambda (c, x): -c)
        return self.data.pop(index)[1]  # just return the data item
    def isEmpty(self):
        return self.data is []
</pre>

<h4>UC Search</h4>

Now, we're ready to study the \texttt{uniform\_cost\_search} algorithm itself. We will start with a simple version that doesn't do any pruning or dynamic programming, and then add those features back in later.

First, we have to extend our definition of a \texttt{SearchNode}, to incorporate costs. So, when we create a new search \texttt{node}, we pass in an additional parameter \texttt{actionCost}, which represents the cost just for the action that moves from the parent node to the state.

Then, we create an attribute \texttt{self.cost}, which encodes the cost of this entire path, from the starting state to the last state in the path. We compute it by adding the path cost of the parent to the cost of this last action, as shown by the red text below.

```python
class SearchNode:
```
Now, here is the search algorithm. It looks a lot like our standard search algorithm, but there are two important differences:

- The agenda is a priority queue.
- Instead of testing for a goal state when we put an element into the agenda, as we did in breadth_first_search, we test for a goal state when we take an element out of the agenda. This is crucial, to ensure that we actually find the shortest path to a goal state.

```python
def ucSearch(initialState, goalTest, actions, successor):
    startNode = SearchNode(None, initialState, None, 0)
    agenda = PQ()
    agenda.push(startNode, 0)
    while not agenda.isEmpty():
        n = agenda.pop()
        if goalTest(n.state):
            return n.path()
        for a in actions:
            (newS, cost) = successor(n.state, a)
```
if not n.inPath(newS):
    newN = SearchNode(a, newS, n, cost)
    agenda.push(newN, newN.cost)
    return None
</pre></p>

<h4>Example</h4>

Consider the following simple graph:

Let's simulate the uniform-cost search algorithm, and see what happens when we try to start from \( S \) and go to \( D \):

- The agenda is initialized to contain the starting node. The agenda is shown as a list of cost, node pairs.

- The least-cost node, \( S \), is extracted and expanded, adding two new nodes to the agenda. The notation \( S-0->A \) means that the path starts in state \( S \), takes action \( 0 \), and goes to state \( A \).

- The least-cost node, \( S-1->B \), is extracted, and expanded, adding one new node to the agenda. Note, that, at this point, we have discovered a path to the goal: \( S-1->B-1->D \) is a path to the goal, with cost 11. But we cannot be sure that it is the shortest path to the goal, so we simply put it into the agenda, and wait to see if it gets extracted before any other path to a goal state.
1: expanding: S-1→B

agenda: PQ([(2, S-0→A), (11, S-1→B-1→D)])

The least-cost node, S-0→A is extracted, and expanded, adding one new node to the agenda. At this point, we have two different paths to the goal in the agenda.

2: expanding: S-0→A

agenda: PQ([(11, S-1→B-1→D), (4, S-0→A-1→D)])

Finally, the least-cost node, S-0→A-1→D is extracted. It is a path to the goal, so it is returned as the solution.

5 states visited; Solution cost: 4

[(None, 'S'), (0, 'A'), (1, 'D')]

Dynamic programming

Now, we just need to add dynamic programming back in, but we have to do it slightly differently. We promise that, once we have expanded a node, that is, taken it out of the agenda, then we have found the shortest path to that state, and we need not consider any further paths that go through that state. So, instead of remembering which nodes we have visited (put onto the agenda) we will remember nodes we have expanded (gotten out of the agenda), and never visit or expand a node that has already been expanded. In the code below, the first test ensures that we don’t expand a node that goes to a state that we have already found the shortest path to, and the second test ensures that we don’t put any additional paths to such a state into the agenda.
```python
def ucSearch(initialState, goalTest, actions, successor):
    startNode = SearchNode(None, initialState, None, 0)
    if goalTest(initialState):
        return startNode.path()
    agenda = PQ()
    agenda.push(startNode, 0)
    expanded = {}
    while not agenda.isEmpty():
        n = agenda.pop()
        if not expanded.has_key(n.state):
            expanded[n.state] = True
            if goalTest(n.state):
                return n.path()
            for a in actions:
                (newS, cost) = successor(n.state, a)
                if not expanded.has_key(newS):
                    newN = SearchNode(a, newS, n, cost)
                    agenda.push(newN, newN.cost)
    return None
```

Here is the result of running this version of `uniform-cost-search` on our bigger city graph with distances:

```python
mapDistTest(map1dist, 'S', 'G')
agenda: PQ([(0, S)])
  0 : expanding: S
agenda: PQ([(2, S-0->A), (1, S-1->B)])
  1 : expanding: S-1->B
agenda: PQ([(2, S-0->A), (3, S-1->B-1->D), (4, S-1->B-2->E)])
  2 : expanding: S-0->A
agenda: PQ([(3, S-1->B-1->D), (4, S-1->B-2->E), (5, S-0->A-1->C), (4, S-0->A-2->D)])
  3 : expanding: S-1->B-1->D
```

107
agenda: PQ(\{(4, S-1\rightarrow B-2\rightarrow E), (5, S-0\rightarrow A-1\rightarrow C), (4, S-0\rightarrow A-2\rightarrow D), (7, S-1\rightarrow B-1\rightarrow D-2\rightarrow F), (9, S-1\rightarrow B-1\rightarrow D-3\rightarrow H)\})

4: expanding: S-1\rightarrow B-2\rightarrow E

agenda: PQ(\{(5, S-0\rightarrow A-1\rightarrow C), (4, S-0\rightarrow A-2\rightarrow D), (7, S-1\rightarrow B-1\rightarrow D-2\rightarrow F), (9, S-1\rightarrow B-1\rightarrow D-3\rightarrow H), (6, S-1\rightarrow B-2\rightarrow E-1\rightarrow H)\})

5: expanding: S-0\rightarrow A-1\rightarrow C

agenda: PQ(\{(7, S-1\rightarrow B-1\rightarrow D-2\rightarrow F), (9, S-1\rightarrow B-1\rightarrow D-3\rightarrow H), (6, S-1\rightarrow B-2\rightarrow E-1\rightarrow H), (6, S-0\rightarrow A-1\rightarrow C-1\rightarrow F)\})

6: expanding: S-1\rightarrow B-2\rightarrow E-1\rightarrow H

agenda: PQ(\{(7, S-1\rightarrow B-1\rightarrow D-2\rightarrow F), (9, S-1\rightarrow B-1\rightarrow D-3\rightarrow H), (6, S-0\rightarrow A-1\rightarrow C-1\rightarrow F), (10, S-1\rightarrow B-2\rightarrow E-1\rightarrow H-2\rightarrow G)\})

6: expanding: S-0\rightarrow A-1\rightarrow C-1\rightarrow F

agenda: PQ(\{(7, S-1\rightarrow B-1\rightarrow D-2\rightarrow F), (9, S-1\rightarrow B-1\rightarrow D-3\rightarrow H), (10, S-1\rightarrow B-2\rightarrow E-1\rightarrow H-2\rightarrow G), (7, S-0\rightarrow A-1\rightarrow C-1\rightarrow F-2\rightarrow G)\})

13 states visited; Solution cost: 7

[(None, 'S'), (0, 'A'), (1, 'C'), (1, 'F'), (2, 'G')]

</pre></p>

<p><h3>Connection to state machines</h3></p>

When we use a {{state-machine}} to specify a domain for a cost-based search, we only need to make a small change: the <tt>getNextValues</tt> method of a state machine can still serve as the <tt>successor</tt> function in a {{search}} (the inputs to the machine are the actions). We usually think of <tt>getNextValues</tt> as returning the next state and the output: now, we will modify that interpretation slightly, and think of it as returning the next state and the incremental {{cost}} of taking the action that transitions to that next state. This has the same form as the the <tt>ucSearch.search</tt> procedure expects a {{successor.function}} to have, so we don't need to change anything about the <tt>smSearch</tt> procedure we have already defined.</p>
Search with heuristics

Ultimately, we’d like to be able to solve huge state-space search problems, such as those solved by a GPS that can plan long routes through a complex road network. We’ll have to add something to \{uniform_cost_search\} to solve such problems efficiently.

Let’s consider the city below, where the actual distances between the intersections are shown on the arcs:

If we use \{uniform_cost_search\} to find a path from \texttt{G}\texttt{X} to \texttt{X}, we expand states in the following order (the number at the beginning of each line is the length of the path from \texttt{G} to the state at the end of the path):

```bash
>>> bigTest('G', 'X')
0 : expanding: G
14.2 : expanding: G->H
20.7 : expanding: G->F
25.5 : expanding: G->I
30.1 : expanding: G->H->T
32.3 : expanding: G->H->D
36.7 : expanding: G->I->J
40.5 : expanding: G->I->C
44.3 : expanding: G->H->T->R
45.9 : expanding: G->H->E
50.4 : expanding: G->H->D->S
50.5 : expanding: G->I->L
52.6 : expanding: G->I->J->M
54.7 : expanding: G->H->D->B
54.7 : expanding: G->I->C->A
54.8 : expanding: G->I->J->K
58.5 : expanding: G->H->T->R->V
```
expanding: G-0->I-3->J-1->K-1->N
expanding: G-0->I-3->J-2->M-1->P
expanding: G-0->I-2->L-1->O
expanding: G-0->I-3->J-2->M-1->P-1->Q
expanding: G-2->H-2->T-1->R-1->V-2->Y
expanding: G-0->I-2->L-1->O-1->W
expanding: G-(->I-2->L-1->O-2->U

This search process works its way out, radially, from G, expanding nodes in contours of increasing path length. That means that, by the time the search expands node X, it has expanded every single node.

This seems kind of silly: if you were looking for a good route from G to X, it's unlikely that states like S and B would ever come into consideration.

What is it about state B that makes it seem so irrelevant?

Clearly, it's far away from where we want to go. We can incorporate this idea into our search algorithm using something called a heuristic function. A heuristic function takes a state as an argument and returns a numeric estimate of the total cost that it will take to reach the goal from there. We can modify our search algorithm to be biased toward states that are closer to the goal, in the sense that the heuristic function has a smaller value on them.

In a path-planning domain, such as our example, a reasonable heuristic is the actual Euclidean distance between the current state and the
goal state; this makes sense because the states in this domain are actual locations on a map.

If we modify the \{uniform\_cost\_search\} algorithm to take advantage of a heuristic function, we get an algorithm called \texttt{A^*} (pronounced 'a star'). It is given below, with the differences highlighted in red. The \textit{only} difference is that, when we insert a \{node\} into the priority queue, we do so with a \{cost\} that is \texttt{newN.cost + heuristic(newS)}. That is, it is the sum of the actual cost of the path from the start state to the current state, and the estimated cost to go from the current state to the goal.

```python
def ucSearch(initialState, goalTest, actions, successor, heuristic):
    startNode = SearchNode(None, initialState, None, 0)
    if goalTest(initialState):
        return startNode.path()
    agenda = PQ()
    agenda.push(startNode, 0)
    expanded = {}
    while not agenda.isEmpty():
        n = agenda.pop()
        if not expanded.has_key(n.state):
            expanded[n.state] = True
            if goalTest(n.state):
                return n.path()
            for a in actions:
                (newS, cost) = successor(n.state, a)
                if not expanded.has_key(newS):
                    newN = SearchNode(a, newS, n, cost)
                    agenda.push(newN, newN.cost + heuristic(newS))
```

111
Example

Now, we can try to search in the big map for a \{path\} from \(<tt>G</tt>\) to \(<tt>X</tt>\), using, as our heuristic function, the distance between the state of interest and \(<tt>X</tt>\). Here is a trace of what happens (with the numbers rounded to increase readability):

- We get the start node out of the \{agenda\}, and add its children. Note that the costs are the actual path cost plus the \{heuristic\} estimate.

0 : expanding: G
agenda: PQ([(107, G-0->I), (101, G-1->F), (79, G-2->H)])

14.2 : expanding: G-2->H

The least cost path is \(<tt>G-2->H</tt>\), so we extract it, and add its successors.

30.1 : expanding: G-2->H-2->T

Now, we can see the \{heuristic\} function really having an effect. The path \(<tt>G-2->H-2->T</tt>\) has length 30.1, and the path \(<tt>G-1->F</tt>\) has length 20.7. But when we add in the heuristic cost estimates, the path to \(<tt>T</tt>\) has a lower cost, because it seems to be going in the right direction. Thus, we select \(<tt>G-2->H-2->T</tt>\) to expand next.
agenda: PQ([(107, G-0->I), (101, G-1->F), (109, G-2->H-0->D), (116, G-2->H-1->E), (100, G-2->H-2->T-1->R)])
</pre>
</li>
<li> Now the path <tt>G-2->H-2->T-1->R</tt> looks best, so we expand it.

44.3 : expanding: G-2->H-2->T-1->R
</pre>
</li>
<li> Here, something interesting happens. The node with the least estimated cost is <tt>G-1->F</tt>. It's going in the wrong direction, but if we were to be able to fly straight from <tt>F</tt> to <tt>X</tt>, then that would be a good way to go. So, we expand it:

20.7 : expanding: G-1->F
agenda: PQ([(107, G-0->I), (109, G-2->H-0->D), (116, G-2->H-1->E), (123, G-1->F-0->D), (133, G-1->F-1->C)])
</pre>
</li>
<li> Continuing now, basically straight to the goal, we have:

58.5 : expanding: G-2->H-2->T-1->R-1->V
</pre>
</li>
<li> Expanding: G-2->H-2->T-1->R-1->V-2->Y
</pre>
</li>
Using \{A\text{-star}\} has roughly halved the number of \{nodes\} \{visit\}ed and \{expand\}ed. In some problems it can result in an enormous savings, but, as we'll see in the next section, it depends on the heuristic we use.

In order to think about what makes a heuristic good or bad, let's imagine what the perfect heuristic would be. If we were to magically know the distance, via the shortest path in the graph, from each node to the goal, then we could use that as a heuristic. It would lead us directly from start to \{goal\}, without expanding any extra nodes. But, of course, that's silly, because it would be at least as hard to compute the heuristic function as it would be to solve the original search problem.

So, we would like our heuristic function to give an estimate that is as close as possible to the true shortest-path-length from the state to the goal, but also to be relatively efficient to compute.
An important additional question is: if we use a heuristic function, are we still guaranteed to find the shortest path through our state space? The answer is: yes, if the heuristic function is admissible. A heuristic function is admissible if it is guaranteed to be an underestimate of the actual cost of the optimal path to the goal. To see why this is important, consider a state from which the goal can actually be reached in 10 steps, but for which the heuristic function gives a value of 100. Any path to that state will be put into the agenda with a total cost of 90 more than the true cost. That means that if a path is found that is as much as 89 units more expensive than the optimal path, it will be accepted and returned as a result of the search.

It is important to see that if our heuristic function always returns value 0, it is admissible. And, in fact, with that heuristic, the A* algorithm reduces to uniform cost search.

In the example of navigating through a city, we used the Euclidean distance between cities, which, if distance is our cost, is clearly admissible; there's no shorter path between any two points.

Would the so-called 'Manhattan distance', which is the sum of the absolute differences of the coordinates be an admissible heuristic in the city navigation problem, in general? Would it be admissible in Manhattan?

If we were trying to minimize travel time on a road network (and so the estimated time to travel each road segment was the cost), what would be an appropriate heuristic function?
1467 </p>
1468
1469 <p>
1470 {{Costs.Heuristics_and.A_star.video}}
1471 {% endblock %}


