Division II Portfolio

Alec Goebel

February 11, 2013

Contents

I Division II **4**

1 Retrospective **5**

2 Multiple Cultural Perspective **8**

3 CEL-2 **10**

II Coursework **11**

4 Summary **12**

5 Hampshire **13**
5.1 Pygame Course (4 evals) . . . 14
5.2 Automatic Lecture Recording (2 evals) . . . 16
5.3 Teaching Assistant - Web Courses (2 evals) . . . 19
5.4 Researching Gamification and DSLs . . . 20
5.5 Creative Programming (2 evals) . . . 21
5.6 Genetic Programming (ta eval only) . . . 24
5.7 OpenGL Independent Study . . . 25
5.8 Video Game Design 2 . . . 26
5.9 Improvisor's Laboratory . . . 28

No Evaluations **29**
5.10 Mobile Computing . . . 30
5.11 Developing TheHub . . . 31
5.12 Application-Oriented Databases . . . 32

6 WPI **34**
6.1 Technical Game Development 1 . . . 35
6.2 Machinima . . . 36
6.3 Human Computer Interaction . . . 37
6.4 Storytelling in IMGD . . . 38
6.5 Accelerated Intro to Program Design . . . 39
6.6 Object-Oriented Design Concepts . . . 40
6.7 Systems Programming Concepts . . . 41
6.8 Intro to Artificial Intelligence . . . 42

7 Other **43**

III Sample Works **44**

8 Programming Languages **45**
8.1 Python . . . 46
8.2 Clojure . . . 47
8.3 Javascript . . . 48
8.4 Java . . . 49
8.5 C/C++ . . . 52

9 Pygame Course **54**
9.1 Course . . . 55

10 Game Development **56**

11 Web Development **57**
11.1 Frontend . . . 58
11.2 Backend . . . 63

12 Other Interfaces **64**
12.1 HCI Course . . . 65
12.2 Android . . . 66
12.3 GUI Libraries . . . 67

13 Miscellaneous **68**
13.1 Waking . . . 69
13.2 Gamification . . . 71

Part I

Division II

Chapter 1

Retrospective

Concentration

Human-Computer Interaction (HCI) has always been where my passion lies. While I started out pursuing game development, my interests rapidly evolved into interactive media and interfaces in general. By interface I do not just mean GUIs; more often I design programmer interfaces: APIs, DSLs, scripts, and frameworks.

For the last few years I concentrated on finding ways to close the semantic gap between how humans and computers "think". I attempt to create more intuitive, automated, and modular interfaces. I develop DSLs and APIs to write software in the problem's domain instead of the language or system's logic. In addition, I have been looking at the problem from the other side; teaching others to program.

Areas of Study

Programming

Every so often I hear Computer Science majors/concentrators exclaim that they are great at Computer Science but bad at programming. I am left baffled, as if a painter told me they cannot mix colors. Although my interests can be academic and I am considering becoming a professor, at the end of the day I am a programmer first. I take pride in being a code monkey and put a lot of effort into honing my craft.

To that end, I primarily studied the art of programming, taking a liberal arts approach to my study of Computer Science. In addition to core CS courses, I studied a broad range of topics, many simply out of curiosity. I learned a variety of programming languages[1] and through them gained a strong understanding of how to work within various paradigms.[2] The majority of my academic work has also involved working with project teams, teaching me to communicate and organize my ideas better.

In addition to my academic work, I spend most of my free time on personal study and programming projects. Since I started programming in high school, I doubt I have gone more than a week without programming something. I try to learn a little bit about everything I come across; the areas I study can be ecclectic at

[1] e.g. Python, Clojure, Java, C/C++
[2] e.g. functional, object-oriented, procedural, declarative

times.[3] I spend a lot of time mastering my tools[4] and learning pragmatic techniques.[5] I also regularly create new tools to improve my development environment or automate my life.[6] At this point, I am far more likely to curl up on a couch with source code than a novel.

This course of study has given me a strong foundation in software development. The largest and most challenging software project I tackled at Hampshire was the lecture recording system with Paul Dickson. During my time on the team I developed a web frontend, setup the backend, and continuously debugged the live system. All the while we continued developing the next iteration of the software. This project drew upon my skills in various languages, interface design, Linux, and sleep deprivation. The work paid off, and the system recorded a full semester's worth of courses for the first time.

At this point, I am on the path to being a "guru"; I have started to develop an intuition and fluency in designing software.

Interfaces

My passion in HCI stemmed from of my passion for video games. More than other interfaces, games need to be incredibly instructive and responsive. I entered WPI as a Technical IMGD major (Interactive Media and Game Design) and studied a wide range of game design topics including puzzle design, player experience and motivation, multiplayer interaction, interactive storytelling, and game balance. As a tech major I also studied game engine architecture, AI, and graphics, ultimately creating games and artist tools. Additionally, I was an active member of our Game Development Club (GDC), and was elected President in my last year at WPI.

Eventually, my interest in game development evolved into a more general interest in how users interact with computers. I took my skills in game creation and started applying them to creating more interactive web and mobile applications. During my time between schools, I taught myself various web backends.[7] I tried to keep up with the changing HTML5 and CSS3 specs as well as a wide assortment of javascript libraries and practices. In addition, I taught myself as much as I could about developing Android applications and their structure.

Over the years I learned HCI fundamentals such as interface design analysis, storyboards, affordances, and usability studies. In my projects I try to factor out any reusable libraries, plugins, and widgets as I go. More recently I have been studying technical user interface design including frameworks, APIs, and DSLs.

Instruction

Before I went to WPI, I took a year off to volunteer for Americorps in Bridgeport, CT. I worked at RYASAP as a Youth Development Coordinator working with youth from inner city Bridgeport and the surrounding suburbs. In my time there, I had my fingers in many pies; I tutored grade schoolers in a public housing after school program, helped plan and run a youth leadership development week for high schoolers, was involved with youth groups of all ages, sat in on various boards, wrote and approved grassroot grants, worked with homeless shelters, and helped out around the office.

Even years later I can only describe the experience as undescribable. I was thrown into situations with individuals who perceived the world radically differently than I. These days, I try to make fewer assumptions and promote the voices of those involved rather than simply raising my own. More than anything else Ive come to appreciate listening more than talking.

[3] e.g. formal grammars, lambda calculus, tripstores
[4] e.g. IDEs, Linux, debuggers, version control, issue trackers
[5] e.g. debugging, logging, unit tests, prefactoring, iterative development, spec gathering
[6] e.g. shell scripts, cronjobs, vim plugins
[7] e.g. django, pylons, PHP, GAE, CGI

When it comes to technology, I have always found myself in the role of instructor. Whether it is people asking for help with computers or me rambling excitedly about CS concepts, I love sharing my knowledge with others. At WPI I more formally pursued IT support as a helpdesk employee, serving as a mouthpiece for the larger IT department. Between schools I continued this path working as a support technician in an ISP call center. Over time these jobs have helped me radically improve my patience and persistence. They have given me the ability to explain myself and understand the perspectives and confusions of nontechnical people.

While I enjoy helping people with their computer problems, I am driven to teach people to program. Originally, I would just help my friends and peers every chance I could with their programming projects. A large part of my involvement in WPIs GDC was running crash courses in pygame, teaching freshmen game design concepts, and supporting the development teams however possible. Starting last year, I took every TA opportunity I could and tried to make myself available for anyone with a programming question on campus. This culminated in what I (half) jokingly call my first Division 3. Taking everything I learned as a CS student and TA, I developed a CS1 course in Python and game development for non-programmers with Paul Dickson (2012S CS-114). I created the overall curriculum, the in-class examples, sample code, homework assignments, homework submission and grading tools, and taught/assisted in every class. In addition, I taught the material to the other TAs and professor, held lab hours 3 times a week (often lasting 5 hours each), and prepared/taught a section on Fridays for the advanced students. In the end, I have a whole new appreciation and understanding on how to develop and run a course. This has only increased my appetite for teaching.

Outcome

Division III

As a final project in my studies of Human Computer Interaction I intend to design and develop a touch-based, visual scripting environment (TVSE). As a practical case, it will allow users to program a robotics kit on the Android platform.

Conclusion

It has been over a decade since I wrote my first few lines of code and about 4-5 years since I completed my Division I at WPI. Despite the sometimes rocky path I took, I am proud of what I have learned and accomplished in that time. I have studied how to create a wide range of interfaces for a wide range of media. I improved on my fundamental skills as a coder. I consumed as much information on as many CS topics as I could. Not content to remain a simple code monkey, I have been trying to improve as an instructor.

Most importantly, to me anyway, I feel I have done and learned what I can as a Division II student, and look forward to what I will be learning next.

Chapter 2

Multiple Cultural Perspective

Growing up I was only vaguely aware of my inherent social safeties. In 2005, I took a gap year after high school to be a fulltime Americorps volunteer in Bridgeport, CT. As a Youth Development Coordinator I worked with people who not only saw the world differently, but had backgrounds which varied widely from my own. These differences were not caused by personality traits or geography; they arose from racial, socio-economic, and gender marginalization. More than anything else, I became aware how much of the agency I possess derives from the color of my skin, where I was born, and how I was raised.

Having been raised a Unitarian Universalist, I was partially prepared for and open to other world views. That year I found that the people I met were not necessarily familiar with ideas and concepts that I had considered universal. Also during this time, I witnessed things I had only heard of; I encountered homelessness, homophobia, and the aftermath of gang shootings.

One thing I was not prepared for was how little the internet had penetrated day to day life in inner city Bridgeport. In high school, I rarely went more than a day without going online. But the people I encountered did not. For them, the internet was more of a luxury, a novelty used by businesses and teens.

Today, the internet is becoming inextricably intertwined with everyday life, shifting from luxury to necessity. Those without access are separated from a large portion of our society, inhibiting their social mobility. Presently, many forces are at work trying to narrow the access gap. But obtaining access is not enough; the true disparity in power arises from an individual's inability to navigate the vast ocean of information to gain knowledge. We need to encourage users to ask questions, search for answers, and filter information; as a whole, we need to work towards "internet savviness". Only then can the internet act as a cultural equalizer, where anyone can be more informed, more connected, and more heard.

Considering its promise, it is unacceptable that a great number of people have no means of access. Physical or financial, language or literacy, the entry barriers are numerous. No longer a mere inconvenience, lack of access leads to disenfranchisement. As more people connect through social networks, those who cannot lose their voice. When job listings are online, those without access have more trouble finding work. The internet is already a necessity for students. Online only services may leave out those who could benefit most.

Most people connect through service providers whose rates are calculated as if access is a luxury. There is a need for lower or subsidized rates so more people can afford to go online at home without being unduely burdened. This is why a great deal of effort and resources are now being focused on making the internet nationally available. Public grants and corporate investments fund computers in schools and libraries, broadband build-outs in rural areas, and free WiFi in cities. The costs of consumer electronics and highspeed access have dropped, and most cellphones are internet capable. In addition the number and variety of services have grown to accomodate the rising demand. Consequently, internet penetration and usage has skyrocketed over the last few years.

Unfortunately, everyone on the internet is a target for social engineers looking to prey on others. Though anyone can fall victim to phishing, scams, "too good to be true" deals, and identity theft, the impact can be more devastating on those less fortunate. However, savvy users can recognize when something "feels wrong", making them less likely to be victimized. Also, they know how to use multiple sources to distinguish fact from fiction, letting them see through misinformation. They are able to see a larger picture drawn from filtering irrelevant or incorrect information. They are more proficient at selecting search terms, yielding more accurate results. They know how to navigate the web: where they are, how they got there, and how to return. They know how to find answers to their own questions, taking advantage of others' expertise. Their savviness comes from knowing how to fish, not just asking for one.

Many services on the internet are free; anyone can sign up and immediately use email, Facebook, Twitter, and innumerable other online communities. These connections have grown into a standard form of communication and are becoming de rigueur, vital for keeping in touch with friends, family, and the rest of the world. This growing need to be part of the 24-7 online discourse is driving smartphone sales and usage. Unfortunately, these devices and the data plans they require are expensive, costing at least three times as much as a landline. Even the most affordable data plans are capped, leading to significant overage charges for unsuspecting users.

Owners may subscribe to a variety of additional services, for yet another monthly fee. Individual songs, videos, books, and apps are priced to seem cheap but quickly add up, surprising many when the bill comes. Apps advertised as free can require an additional purchase to unlock the full functionality. Pay-to-play games offer shortcuts in exchange for every nickel and dime. At the end of the month, additional luxury purchases have eclipsed the anticipated financial burden; what started as a necessity quickly becomes unaffordable.

The exploding internet population will continue to face challenges surrounding access, inexperience, and exploitation. Social policies will attempt to reduce access inequity and explotation. But it is up to individuals to begin a cultural shift to "internet savviness". Be patient when people are uninformed, taking the time to share personal experience. Encourage new users to examine less expensive solutions that still meet their need. Advocate alternative software and services, especially ones which are free and/or open source. Encourage people to question what they are told, to protect themselves from becoming victims. Show people how to find answers instead of just spoon feeding them. As individuals, we should share our knowledge to help others become more savvy and reduce the power disparity between users.

Personally, I have tried to hold to these ideals in my own life: as a coder, IT technician, nonprofit worker, and now aspiring educator.

Chapter 3

CEL-2

TODO: WEB APP TA credit

Part II

Coursework

Chapter 4

Summary

Chapter 5

Hampshire

5.1 Pygame Course (4 evals)

:Courses:	CS200-7 CS200 TA-CS-2 CS-300-3
:Prof:	Paul Dickson
:Terms:	2011 Fall 2012 Jan 2012 Spring
:Lang:	Python
:Lib:	Pygame

Self Eval

This course was my "first div 3". It was the primary thing I was working on the entire school year. First, I roughly planned out the curriculum as an independent study. Over Jan term I taught Pygame to the other TA's and Paul, as well as fleshing out the actual in class examples and homework assignments. Then, in the spring I TA-ed and ran an advanced section on Fridays.

Although I program all the time, I don't think I have ever written so much for a single project before. I created turnin scripts, step by step versions of the in class examples, the homeworks assignments (including ones for the advanced section), and unit tests to grade some of the homework assignments. When the class moved on to final projects, I created several small samples to show how to create platformers, cameras, and menus. By the end of the class, I had a working final project of my own (Super Coin Get) and a multiplayer beta (Super Coin Get Madness).

I pushed myself a little too hard on the time commitment. In addition to attending each class and sticking around after, I held about 10 lab hours a week (my days from 6-12 and the other TA's days.) Then it was back to writing new class examples and the homework for the next week.

In addition to doing a substantial amount of the prep work, I taught a Friday addendum course and on occasion when Paul was out. The advanced section was mixed, and we covered a wide range of things: from design patterns to python nicities. Still, almost everyone in the advanced section created wonderful things for their final using some of the techniques we covered. It wasn't what I planned, but it worked out.

Overall, the course was flawed but it was a good first run. There are so many things I'd do differently were I to get the chance to teach it again. I'd slow the pace, replace some of the examples, get rid of github entirely, etc.

This course was the most rewarding part of my education to date and something I am truly proud of, especially when I see how far some students have come since they started.

Evaluations

Independent Study - Designing a Course in Pygame

Alec Goebel did a great job in this independent study and succeeded in building an introductory programming course in Python that uses Pygame as motivation. He designed a course of study that covers the

fundamentals of computer science and presents them in an order that logically fits with game design. He has developed lecture notes for the course and the homework assignments that correspond as well as building an infrastructure for students to turn in assignments. Alec designed a course that uses standard practices for program development in industry to ensure that students will learn real world programming concepts. Alec would have been hard pressed to do a better job in this independent study or with this course.

Preparation of a Python CS 1 Course

For this independent study Alec Goebel worked closely with two other students and a faculty member to prepare an introductory computer science course that uses game development to teach Python. This group filled in details of the course that Alec developed the previous semester in a different independent study. He took the lead in creating code for lecture examples and homework assignments. Alec was the leader who kept the other two students working on this project on task and moving towards the final goal. Through the development of this material Alec was able to improve his understanding of teaching and to gain a new perspective on course development. Alec could not have done a better job in this independent study and a course could not have been created without his contributions.

Teaching Assistant - CS-112: Programming Games - Introduction to Programming Using Python and Pygame

Alec was the lead teaching assistant for CS112 an introductory programming course in Python that used Pygame and game development to motivate learning to program. In this roll he organized the other TAs, did the majority of the grading, held lab hours, and developed class examples and homework assignments. In addition he attended class and helped students during class time. Where Alec really excelled was in his contributions to the lectures. He both added insights to material presented in class and taught the 3 lectures when I was out of town. Alec was the glue that held this course together and without his efforts the course could not have happened. He did have a little trouble meeting deadlines for preparing material but this was outweighed by his overall effort. Alec was a phenomenal TA for this course.

Addendum to CS-112 Programming Games

For this independent study Alec Goebel took the material from the CS 112 course he was TAing and added to it, teaching a small group of students additional material. In order to do this Alec taught one 80 minute lecture a week that built on the CS 112 course material from that week. The purpose of this was to fill in details that are important for computer science but don't fit into an introductory game programing course. Alec designed all of the material for these lectures and wrote and graded homework assignments that went along with them. He did a great job organizing this and teaching it and the students who attended his lectures benefited from them. Alec could not have done a better job with this and the experience should serve him well when next he teaches a course.

5.2 Automatic Lecture Recording (2 evals)

:Course: IS-CS-200
IS-CS-200-7

:Term: 2011 Jan
2011 Spring

:Prof: Paul Dickson

:Langs: bash
python
javascript
c++

:Libs: Django
OpenCV

Self Evaluation

The largest portion of my first year at Hampshire was spent working on Paul's automated lecture recording system. Since I spent my time away from school learning linux system administration and python, this was a "final project" in many ways.

Due to filing issues, one of my Spring studies actually took place during January. With Warshow, I spent Jan term working 10-15 hours a day to get the system up and running in time for the semester. This involved writing the capture software, bash/cron jobs, rewriting the web server, and connecting it all up. It was intense, but in the end we had the system more or less together by the time courses started.

At first the system had some troubles, but I helped make sure to plan ahead during our crunch so it was more debugging than reworking. Pretty soon, it was trivial to add new courses to the system and aside from checking the status remotely or running a cleanup command, the system was running smoothly.

That said, the system was quite hacky. It was stuck together with a few bash scripts, shell based mutex locks that spanned two servers, and other little nasties scattered throughout. Though working, it wasn't maintainable. This was fine though, since the system was due for a reworking that summer.

During the semester, I intended to do more HCI design on the page. Sadly, the system did not seem far enough along to really analyze how people were using it or what could make it better.

That isn't to say I wasn't busy. Instead I worked in all the other facets and did what little research and GUI stuff was available. I did get to help with the end of semester usability study. Also, since it was recording classes, the only time we could go in for repairs was at night, leaving us in a constant "crunch".

While a challenging few months, I learned all about how hard and precarious it is to work on a production system. I learned how to better deal with things when they are on fire so it doesn't just catch fire again next week. I also got much better with the linux OS, specifically making separate programs play nicely together without a formal IPC. Most of all, I'm just glad I had already learned to plan for future features so most of the modifications were simple once things were running.

This project taught me a fair amount about API interfaces: making sure to document and set everything up with the future in mind so that new components can just be dropped in with little effort. Early on I added django commands which proved useful so all it took to create new courses was a quick ssh command and editing the crontab.

Evaluation: CS-200-7 (ISP) (2011S)
Web Interface Design and Implementation

Instructor Narrative

The goal of this independent study was to get a previously built automatic lecture capture system deployed and capturing lectures. Alec was successful in achieving this goal. He was able to get the system capturing, processing, and posting lectures robustly without requiring any human interaction. He was also able to get this system to record five separate courses that were taught in the classroom containing the lecture capture equipment, and to have the captured lectures posted within twenty-four hours of the class being recorded. This was especially impressive considering that a graduate student who had been working on the same project had been unable to accomplish this same task in a period of two years. Alec could not have been more successful in this independent study.

Evaluation: CS-200 (ISP) (2011S)
Web Interface Design and Implementation

Web Interface Design and Implementation

The goal of this independent study was to update the lecture presentation software associated with a lecture capture system, and Alec was successful in achieving this goal. Alec was able to take the existing presentation software and smooth out its performance by including a new video player and adding video streaming. He was also able to expand the list of web browsers within which the software runs. After completion of this update, Alec was able to run both focus groups and surveys aimed at finding out what parts of the interface to improve. Alec was very successful in this independent study and made significant progress in improving this software.

5.3 Teaching Assistant - Web Courses (2 evals)

Course: CS106, CS206 Prof: Paul Dickson Term: 2011 Fall, 2012 Jan Lang: javascript, python Libs: google app engine

Self Eval

These two courses were my first time TAing and completely changed my outlook on education. I knew on some level that it took more effort to teach than do, but I didn't really have an appreciation for it. What surpised me most was how long it takes to create and grade an assigment versus just doing it.

There isn't much to say about the courses themselves. Students made websites and I helped mostly with the javascript and CSS. Still, I got hooked on TAing. Finding out how to translate my own thoughts into how others think is something I was good at from working previously in IT. Being able to do it in the field I'm passionate about was even better.

Evaluations

CS106: Programming Web Pages For Poets, Artists, and Scientists

Alec Goebel did a great job as a teaching assistant for this web design course. He attended one lecture a week and added greatly to those lectures by giving students a different perspective on the course material. Alec ran one session of lab hours every week where he went above and beyond answering student questions, often staying late because of student need. He also graded three of the homework assignments. Alec's presence as a teaching assistant improved the course.

CS206: Building Web Applications

Alec did not originally intend to TA this class but as Jan-Term progressed it became obvious that his help was needed with the course and he volunteered. Alec helped students in class and out and made sure that they could build their websites. He helped with lectures and made sure that all concepts were explained clearly. This course would not have been successful without Alec's contributions. He did a great job as a TA for this course.

5.4 Researching Gamification and DSLs

Course: IS-CS-200-3 Prof: Paul Dickson Term: 2012 Spring

Self Eval

This study shifted a fair amount over the semester. Originally I was trying to design a system to quickly create game elements for Hampshire campus. This would be things like CASA being able to give out achievements to mark one's progress through the divisional system. Clubs could quickly generate challenges for their members. As I figured out more aspects, I attempted to create the design for an easy to use interface for non-technical users.

As the year went on, the design became unwieldy so I shifted my focus down to how the system would work for a single course. Taking elements of game design, I worked out a way for text book material to be laid out in an "explorable" graph. In addition, I spent a lot of time planning how small homework assignments would work so students couldn't actually fail them while still being challenged.

The one thing I noticed was that although a little research had been done, I could find very little information on the area I was looking for. Though people had documented how to create badges, I could find no simple ways to create to layout a "game world" out of an n-dimensional graph.

Overall, I'm proud of the results. I am currently about 10 pages into the specification documentation, with many times more notes and thoughts. Though this was originally my intended Div 3, as I thought more about it over the summer I realized the project was a bit nebulous and vague. It was hard to imagine having something I was proud to show off in a year, especially with my current experience level. At some point I'd like to work more on it, but not now.

Evaluation

The goal of this independent study was to learn more about the field of Gamification. Gamification is the process of taking some everyday task and turning it into a game which gives you points based on doing that task. Alec Goebel spent the beginning of the semester researching Gamification before deciding to Gamify an introductory programming course. To Gamify the course he figured out how material could be broken into individual pieces that could become challenges. An example of what Alec came up with was doing problems related to course material, the more problems they could solve on the course material the higher the player's score and the more merits they win. This is only one of the ways that Alec created to make learning the material a game and to relate score to learning. Alec was successful in this study because he developed all of the mechanisms necessary to turn a course into a game.

5.5 Creative Programming (2 evals)

:Course: TA-CS-5
CS-236

:Term: 2012 Fall
2010 Fall

:Prof: Lee Spector

:Lang: Clojure

:Lib: quil

Self Evaluation

Student

As with many of my other courses in Fall 2010, being back in college as an experienced programmer was interesting. I already knew Lisps and had worked briefly with Clojure the summer before.

The problem that plauged me during the course is one I still wrestle with at Hampshire: scope creep. In attempts to challenge myself in new and interesting ways I would come up with projects which in reality would take me several months to do as a full time job. As a result, I bounced between ideas and never really settled on anything I wanted to work on AND had time to work on. In the end, I only turned in the toy processing problems I had created to get used to atoms.

The big regret I have for this course is not finishing my Google Code Jam parser DSL. It was a way to rapidly parse out the input files into variable bindings. I got caught with nested macros or some other strange bug.

Teaching Assistant

Looking at this course from the other side reinforced how I was expecting too much from myself the last time around. The projects most of the students were working on were ones I could do in an afternoon. And often did.

For TAing this class I tried making functional prototypes of people's work on my own before I spoke with them to get a better idea on how they should approach the problem. This wound up being a double edged sword. On one hand, my thoughts were clearer and I didn't stumble through an explanation, figuring it out as I went. On the other hand, it was harder for me to change paths on the fly to go with a simpler solution for a student having more trouble.

Overall, I think I did well as TA. Despite sometimes helping too much I'm confident students were still learning from what we did together. Going forward, I'll probably try to give students a best guess on how to make something and send them off to work while I try implementing it on my own. This should allow for a more immediate discussion, give them time to get started, and give me time to figure out where they will probably get stuck.

Evaluation: TA-CS (CS-109) (2012F)
Programming Creativity

Course Description

This course is an introduction to computer science and programming framed by the question, Is it possible for a computer to be creative?" The core areas of computer science will be introduced, including algorithms, complexity, computability, programming languages, data structures, systems, and artificial intelligence, with an eye toward the insights that they can provide about issues of computational creativity. Students will complete several programming projects to demonstrate developing technical skills and engagement with the themes of the course." Cumulative Skills: QUA, IND

Instructor Narrative

Alec was the sole teaching assistant for this introductory computer science course and his duties included attending class regularly, helping students during in-class coding sessions, helping to administer in-class presentations, answering student questions during office hours and by email, and helping to organize submitted student work.

This was a particularly challenging teaching assistantship because the students in the class had an unusually wide range of backgrounds and because many students had ambitious project ideas that required the rapid acquisition of new programming skills. Alec worked hard and consciously to help all students who approached him, but the conditions of the class made it inevitable that he would sometimes provide students with more new information than they could digest, or put students on a path that they could not complete on their own. Alec appreciated these difficulties, he mitigated them to the best of his ability, and he wrote about them intelligently in his self-evaluation. In the end, despite these difficulties, it is clear to me that Alec did an excellent job of helping students, and that many of the students in the class owe much of their success to Alec's help. Alec's performance on his other duties – helping me, rather than the students – was uniformly excellent. In sum, Alec did a great job on this difficult assignment, and I was very happy to have his assistance.

Evaluation: CS-236 (2010F)
Creative Programming Workshop

Course Description

In this course students will work on collaborative programming projects, using programming tools and methodologies presented in class. The course will include topics from software engineering, graphics, and functional programming, but will be focused on student-directed, faculty-mentored programming projects. Students will program continuously and read, run, and criticize on another's programs. These programs may be written for any application area and may include utilities, games, artworks, cognitive models, and environmental or social simulations. We will develop the ability to critique programs from a variety of perspectives ranging from complexity theory to aesthetics. Prerequisite: one programming course (in any language). This course satisfies Division I distribution requirements. PRJ, PRS, QUA

Instructor Narrative

Alec's performance in this class was mixed. He was one of the most proficient and engaged programmers in the class, and he clearly mastered the material that we covered while working on a range of personal projects including an AI search algorithm, a termite colony simulator, and a simulator for the "prisoner's dilemma" economic game. Alec gave an excellent presentation to the class on ways to combine two of the programming tools that we were using, and he was exceptionally helpful in class discussions and in collaborative coding sessions. On the other hand he missed several classes at the end of the semester, he failed to give a final presentation, and he provided insufficient documentation with his final portfolio. Nonetheless, Alec clearly demonstrated strong computer science skills in this course and I believe that he learned a fair amount through his participation. I look forward to seeing his future work.

5.6 Genetic Programming (ta eval only)

Course: CS-254 Prof: Lee Spector Term: 2011 Fall, 2012 Fall Lang: Clojure

Self Eval

This was a course I didn't put the time into properly the first time around. I was very involved in everything else I was taking/working on and this one fell to the back burner for too long. In addition, though my final project wasn't undoable I made too much out of it. By the end of the semester I had gone half insane and was determined the only way I could get things working was using cascalog (a prolog-like clojure library built for hadoop clusters.)

That said, I spent a fair amount of time creating a few tiny DSL's for my creatures to evolve. I wouldn't say I learned a whole lot, but I got a better understanding of GA and clojure data structures.

TAing the course went much better. I was more involved and realized just how difficult I made things myself last time around. Still, GP isn't my field and I still find it a bit like black magic. I was able to help students debug and design their software, but still don't have a great sense on how they can improve things.

Evaluations

Student

No Eval: I never completed my final project.

Teaching Assistant

Alec was one of two teaching assistants for this upper-level computer science course, and he was the one with more relevant prior experience. His prior experience with "functional-style" programming and with the Clojure programming language, which we used in this course, was particularly relevant and helpful. Alec's teaching assistant duties included attending class regularly, helping students during in-class coding sessions, helping to administer in-class presentations, answering student questions during office hours and by email, and helping to organize submitted student work.

Alec's performance in all aspects of his work for this course was excellent. His strong programming skills made him particularly valuable as a resource outside of class, and he was reliable and conscientious in all of his duties. I think that this was also significant learning experience for Alec, both because of the teaching experience that he gained and because of his deeper immersion in materials covered in the course. The course was substantially improved by his participation and it was a pleasure to work with him in this capacity.

5.7 OpenGL Independent Study

Course: Directed Study in OpenGL (IS: CS-200-6) Prof: Paul Dickson Term: 2012 Spring Lang: python 2.7 Libs: pyglet

Self Eval

I read through a couple of OpenGL textbooks and many tutorials. In addition to reinforcing and relearning the little OpenGL I had picked up over the years, I got a better sense of the pipeline, how to use buffers, and shaders. Most of my time was spent converting tutorial C code to python, and redesigning it to be more pythonic and modular. By the end of the course I had written a pyglet program to load and display collada mesh data and shaders.

Evaluation

Alec Goebel's goal with this independent study was to increase his knowledge of and comfort using OpenGL. To achieve this goal he decided to work through a couple of OpenGL text books doing examples until he felt comfortable to build an OpenGL renderer that could hand image files generated by standard software like Blender. Alec ran into a lot of problems with this study ranging from poorly written textbooks and online references to software inconsistencies between the C++ examples he was looking at and the Python code he was writing. While Alec did not find total success with the renderers from standard file formats he was hoping for he did increase his overall knowledge of OpenGL. As Alec is now comfortable working in OpenGL to render any type of image he sees fit, the independent study was a success.

5.8 Video Game Design 2

:Course: CS-360
:Term: 2010 Fall
:Prof: Paul Dickson
:Lang: Javascript
:Tools: Cabana

Self Evaluation

I have mixed feelings about the results of this course. On one hand, I am mostly proud of what I accomplished. I was able to create a variety of libraries to work around Cabana's limitations. In Cabana itself I created a variety of fun widgets that even the developers took notice of. I also created decent Javascript game engine and level generator framework. Most importantly, I improved upon how I go about making games from scratch; I started with a decent modular framework in mind instead of just hacking it bit by bit.

However, this was my worst performance as a group member. There were a few contributing factors. First, I was completely out of practice with school work since this was the first semester back in 2 1/2 years. Second, my personal life had a huge hiccup halfway through the semester. Third, I was again trying to challenge myself which meant I took on too much and made things too complicated for the others. Worst of all was just my hubris. There was no question I had more experience with game development than my group, but instead of working with them I decided to "it would be faster to do it myself."

There comes a point where my ambition is more of a hindrance to myself and others. I have since gotten a bit better at trusting others with my code and working with others at their pace. Still, it is something I am constantly working on.

Evaluation: CS-360 (2010F)
Video Game Design 2: Continuing Building Video Games from Scratch

Description

This course is the continuation of CS260 Video Game Design. It focuses on video game theory and active development of games whereas CS260 focused on the introduction of tools. Students will use their knowledge to lead groups of current CS260 students while actively participating in the development process. This course will also focus on development of games for the iPhone/iTouch platform. Prerequisite: Students are required to have taken CS260 Video Game Design.

Instructor Narrative

Alec's performance in this course was generally good. His work on the individual assignments at the beginning of the semester was superb and he managed to push the software we were using beyond the scope expected by its designers. Alec's work on the group final project varied.

Alec was the group member who built the game engine and essentially built the game. Alec's programming of the game was enough beyond his group mates that this seemed like a reasonable plan. Unfortunately when Alec missed a couple of weeks of class it threw his group into disarray and left his group mates with nothing to work on until he reappeared. If Alec had been willing to trust his group mates with more of the project and had kept in communication a little better the project would have run a lot smoother.

One crucial detail is that in the game post-mortem Alec and his group recognized this issue. Alec was the first to say that it was his actions that caused the greatest issues with group dynamics. As a group they were able to recognize where their group dynamics failed and were able to discuss possible ways of fixing them for future group work. This ability to recognize and address issues is crucial in game design and other fields and is a big success for Alec.

Ultimately they were able to build a successful game and thus succeeded in the course. Alec did well but could have been the strongest student in the course and needs to work on his communication.

5.9 Improvisor's Laboratory

Course: HACU-251 Prof: Martin Elrich Term: 2011 Fall Lang: Chuck

Self Eval

This course was a unique experience. It became quickly apparent that I didn't have the same knowledge background as my peers and often felt lost. Also, as a "math" musician, the sort of "just feel it" mentality was one that gave me trouble. I thought about what I was doing instead of just going with it.

That said, as an HCI guy it's always good to be exposed to a new way of thinking. Learning how musicians just go with things has changed my thoughts on digital instrument design significantly, opting for more of an intuitive interface than explicit controls.

Still, "you have to dig it to dig it," and clearly I don't.

Evaluation

Alec took a very pro-active role in this ensemble. He made cogent contributions to the in class rehearsal process. He gave engaged critiques to his colleagues. Alec has a range of skills as an instrumentalist, and showed growth during the semester as an improviser in a soloistic and collective context.

Alec's writing on music showed a comparable level of engagement. He gave a compelling narrative of the growth of electronic music in current contexts. I look forward to his ongoing listening and reading in related areas of music outside his own area of expertise. He is ready to make a solid expansion of his musical world.

No Evaluations

5.10 Mobile Computing

Course: CS-227 Prof: Paul Dickson Term: 2011 Spring Lang: Java Libs: Android

Self Eval

Another class where my ambition got the best of me. Having used cabana previously, I knew I wasn't going to use it again if I could help it. Instead I taught myself how to program Android. Over the course of the term I got familiar with the software's structure, how to take advantage of layouts, and more.

However, I never quite settled on a final project. I bounced from a game, to a webcomic reader, to a Wings Over Amherst ordering store, and finally a Scrabble Helper. Each project turned out to be too large for the time I had left in the course so I tried to switch to something easier, never finding it. In the end, I should have just partially made one of my ideas and called it a prototype.

Still, I got much better at storyboarding and prototyping for touch devices. I created a number of reusable and skinnable widgets, as well as different ways to lay out various menus on the device. Since learning how to program Android and getting better with touch design was my primary goal, the course was not a waste.

Evaluation

No Eval: I didn't turn in a final project.

5.11 Developing TheHub

Course	CS-212
Professors	Jeff Butera
	Chris Perry
Semester	2011 Fall
Lang	Javascript
	Python
Libs	web.py
	jQuery.js
	Spine.js

Self Eval

Having previously spent a lot of time learning about python web frameworks, I spent most of my time learning about the clientside. Mostly I was using Spine as an ORM around JSON data passed from the server. Using Spine and jQuery I was able to easily map the data into existing DOM elements.

In addition, I analyzed and prototyped responsive widget design ideas (e.g. adding sensible defaults to checkbox filters).

Evaluation

No Eval: The actual hub team and I were not a good match. While I taught myself a lot, I failed to meet the course goals according to the instructor.

5.12 Application-Oriented Databases

:Course: CS-154
:Term: 2011 Spring
:Prof: Jeff Butera
:Langs: SQL
Javascript
:DBs: MySQL
PostGres
CouchDB
MongoDB

Self Evaluation

This course was not what I expected when I signed up. I was hoping to get an actual database course like I would have at WPI. The concepts covered in class were the ones I had known for years from working with the web.

Instead, I spent a lot of time on directed study. I taught myself more about views and stored procedures. I ran through some of the "SQL for Smarties" series and learned a couple of different ways to represent graphs and trees. At the beginning of the semester I spent a fair amount of time getting better at map-reduce in CouchDB and MongoDB as a NoSQL alternative.

Where I failed was the final project. Instead of just creating and documenting an arbitrary database I tried to challenge myself. I believe I bounced between trying to build a rules engine in SQL for twitter users, a trip store with a SPARQL implementation, and a dynamic hierarchical tag system which could deduce relations. The rules engine possibly got the farthest but still nowhere near working.

In hindsight, I should have just turned in one of the many databases I created for my other projects over the semester. I had created at least two with the lecture recording software, a few to hold arbitrary data as I created my views and stored procedures. Sadly, I didn't get out of my own way and demanded too much from myself for a final project.

Evaluation: CS-154 (2010F)
Application-Oriented Database Theory

Course Description

Application-Oriented Database Theory: Databases are ubiquitous in society – there are few websites, applications or devices you encounter on a daily basis that aren't tied to a database. This course will examine the history and theory of databases starting with E.F. Codd as "the father of relational databases" from his groundbreaking work published in 1970 through the current state with the "usual suspects" (mysql, oracle, MSSQL) coupled with some less known players (Unidata/Universe, Cache', OpenQM, jBase...) This is not a course in mysql. This is a course where we'll play and plan on getting our hands dirty. Students will be expected to do reading/research on assigned topics and presenting them in class for discussion. Likewise, students should be prepared to use some of these databases and objectively look at their merits as well as shortcomings. PRJ, PRS, QUA

Instructor Narrative

Alec's work in this course was very good. Unlike many in the course, he arrived with a strong background in databases so Alec spent more of his time working independently, coming to me with questions that were more advanced than most in the course might've been able to grasp.

In short, Alec's work was excellent. He completed the work requested and went much further than others in an attempt to investigate databases other than mysql for some of his personal interests.

Chapter 6

WPI

Academic Structure

Every school year is broken into four seven-week terms, and students take on average three courses a semester or overload with four. Each course covers the same material as a semester or trimester at most other colleges. Due to the intense nature of the BS program, students are expected to declare their major by November of their freshman year.

Courses meet four days a week for 50 minutes a day, with homework averaging about twice the amount of time spent in class. Many classes also had an additional lab component, including most Computer Science courses.

WPI has a unique grading system to encourage students to take courses outside of their comfort zone. Students can earn an A, B, C, or No Record (NR); nothing below a C appears on the student's transcript and so there is no record of the student taking the course.

The course numbering is standard with the first digit referring to the year *most* students will take the course (e.g. 1000 courses are taken freshman year.) However courses only have recommendations, not requirements, so they can be taken at any time. In addition any course ending in an "X" is experimental, including when a course is first introduced.

6.1 Technical Game Development 1

:Course: IMGD-3000
:Term: 2007 Fall-B
:Prof: Rob Lindeman
:Lang: C++
:Lib: C4 (3D Game Engine)
:Tools: Visual Studio 8

Course Description

This course teaches technical Computer Science aspects of game development, with the focus of the course on low-level programming of a computer games. Topics include 2D and 3D game engines, simulation-type games, analog and digital controllers and other forms of tertiary input. Students will implement games or parts of games, including exploration of graphics, sound, and music as it affects game implementation.

Self Evaluation

In this course we worked in teams of four: two from Tech Game Dev 1, two from Artistic Game Dev 1. We were programming Windows games using the C4 engine. For our project, my group created a 2-1/2d sidescroller. Our project was an overall success, winning "best tech" for the class.

The largest piece of my work was a camera system. The games camera could easily be set from triggers in the level editor. In addition, there were trigger regions which splined the camera according to the player's position. This made it easy for the artists to get the exact camera angle they wanted for each section of a level.

This project was the first time I noticed how much effort I put into designing interfaces versus game design itself. Creating useful tools for the artists was much more rewarding than hard coding something that worked. It was the final revelation that made me leave game design for the more general concentration of HCI.

Grade: A

6.2 Machinima

:Course: IMGD-200X
:Term: Spring 2007-D
:Prof: Joseph Farbrook

Course Description

This course will address the cinemagraphic and narrative techniques involved with film making using video-game technology. Creation and development of characters, environments, and narrative structures will be explored. Using commercial game engines and audio/video editors, students will write, design, and produce complete animated movies. Industrial and artistic applications of this film making technique are discussed as well as how Machinima is contextualized in the history of film animation and visualization.

Self Evaluation

This was my first major group/project work. I lived and breathed the final project for this course and it paid off.

For most of the course we were just making movies. For the project, I was part of a "super group"; we joined two groups of four and did twice the work. Though we shared responsibilities, I was one of the driving forces and acted as producer/director/editor.

After the four weeks of work, "Waking" was done. The end result was a 4 part series, about 31 mins long. We used four game engines, Maya, and real life. Coordinating eight people, different video files/formats and everything else was a surprisingly tough, though rewarding, undertaking. In the end, we exceeded our expectations and got second place in a "Machinima Premiere" contest.

Grade: A

6.3 Human Computer Interaction

:Course: CS-3041
:Term: 2007 Fall-A
:Prof: David Brown
:Lang: None
:Libs: VB.Net

Course Description

This course develops in the student an understanding of the nature and importance of problems concerning the efficiency and effectiveness of human interaction with computer-based systems. Topics include the design and evaluation of interactive computer systems, basic psychological considerations of interaction, interactive language design, interactive hardware design, and special input/output techniques.

Self Evaluation

"I am not normal". The litany we said everyday in class to remind us that the way tech people think is not how end users do. While the course itself was not wonderful, this brief look into HCI made me leave game design in the long run.

The course involved no programming. Instead, we analyzed various UIs, ran usability studies, and made a few interfaces. Emphasis was put on the fact that HCI relies on standards and intuition on some level. The final project for this course was working with a group to design a language free "pen pal" application.

Fun tidbit about this course. Although I got a fair amount out of it, I didn't think it was quite what I was interested in. For a long time (until speaking with Paul) I insisted on saying I wanted to study CHI, distinguishing my interests from what this course represented.

Grade: B

6.4 Storytelling in IMGD

:Course: IMGD-1002
:Term: 2007 Spring-D
:Prof: Dean O'Donnell

Course Description

This course explores different types of story within gaming and other interactive media. It delineates between linear, branching, and emergent storytelling, identifies hybrids, and finds new modes of making compelling narrative. A variety of games are discussed, including early text-based adventures, role- playing games, shooters, and strategy games. Students will construct characters, situations, and narratives through game play and scripted cut scenes. Students will explore and use visual storytelling techniques.

Self Evaluation

This course could best be described as eccentric but effective study of interactive storytelling. We learned about improv theater, Dungeons and Dragons, and ran an ARG (Alternate Reality Game). Still, the skills from these activities have been surprisingly useful. The techniques of guiding players and dealing with new situations on the fly has a lot in common with teaching.

Grade: B

6.5 Accelerated Intro to Program Design

:Course: CS-1102
:Term: 2006 Fall-A
:Prof: Michael Gennert
:Lang: Scheme
:Lib: Dr. Scheme (Big Pack)

Course Description

This course provides an accelerated introduction to design and implementation of functional programs. The course presents the material from CS 1101 at a fast pace (so students can migrate their programming experience to functional languages), then covers several advanced topics in functional programming (potential topics include macros, lazy programming with streams, and programming with higher-order functions). Students will be expected to design, implement, and debug programs in a functional programming language.

Self Evaluation

This was the course that first opened my eyes to the beauty of functional programming. We spent most of our time practicing how to use recursive algorithms. For my final project I created a graphics simulation DSL.

Grade: A

6.6 Object-Oriented Design Concepts

:Course: CS-2102
:Term: 2006 Fall-B
:Prof: George Heineman
:Lang: Java

Course Description

This course introduces students to an object-oriented model of programming. Building from the design methodology covered in CS 1101/CS 1102, this course shows how programs can be decomposed into classes and objects. By emphasizing design, this course shows how to implement small defect-free programs and evaluate design decisions to select an optimal design under specific assumptions. Topics include inheritance, exceptions, interface, design by contract, basic design patterns, and reuse. Students will be expected to design, implement, and debug object-oriented programs composed of multiple classes and over a variety of data structures.

Self Evaluation

I had already learned Java in high school for the AP, so most of this course was a review. Still, I reinforced my understanding of basic data structures and OOP. Every assignment we did required the creation of unit tests. By the end of the term, I was much better at TDD.

Grade: A

6.7 Systems Programming Concepts

:Course: CS-2303
:Term: 2007 Spring-C
:Prof: Michael Ciraldi
:Lang: C/C++

Course Description

This course introduces students to a model of programming where the programming language exposes details of how the hardware stores and executes software. Building from the design concepts covered in CS 2102, this course covers manual memory management, pointers, the machine stack, and input/output mechanisms. The course will involve large-scale programming exercises and will be designed to help students confront issues of safe programming with system-level constructs. The course will cover several tools that assist programmers in these tasks. Students will be expected to design, implement, and debug programs in C++ and C.

Self Evaluation

Although I had touched C++ before, this was the first time I actually used the language. We spent most of the course programming in C, learning about memory allocation, and implementing in-place data structures. This course showed me how much I dislike low level languages.

Grade: A

6.8 Intro to Artificial Intelligence

:Course: CS-4341
:Term: 2007 Fall-B
:Prof: David Brown
:Lang: Scheme

Course Description

This course studies the problem of making computers act in ways which we call intelligent. Topics include major theories, tools and applications of artificial intelligence, aspects of knowledge representation, searching and planning, and natural language understanding.

Students will be expected to complete projects which express problems that require search in state spaces, and to propose appropriate methods for solving the problems.

Self Evaluation

When I entered WPI, I thought I would go into video game AI. As I learned more about game AI, my interest waned.

This course was quite useful though. We had to implement a wide range of topics including DF/BF/A* searches, a forward chaining rules, and genetic algorithms. I still find myself consulting the text books from this course.

Although I did well on the code portions, that was not enough to get a good grade. Our code was not run. Instead, we were graded on our debug statements and accompanying essay. Between this and the exams which covered the readings (not the homeworks or lectures) my grade went down.

Grade: C

Chapter 7

Other

TODO

Part III

Sample Works

Chapter 8

Programming Languages

TODO: Everything about all the languages.

TOOD: Table

8.1 Python

TODO: find examples

8.2 Clojure

TODO

8.3 Javascript

Listing 8.1: A fake way to do tail recursion with setTimeout
web/jsgui/autoscroll.html

```
77
78 function tail(fn) {
79     args = [];
80     for(var i=1; i<arguments.length; i++)
81         args.push(arguments[i]);
82     setTimeout( function() { fn.apply(this,args) }, 100);
83 }
84
85 function step() {
86     terminal.log("I'm␣running␣forever");
87     if(running===true)
88         tail(step);
```

Listing 8.2: Extending Array.prototype to add useful functions
web/jsgui/bits/utils.js

```
55 function isArray(arr) {
56     return !arr.propertyIsEnumerable('length') && typeof(arr) === 'object' && ...
           typeof(arr.length) === 'number';
57 }
58
59 Array.prototype.max = function max( ){
60     return Math.max.apply( Math, this );
61 };
62
63 Array.prototype.min = function min( ){
64     return Math.min.apply( Math, this );
65 };
66
67 Array.prototype.copy = function copy() {
68     return this.concat([]);
69 }
70
71 Array.prototype.remove = function(from, to) {
72   var rest = this.slice((to || from) + 1 || this.length);
73   this.length = from < 0 ? this.length + from : from;
74   return this.push.apply(this, rest);
75 };
```

Listing 8.3: Extending a function to be able to return it's signature
web/jsgui/bits/utils.js

```
77 Function.prototype.signature = function signature() {
78     var matches = this.toString().match(/^function (.*?)(\x28.*?\x29).*/);
79     var name = matches[1];
80     var args = matches[2];
81     if (name === "")
82         name="function";
83     return name+args
84 }
```

8.4 Java

Though not my favorite language, I have been programming Java the longest. Since it is my father's primary language, I first picked it up before high school. That said, I didn't really "get it" until the AP Computer Science exam in 2004. After that, it was my language of choice for a long time, and I quickly tried to master everything OO.

These days, I mainly use it for Android and interop-ing with other languages like Clojure. Still, on occassion I break into it for knocking out a quick universal program, like creating a birthday card applet.

I have a solid foundation for Java, OO Design, and AWT/Swing UI's which has helped me enormously for learning other languages. Java was the first language I knew as a computer scientist, not a hobbyist.

Listing 8.4: Sample Java code from a Game of Life Applet
other/java-bday/src/com/swordandspade/bday/GameOfLifeGrid.java

```
1  package com.swordandspade.bday;
2
3  import java.awt.*;
4  import java.awt.geom.*;
5  import java.awt.image.*;
6
7  public class GameOfLifeGrid implements BufferedImageOp {
8      private final int transparent = new Color(0,0,0,0).getRGB();
9
10     private volatile boolean[][] data;
11     private volatile boolean[][] next;
12     private int xSize;
13     private int ySize;
14
15     private final int cellSize = 4;
16
17     private boolean simulation = false;
18     private boolean remove = false;
19
20     public GameOfLifeGrid(int xSize, int ySize) {
21         this.data = new boolean[xSize/cellSize][ySize/cellSize];
22         this.next = new boolean[xSize/cellSize][ySize/cellSize];
23         this.xSize = xSize/cellSize;
24         this.ySize = ySize/cellSize;
25     }
26
27
28     public void step() {
29         if (simulation) {
30             for(int y=0; y<ySize; y++) {
31                 for(int x=0; x<xSize; x++) {
32                     int n = getNeighbors(x,y);
33
34                     if(data[x][y]) {
35                         if (n<2 || n>3) next[x][y] = false;
36                         else next[x][y] = true;
37                     }
38                     else {
39                         if (n==3) next[x][y] = true;
40                         else next[x][y] = false;
```

```
41                      }
42                  }
43              }
44              boolean[][] tmp = data;
45              this.data = next;
46              this.next = tmp;
47          }
48      }
49
50      public int getNeighbors(int x,int y) {
51          int total = 0;
52          for(int i=-1; i<2; i++) {
53              for(int j=-1; j<2; j++) {
54                  if (i==0 && j==0) continue;
55                  int nx = x+i;
56                  int ny = y+j;
57
58                  if (nx<0) nx+=xSize;        if (ny<0) ny+=ySize;
59                  if (nx>=xSize)  nx-=xSize; if (ny>=ySize) ny-=ySize;
60
61                  if (data[nx][ny]) {
62                      total++;
63                  }
64              }
65          }
66          return total;
67      }
68
69      public void setMode(int x, int y) {
70          remove = data[x/cellSize][y/cellSize];
71          addCell(x,y);
72      }
73
74      public void addCell(int x, int y) {
75          x/=cellSize;
76          y/=cellSize;
77          if (remove)
78              data[x][y]=false;
79          else
80              data[x][y]=true;
81      }
82
83
84      public final BufferedImage filter(BufferedImage src, BufferedImage dst) {
85          if (dst==null) dst=createCompatibleDestImage(src,null);
86
87          for(int y=0; y<src.getHeight(); y++) {
88              for(int x=0; x<src.getWidth(); x++) {
89                  int srcPixel = src.getRGB(x, y);
90                  int c;
91                  if (data[x/cellSize][y/cellSize])
92                      c = srcPixel;
93                  else
94                      c = transparent;
95
96                  dst.setRGB(x, y, c);
```

```
97                  }
98              }
99
100             return dst;
101         }
102
103         public BufferedImage createCompatibleDestImage(BufferedImage src, ColorModel ...
                dstCM) {
104             BufferedImage image;
105             if (dstCM==null) dstCM = src.getColorModel();
106
107             int width  = src.getWidth();
108             int height = src.getHeight();
109             image = new BufferedImage(dstCM,
110                     dstCM.createCompatibleWritableRaster(width, height),
111                     dstCM.isAlphaPremultiplied(), null);
112             return image;
113         }
114
115         public final Rectangle2D getBounds2D(BufferedImage src) {
116             return src.getRaster().getBounds();
117         }
118
119         public final Point2D getPoint2D(Point2D srcPt, Point2D dstPt) {
120             if (dstPt == null) dstPt = new Point2D.Float();
121             dstPt.setLocation(srcPt.getX(), srcPt.getY());
122             return dstPt;
123         }
124
125         public void toggleSimulation() {
126             simulation = !simulation;
127         }
128
129         public final RenderingHints getRenderingHints() { return null; }
130
131         public boolean isRunning() {
132             return simulation;
133         }
134
135         public void placeRandom() {
136             for(int i=0; i<400; i++) {
137                 int x = (int)(Math.random()*xSize);
138                 int y = (int)(Math.random()*ySize);
139                 data[x][y]=true;
140             }
141         }
142 }
```

8.5 C/C++

In general, system level programming is my weakest area. As much as I have delved into functional style programming I have avoided memory allocation. As a result, many of my elegant programming interfaces can run horrifically slow. That said, I am certainly comfortable enough in C/C++ to program in it or modify existing code as I need to.

I first picked it up in my early days of high school and then didn't touch it again until college. Once I got the hang of pointers at WPI, I started regularly using it for Game Development, both in class and with a GDC development team. I used it for a wide range of game-engines[1] before I stopped being a game developer. Most recently, I needed to use it for the lecture recording project to get the capturing and processing code to play nice.

Listing 8.5: Binary Tree in C (header)
wpi/sysprog/assn6/tnode.h

```
1  /** Struct to define a stack; each entry can hold a pointer to anything.
2   */
3  struct tnode {
4    char *data; // Pointer to data (c-style string)
5    tnode *left; // Pointer to location of left tnode
6    tnode *right; // Pointer to location of right tnode
7  };
8
9  typedef struct tnode Tnode;
10
11 /** Add a tnode to the current list of tnodes
12     @param current_tnode the tnode to add new value to
13     @param value the string to add as data to the next node
14     @return pointer to the current node (if node was null, the new node)
15 **/
16 Tnode* add_tnode(Tnode *current_tnode, char* value);
17
18 /** Print the tnode INORDER (left, self, right)
19     @param current_tnode the tnode to add new value to
20 **/
21 void print_tnode(Tnode *current_tnode);
22
23 /** delete all nodes in the tree, left then right
24     @param current_tnode the tnode to add new value to
25 **/
26 void delete_tnode(Tnode *current_tnode);
```

Listing 8.6: Binary Tree in C (implementation)
wpi/sysprog/assn6/tnode.cpp

```
1  #include <stdlib.h>
2  #include <cstdlib>
3  #include <cstdio>
4  #include <string.h>
5  #include "tnode.h"
6
7  Tnode* add_tnode(Tnode *current_tnode, char* value) {
8    // if current_tnode has null for data, allocate and return
```

[1] e.g. C4, OGRE, SDL/OpenGL

```
 9    if(current_tnode == NULL) {
10      Tnode *new_node;
11      new_node = static_cast<Tnode *>(malloc(sizeof(Tnode)));
12
13      if (new_node == NULL) return NULL; // Error—unable to allocate.
14
15      // Fill in the Node.
16      new_node->data = strdup(value);
17      new_node->left = NULL;
18      new_node->right = NULL;
19
20      return new_node;
21    }
22
23    // add to left if <= else add to right
24    if(strcmp(value, (current_tnode->data)) <= 0) {
25      current_tnode->left = add_tnode((current_tnode->left), value);
26    }
27    else {
28       current_tnode->right = add_tnode((current_tnode->right), value);
29    }
30
31    return current_tnode;
32  }
33
34  void print_tnode(Tnode *current_tnode) {
35    if(current_tnode == NULL) return;
36
37    print_tnode((current_tnode->left));
38    printf((current_tnode->data));
39    print_tnode((current_tnode->right));
40  }
41
42  void delete_tnode(Tnode *current_tnode) {
43    if(current_tnode == NULL) return;
44
45    delete_tnode((current_tnode->left));
46    delete_tnode((current_tnode->right));
47    free(current_tnode);
48  }
```

Chapter 9

Pygame Course

9.1 Course

Line Count	22k
Class Time	-
Office Hours	-

In-Class Examples

Homeworks

Tools

Advanced Section

Final Project Sample Code

Chapter 10

Game Development

Chapter 11

Web Development

11.1 Frontend

Tweaks

Listing 11.1: Prototype to make a resumable "back to top" link. When the user clicks a "#top" link, the browser remembers their location. Then, when they click a return button, they jump to where they last were.
web/jsgui/backtoloc/backtoloc.js

```
var SCROLL_POINT = 0;
$(document).ready( function() {
    // grab all back to top links
    $("a[href=#top]").click(function(evt) {
        SCROLL_POINT = $("body").scrollTop();

        $(window).scrollTop(0);
        $("#return").fadeIn("slow");
    });

    // setup return button
    $("#return").click( function(evt) {
        $("#return").hide();
        $("body").scrollTop(SCROLL_POINT);
    });
});
```

Listing 11.2: Prototype to make a tag automatically scroll if the user was already on the bottom. Similar to how most terminals behave.
web/jsgui/autoscroll.html

```
var terminal = {
    lineNumber:0,
    node: document.getElementById("terminal"),
    log: function log(msg) {
        var scroll = this.node.parentNode.scrollTop;
        var wrapHeight = $(this.node.parentNode).height();
        var height = $(this.node).outerHeight();
        var next = document.createElement("div");

        // append another line
        next.innerHTML = "<span class='line'>"+this.lineNumber+":</span>"+
                         "<span class='msg'>"+msg+"</span>";
        $(this.node).append(next);

        // if we don't need to scroll or this is the first time
        newHeight = $(this.node).outerHeight();
        if ( (height<=wrapHeight && newHeight>wrapHeight) ||
             (height-wrapHeight===scroll) ) {
            this.node.parentNode.scrollTop = newHeight-wrapHeight;
        }

        this.lineNumber++;
    }
}

```

```
78  function tail(fn) {
79      args = [];
80      for(var i=1; i<arguments.length; i++)
81          args.push(arguments[i]);
82      setTimeout( function() { fn.apply(this,args) }, 100);
83  }
84
85  function step() {
86      terminal.log("I'm␣running␣forever");
87      if(running===true)
88          tail(step);
89  }
90
91  var running;
92  $(document).ready(function() {
93      $("#control").click(function() {
94          if (running) {
95              $(this).text("Start");
96              running=false;
97          }
98          else {
99              $(this).text("Stop");
100             running=true;
101             step();
102         }
103     });
104 });
```

Listing 11.3: A prototype to create clickable overlays which give instructions when selected. *web/jsgui/help_overlays/help_overlays.js*

```
function drawOverlays() {
    $('*[help]').each(function (i) {
        var overlay = document.createElement('div');
        var h = $(this).outerHeight();
        var w = $(this).outerWidth();
        var off = $(this).offset();
        $(overlay).height(h-4).width(w-4);
        $(overlay).css({top:off.top, left:off.left});
        $(overlay).appendTo("#overlays");
        $(overlay).addClass('help_overlay');
        $(overlay).data("node", this);
    });
}

function showHelp(node) {
    var node = $(node).data('node');

    var div = document.createElement('div');
    $(div).text($(node).attr("help"));
    $(div).prependTo("#instructions");
    $(div).show()
    .animate({opacity: 1.0}, 3000)
    .fadeOut("slow", function() { $(this).remove() });
}

function enableHelp(evt) {
    $('body').click(overlayClick);
    //draw overlays
    drawOverlays();
    $('#enable_help').each( function(i) {
        $(this).unbind('click',enableHelp);
    });
}

function overlayClick(evt) {
    $('body').addClass("get_help");

    if ($(evt.target).hasClass('help_overlay')) {
        showHelp(evt.target);
        disableHelp();
    }
}

function disableHelp() {
    $('#overlays').html("");
    $('body').removeClass("get_help").unbind('click',overlayClick);

    $('#enable_help').each( function(i) {
        $(this).click(enableHelp);
    });
}
```

```
52
53 $('document').ready(function () {
54     $('#enable_help').each( function(i) {
55         $(this).click(enableHelp);
56     });
57 });
```

Listing 11.4: A prototype to recreate Ableton Live's Keymapping.
web/jsgui/keymap/keymap.js

```
151 $(document).ready(function () {
152
153     $(window).keydown( function(evt) {
154         if ( ! $("body").hasClass("keybinding") && evt.keyCode in KEYFUNCTIONS) {
155             evt.preventDefault();
156             var id = KEYFUNCTIONS[evt.keyCode]
157             if (!BUTTONS[id])
158                 $("#"+id).toggleClass("on");
159             else
160                 $("#"+id).addClass("on");
161         }
162         else if (evt.keyCode==27 && $("body").hasClass("keybinding")) {
163             evt.preventDefault();
164             $("body").removeClass("keybinding");
165             $(".on").removeClass("on");
166             render_mapped();
167         }
168         else if ( $("body").hasClass("keybinding") && evt.keyCode != 116) {
169             evt.preventDefault();
170             var selected = $(".selected");
171             if (selected.length>0) {
172                 selected = selected[0];
173                 $(selected).removeClass("selected");
174                 setup_button(evt.keyCode,$(selected).attr("overlays"));
175                 render_mapped();
176             }
177         }
178     });
179
180     $(window).keyup( function(evt) {
181         if ( ! $("body").hasClass("keybinding") && evt.keyCode in KEYFUNCTIONS) {
182             evt.preventDefault();
183             var id = KEYFUNCTIONS[evt.keyCode];
184             if (BUTTONS[id])
185                 $("#"+id).removeClass("on");
186         }
187     });
188 });
```

Listing 11.5: A prototype to give checkboxes a default state if nothing is selected. *web/jsgui/checkbox/checkboxdefaults.js*

```
1 // basic checkbox plugin function template/decorator
2 function checkboxFn(fn, returned) {
3     return function() {
4         var chkbx = this.get(0);
```

```
5           if (chkbx !== undefined && chkbx.tagName == "INPUT" && ...
                chkbx.type.toLowerCase() == "checkbox")
6               return fn(chkbx.name);
7           else
8               return returned;
9       }
10  }
11
12
13  // Checkbox helper functions
14  function getCheckBoxes(name) { return $("input[name="+name+"][type=checkbox]"); }
15  function getChecked(name) { return getCheckBoxes(name).filter(":checked"); }
16  function anyChecked(name) { return (getChecked(name).length) != 0; }
17
18
19  // checkbox jquery functions
20  $.fn.anyChecked = checkboxFn(anyChecked, false);
21  $.fn.getCheckBoxes = checkboxFn(getCheckBoxes, $());
22  $.fn.getChecked = checkboxFn(getChecked, $());
23
24
25  // active as opposed to checked.  If a checkbox is active if nothing
26  // else in it's group is checked AND it has the default attribute
27  $.fn.isActive = function() {
28      var chkbx = this.getCheckBoxes().get(0);
29      if ( chkbx !== undefined) {
30          if (chkbx.checked) return true;
31
32          if (chkbx.name != "" && !anyChecked(chkbx.name)) {
33              return chkbx.getAttribute("default") == "true";
34          }
35      }
36      return false;
37  }
38
39  // fetch all active boxes
40  $.fn.getActive = function() {
41      return this.getCheckBoxes().filter(function() { return $(this).isActive(); });
42  }
43
44  // check all selected boxes
45  $.fn.check = function(state) {
46      state = (state === false) ? false : true; // type correction
47      this.filter("input[type=checkbox]").each( function() {
48          this.checked = state;
49      });
50  }
```

11.2 Backend

Listing 11.6: A pylons helper to create a mini-DSL for typing messages. It was primarily used for logging levels

web/advflash.py

```
from functools import partial

class AdvFlash(object):
    class _Message(object):
        def __init__(self,v): self.text = v
        def __repr__(self): return self.text
        def __str__(self): return self.text
        def __unicode(self): return unicode(self.text)

    def __init__(self, session_key='flash', typ="info", **kw):
        self.key = session_key
        self.type = typ
        self.defaults = kw

    def __call__(self, message, typ=None, **kw):
        from pylons import session
        message = self._Message(message)
        message.type = typ or self.type

        info = self.defaults.copy()
        info.update(kw)

        for k,v in info.items():
            setattr(message,k,v)

        session.setdefault(self.key, []).append(message)
        session.save()

    def __getattr__(self,k):
        "for any given attribute, simply return a partial with the name as the type"
        return partial(self, typ=k)

    def pop_messages(self):
        from pylons import session
        messages = session.pop( self.key, [] )
        session.save()
        return messages
```

Listing 11.7: Example usage of AdvFlash helper.

```
    flash = AdvFlash()
    flash("Hello")
    flash.error("HELP, EVERYTHING IS WRONG")
    flash.info("User record updated")
```

Chapter 12

Other Interfaces

12.1 HCI Course

12.2 Android

12.3 GUI Libraries

Chapter 13

Miscellaneous

13.1 Waking

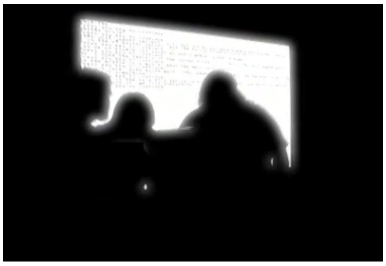
Waking was a 4 part movie (approx 31 mins) created for my machinima course. Filming/production took place over approx 4-5 weeks. We used a variety of games, rendering, and real life.

Synopsis

A sentient multiplayer bot, Vatar is the first true AI in existence. Programmed to know only the game world and servers that govern them, Vatar's perception of existence lasts from the moment of spawn to the final kill of the match. When the creators decide to test the limits of Vatar's capabilities, Vatar must come to grips with the newfound abilities and unfamiliar emotional bonds made.

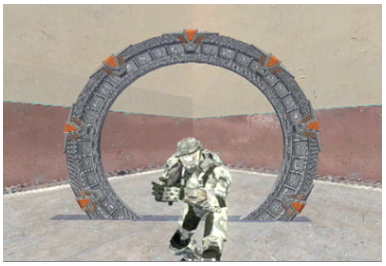
Figure 13.1: Screenshots

(a) Unarmed [Act 1] [Halo 2]



(b) Shadowy Figures [Act 1] [IRL]

(c) Meeting Sigil [Act 1] [UT2K4]



(d) New World [Act 2] [GMod10]

(e) Bathtub Car [Act 2] [GMod10]

(f) Dealloc-ed [Act 2] [UT2K4]

(g) Peace of Flowers [Act 3] [SecondLife]

(h) Server Crashes [Act 3] [SecondLife]

(i) Suiting Up [Act 3] [SecondLife]

(j) Trace Failes [Act 4] [IRL]

(k) Face to Face [Act4] [IRL/SecondLife]

(l) Memories [Act 4] [Maya/UT2K4]

13.2 Gamification

This is the "final" document I created for my independent study on Gamifying Coursework. The document is largely incomplete since it was originally supposed to just be the research/preliminary phase of a Div III. Still, the ideas behind the Topics, the game World, and Quests were largely "finalized" and thought out.

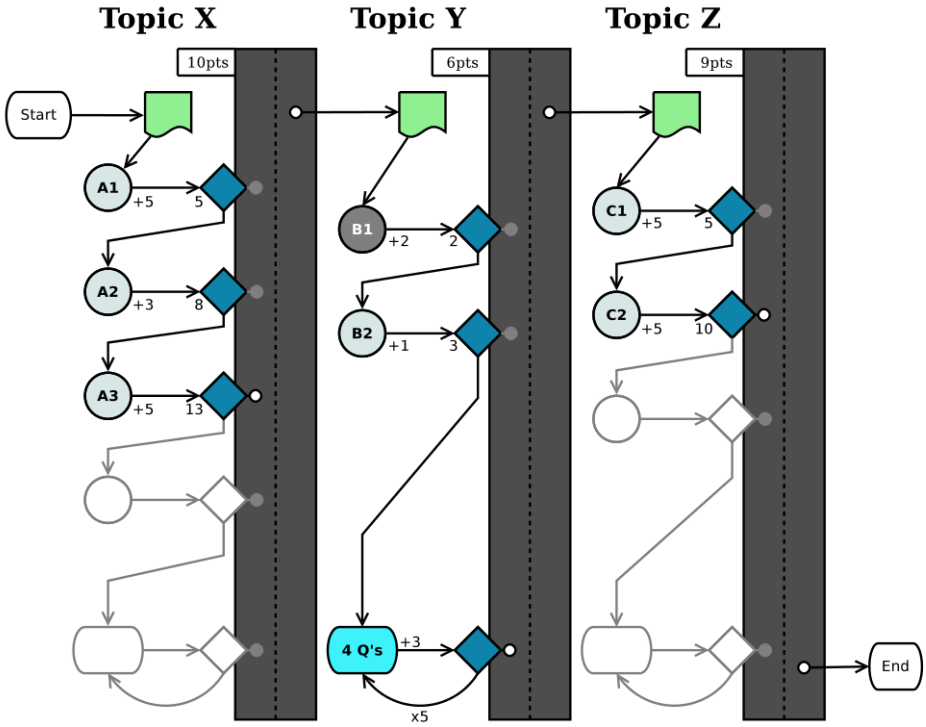


Figure 13.2: An example of a "quest" through multiple topics

Excerpt: Topics Chapter

Chapter 2

Topics

All games require a game world and a gamified course is no exception. However, since a player is navigating through knowledge and facts, creating a simple 2d board is not quite enough. Instead, we must navigate through a more conceptual space. Traditionally, a course has a collection of knowledge resources in the form of notes, excercises, assignments, and readings. A student would simply navigate through these materials by picking one up, reading or working a bit, and putting it down. For a game world to be built out of these resources, something must be defined to represent a location as well as a way to move between locations.

Conditional Topics

- booleans
- "and" statements
- "or" statements
- using both "and" and "or"
- conditionals (if-then)
- branching (else)
- multiple branches (elseif)

Figure 2.1: Example Topics

Like a room in an interactive text adventure or a square on a chessboard, a topic will be a single atomic location in our game world. However, unlike rooms and squares a topic has no real location in space. However, it is an atomic unit of "knowledge". A topic can be thought of as the smallest unit of fact or knowledge attempting to be taught. Think of it as the smallest subsection within a book; a topic conveys one and only one piece of information.

The topics contain everything necessary to teach this one concept. This includes tasks, questions, and information. These will be explained more thoroughly in 2.2.

However, it is necessary to be able to group topics. Figure 2.1 shows a few possible topics for conditionals. Although each topic stands by itself, they are clearly related in subject. Subjects will be defined as any term which a given

Excerpt: Topics Chapter

topic relates to. Therefore, each individual topic can have more than one subject. The topic "using both 'and' and 'or"' has the subjects of "boolean logic", "basic coding", "conditionals", "and", "or", and potentially more.

To avoid having to enter every single related subject, subjects can also relate to other subjects. For example, "and" and "or" could both fall under "basic coding" and "boolean logic." In this way, subjects are essentially a form of hiearchical tagging. However, a topic's subject set refers to all explicitly defined subjects for a topic, not any subjects implied by hierarchy.

2.1 Topic Space

Subjects alone are not enough to define how topics relate to each other. There must be a clear way to get from one topic to another. While grabbing a random topic in a subject could work, more likely than not it will result in topics being completely out of context and order. Therefore topics need a better way to define how they connect to one another. We will call the set of these topics and their edges "topic space".

As with sections in a book, there is sometimes a clear order to teach topics. It is also sometimes the case when it makes just as much sense to teach one of many topics next. For this there is a next-topic edge. Next-topic edges are the strongest edges in our topic space, creating the branching paths throughout topic space.

Just as in any game world, topic space has regions. These regions are defined as a set of topics with the same subject-set and their next-topic edges. These regions are analogous to a chapter in a textbook; regions contain multiple related topics and a path through itself. The set of edges creates a directed a cyclic graph within a region; while the path in a region can branch, it can never repeat.

Next-topic edges can also connect to a topic outside of a region. These connections can be thought as a path between regions. This style of edges can create sidebars or an exit for the region.

However, simply traveling to the next-topic does not always make the most sense. Some regions do not fit in naturally in a specific part. This is especially true for sidebar or enrichment subjects. However, even in this case a certain amount of knowledge is required before they can be started. For this, there is the requires edge. A requires edge points to another topic which must be completed before starting the current one. In general, this is the most common way to connect regions.

Figure 2.2 shows a potential subset of topics space. In this diagram, the letters at the start of the topic name refer to the subject the topic is in; "A1" has a subject of "A", "B3" has a subject of "B", and "AB" has a subject of "A" and "B". The color coding defines the region of the topics. As expected, the next-topic

Excerpt: Topics Chapter

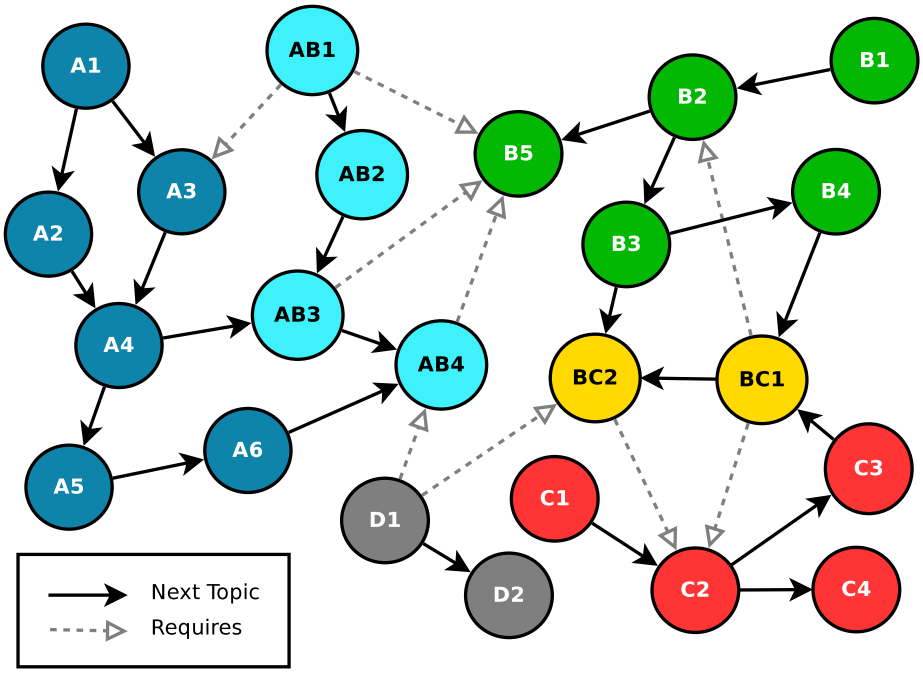


Figure 2.2: Topic Space

edges point to the next topic and requires edges point to the topics that must be completed before the given one. For example, to start topic D1, AB4 and BC2 must be completed.

There are some interesting paths in figure 2.2. For example, a player can take A1 ▷ A2 ▷ A4 or A1 ▷ A3 ▷ A4.

Through next-topic and requires edges, multiple paths through between regions are quickly created. Assuming the player has already completed B5 (the requirement for the entire AB region), the player could get to AB4 in the following ways:

- A1 ▷ A3 . . . AB1 ▷ AB2 ▷ AB3 ▷ AB4
- A1 ▷ A2(A3) ▷ A4 ▷ AB3 ▷ AB4
- A1 ▷ A2(A3) ▷ A4 ▷ A5 ▷ A6 ▷ AB4

These multiple branching paths allow players to create their own way through the structure. Some players will master a subject-set early on and quickly move on to another related region. Others will have trouble grasping the basics of a subject-set and complete many more topics in a given region, but then have to do less examples in a related region.

Excerpt: Topics Chapter

Note:

- Next-topic does not have to point to a topic where all requirements have already been met
- Next-topic can point outside of it's own region creating a sidebar or exit.
- Next-topic has an implicit requires pointing in the opposite direction. However, while all requires edges must be met, only one next-to edge is necessary.
- Requires glues regions together to create a looser sequence/order.
- Any topic which has no requires edges or next-topic edges pointing at it can be considered an entrance point

Expanding Topic Space This is the simplest definition of topic space. Depending on the requirements of the course there could be any number of addtional topic groupings and edges. For examplem, instead of only allowing a requirement edge between two topics, requires could be attached to a region as well. In figure 2.2, instead of explicitly defining a bunch of requires, the region AB could simply require B5.

2.2 Topic Contents

There is no point in defining topic's without defining what goes inside them; what would be the point of a game that only contained empty rooms? Just as a room can have monsters and items, a topic can contains three types of things: information, questions, and tasks.

2.2.1 Information

Information is the simplest piece of a topic: it is the explanation of a topic. Using the textbook analogy, this is the actual text of a section explaining the material covered. It is not, however, anything having to do with explaining or giving a full example/excercise.

2.2.2 Questions

Questions are simple questions having to do with a topic. These would be the equivilent of what might be found at the end of a textbook chapter or on a pop quiz. Most often they will be multiple choice or short answer. Their one requirement is that they must have a clear correct answer. This excludes essay questions or anything which needs to be graded by a person.

Excerpt: Topics Chapter

2.2.3 Tasks

Tasks are the actual meat of a topic; tasks are the actual monsters of this game. They are the primary obstacle that a player must overcome to actually complete a given topic. Just as a player may enter a region in an MMO and kill monsters to grind for points or complete a quest, players will enter topic regions and complete tasks.

In a traditional course, a task would be the combination of an excercise and an example. When a player first sees a task, they are presented with a traditional excercise. This would be no different than what would be found on a problemset type of homework or an example out of a text book. Completing this task would award the player a certain number of points (defined in chapter 4.1.) Each task should be about the size of a single word problem.

In addition to just being an problem, the task is also an example. Using something similar to the universal hint system, a player can get incremental amounts of help with a problem. For each task, there are groups of hints. Each group has an initial question, creating a line of questioning each hint in the group answers more about. Groups of hints can lead into one another by linking to another question. As the player requests a hint, cost of the hint is subtracted from the value of the task lowering what it is worth.

Each hint belongs to a line of questioning. This means that various parts of the excercise can have various groups of hints associated with them. In figure2.3, we can see an example of multiple lines of questioning. There is a very general hint which explains what formulas to use and where to find them, and a second line of questioning about how to get the x and y values out of the arguments.

In this way a task is sort of like working through a problem with the option of asking a virtual TA for help. While hints cannot possibly answer every possible question, they can answer the most common ones.

At any point the player can give up on a task and ask for an explanation. The final value of the task is automatically the lowest possible score, usually a single point. This explanation gives the answer and walks through how it was solved.

Because of the option to get the answer, tasks are not able to be lost. This is because topics are a replacement for course materials like a textbook. Just as you cannot lose at reading a chapter, you should not be able to lose at solving a task.

Excerpt: Topics Chapter

Task: Create a distance function (7pts)

Define a function which takes two points "a" and "b" as arguments and then calculates the distance between them. A point is a tuple containing (x,y).

Hints:

- How do I get started?
  1. *(-1pts)* Use the distance formula: $d = \sqrt{(b_x - a_x)^2 + (b_y - a_y)^2}$
  2. *(-1pts)* You can find the squareroot function in the python math library.
- How do I find b_x?
  1. *(-1pts)* Since the points are tuples, you can get the x and y components out with indexing
  2. *(-2pts)* You can find $b_x$with b[0] and a_y with a[1]

Explanation (1pts): how to write the distance function

```
def distance(a, b):
    return math.sqrt((b[0]-a[0])**2 + (b[1]-a[1])**2)

# you can also unpack the variables
def distance(a, b):
    ax, ay = a
    bx, by = b

    return math.sqrt( (bx - ax)**2 + (by - ay)**2 )
```

Figure 2.3: Example Task

Excerpt: Topics Chapter

Note:

- The value of a hint can be weighted by how many points it subtracts
- the total points of a task must be at least the total cost of the hints plus the value of the explanation.
- the game state remembers
  - which tasks have been completed
  - which hints have been unlocked and they will always be unlocked
- you get no points for completing the same task twice