Computational Social Science 605
Fall 2012 Semester
SYLLABUS

George Mason University
Innovation Hall, Room 320
Thursdays, 4:30-7:10pm

Object-Oriented Modeling in Social Science
or
"How to think like a Computer Scientist"

Maksim Tsvetovat, Ph.D.
Asst. Professor of Computational Social Science
mtsvetov@gmu.edu
Tel. (703) 993–1405

Welcome to CSS 605! This is a course about discovery and invention in the social sciences, so be prepared to learn in a way that may be unlike any previous social science course you may have taken. We hope you will learn from this course as much as we have learned in envisioning, designing, developing, and implementing it for you.

This syllabus covers the main features of CSS 605—or "attributes" and "methods", as you shall soon learn to view and understand these characteristics in the "object paradigm"—while additional information will be posted online as the course develops.

This Syllabus covers course description, assumptions, learning objectives, grading guidelines, material to be covered, and some initial references. Welcome aboard!

Description: This graduate course presents and applies concepts and principles from the object-based modeling paradigm, specifically applied to social science domains. We will learn about modular design, interfaces,

We will start with a crash-course on programming Python -- and continue with fun problems that will "bake your noodle" (to quote The Matrix) -- but teach you to churn out beautiful code.

This course moves at a very fast pace. I expect you to study on your own, and to help each other. This especially applies to novice programmers. Work in pairs and you'll be less likely to suffer.

Prerequisites or co-requisites: CSS 600 Introduction to Computational Social Science, or permission of instructor. This is also a core requirement for candidates in the computational social science Ph.D. program, and an elective course for the CSS Certificate.

This course assumes basic foundations in computing and some programming language (e.g., Visual Basic, Fortran, or SAS code). *Neither* knowledge of object-oriented programming (OOP) or advanced mathematics are required.

On the first day of class, you will get a crash-course tutorial in Python programming. The goal is to both teach you the basics (d'oh!) and understand your abilities so we can tailor the course to be neither "over-the-head" nor too easy for all.

Speaking of "too easy"... Don't count on it :-)

Learning Objectives and Grading

I'm an easy grader. Way too easy. I don't care what you do on a particular test or a homework -- I care that you learn. Learning your way with computers is like learning to play guitar -- it requires practice. My guitar teacher can show me where the notes are on the fret-board, but if I want to sound like Stevie Ray Vaughn I can only sit there for hours and play the blues.

This course will make you practice, a lot. Some of you will find it too easy (if you had programming experience before) -- in which case I'll find new ways to challenge you -- AND I will ask you to teach what you do to others.

Homeworks

There will be something due every week. Mostly incremental steps building from previous week.

If your code works and is not too messy, you'll get an **A** and some comments on how to make it better. If it doesn't, I'll point out ways to fix it. When you do figure it out, I'll give you an **A** anyway (because you learned the material -- isn't that the point?)

Warning 1: you can't get away with procrastination. If you skip a week, you'll be swamped with work and then you'll be *eaten by the grue*[1]. Also, you cannot get away with not finishing an assignment -- the next assignment will build upon it and you'll suffer twice as much.

---

[1] gratuitous references to nerd folklore and ancient computer games are par for the course ;-) If you don't know what a grue is, google it. Or it will eat you.

Warning 2: Bad grades are reserved for people that do not work diligently week to week and do not show improvement. Also, these that don't show up to class for weeks, or don't demonstrate skills they should've learned in kindergarten (like being nice to their peers)

Warning 3: In my other life I run a startup company. I'm not just busy, I'm busy *like hell*. If I didn't comment on your code, it's probably because I think it's OK. I **will** read most of the source code, though. If you think I missed something, ping me again.

Final Project

This will be your crowning achievement and your main grade contribution, and the source of your headaches for the next 15 weeks.

We, as a collective, will undertake building an ambitious model of the world political system, one country at a time. Each of you will build a model of a country -- complete with a simple economy, peasants, natural resources, and an army -- and compete with your classmates for WORLD DOMINATION.

As the end of semester gets closer, we will provide a simulation framework that will integrate all of our simulated states into a single world-system. At the final presentations for the course, we will put all of our simulated states together, and watch in amazement as they battle to the death - or build a peaceful and harmonious world society.

What happens at the final battle is up to you (oh, and we'll hold the final battle in a pub!)

You should work in small groups (3 people max). **Working alone is not an option -- a real software project is never a one-person enterprise and you should learn to collaborate in groups.**

Everyone in the group will receive the same grade.

The grades will not depend on whether your agent takes over the world, but rather on the quality of implementation and your understanding of the modeling paradigms, as well as on the realism of your strategies.

Also, feel free to share information about your strategies to the other groups (strategic lying is par for the course, naturally).

Now here's the kicker:

Shortly after the end of the semester, my new book titled "Hacking Social Complexity" will go to print. I'd like to feature results of the final project in the book.

Everyone involved will be credited in print and online[2] and I'll buy people dinner and drinks when I get my advance[3]

---

[2] if you'd rather not, that's OK too -- let me know in advance

[3] OK, I always buy students dinner at the end of the semester... but this one will be special

Source Control

Source control is a religion. I will teach you to use it daily, and require it of everyone.

It will save your a later.**

The repository for the class is here:

https://github.com/maksim2042/css605_2012

LEARN IT, USE IT!

There is also a Wiki connected to the repository, I will use it to post relevant stuff, and so should you.

Schedule of Study: Calendar, Topics, Assignments

Schedule for this course will change a lot, depending on (a) who is here, (b) how well people are doing. I will adjust topics and assignments to make sure that everyone is challenged but nobody is dying of exhaustion.

Books

How to think like a computer scientist --> excellent NEW textbook. FREE e-book.

Learning Python --> O'Reilly ((use my magic discount code -- "AUTHD" -- 40% off)

learnpython.org and CodeAcademy have excellent online tutorials. If you're new, spend some time with them.

Tools:

Python -- www.python.org .

Get Python 2.7, not 3.<anything>. Why? Python 3 is rather different from Python 2, and not backwards compatible. Also, if you have a Mac, you already have a workable installation.

For added goodies, I like the Enthought Python distribution (EPD-Free).

You can write Python code using any text editor -- but I strongly suggest getting a good one, TextMate or SublimeText on the Mac and NotePadPlus on PC. Of course, the old standby's Emacs and VIM will work on every platform as well.

If you like clicky GUI thingies, check out PyCharm -- it's pretty good if you like working in an IDE.

Some graphics program (OmniGraffle, Visio) (NOT UML software -- it's too smart for its own good and it won't teach us what we need to learn).

Paper and pencil (seriously. Get yourself a Moleskine and a good drafting pencil)

Plagiarism and Group Work

Our software is open-source. Everything done in the final project for the will be under provisions of the BSD license, meaning -- you can use other people's work if (a) you do not pass it off as your own, and (b) give credit and attribution to the author.

In homeworks, the rule of thumb is different -- I want you to do them in pairs, and we will bake them FROM SCRATCH. We will use NO code from anyone else, and will even avoid some of the standard libraries. Why? I want you to learn how it's done -- not how to cut-n-paste some examples from a book and get it to work by some voodoo.