CCTP645: The Poetics of Mobile

Spring 2014
Prof. Garrison LeMasters
Georgetown University

Wednesdays, 3:30PM–6PM

Syllabus 1.4b / Modified 2 February 2014

Contact Information

Prof. Garrison LeMasters

GL75@georgetown.edu
garrison@colophon.org

(202) 271.8784 (cell, please use with discretion)

Office Hours

Wednesday 1PM – 3PM
Thursday 2PM – 3PM

I am generally on campus Tuesday through Friday, from 9:30AM to 4PM, and can meet when it is convenient to you. *It is always best to coordinate with me by email.*

Conceptual Overview

"…I think that what we need, more than a fixed and fixing definition, is a 'poetics,' an open, ever-changing theoretical structure by which to order both our cultural knowledge and our critical procedures." (J-F Lyotard)

"...[M]aybe we could begin to study the implications of both **our *making*** and **our *making sense*** of our culture." (Linda Hutcheon)

Four-fifths of the world's population own a cell phone. According to one analyst, in only four years' time there will be nearly as many smartphones on this planet as literate adults.

In both popular and academic discourse, however, mobile has typically been framed in exclusively utilitarian terms, evoking themes ranging from personal privacy to political agency, from the economics of microtransactions to interoperability in near field communications (NFC). In **CCTP645**, **The Poetics of Mobile**, we will explore how the craft of coding — a practice both *expressive* and *critical* — might play a philosophical role in our lives, serving as an integral component of 21st Century *being*: *Software is an object that we can do our thinking through*.

Working collaboratively across popular object–oriented languages and development platforms, we will make, hack, publish, and break code, developing essential programming skills as we explore the contours of a mobile, cybernetic mode of critical, procedural inquiry that these technologies may offer *Dasein*. But this inquiry is only possible if we rethink our approach to computation.

Hayden White has observed that the realist-empiricist sensibility that informs contemporary historiography is not in itself intrinsic to the study of history, but is rather an accident of time and space. As a discipline, History is a product of the 19th century

Imperial West, launched by the positivism of scholars like Leopold von Ranke, whose attention to primary sources promised to yield the past "*wie es eigentlich gewesen*."

Asks White: **What if history had been born during the years when the Surrealists held sway?**

This epistemic determinism is likely true, too, of code work and its study. The contemporary mania for a science of computers is a predictable response to the conditions of the second half of the Twentieth Century, where rapid innovation and deployment of computation were driven by the twin engines of capital and militarism. Recall, for example, that the Internet of Lil' Bub and Grumpy Cat is a military technology. Contemporary computer science is inextricably bound to a framework fashioned out of MIL–SPEC requirements by post–industrial capitalism. Our task is to think about computation, software, and mobile in ways that elide these contemporary vocabularies.

Practical Overview

In this course, you will design, code and debug a total of seven applications: Six that we work on throughout the semester, and a seventh application that you build and deliver by the end of the term (date to be announced). You may build these applications for deployment on either the iOS or Android platforms, or on both. Additionally, the Corona SDK offers output to Android–powered devices like the Kindle e-reader and the Ouya video game console: Pushing your apps to these devices is reasonable, but will likely require that we revisit the requirements of the assignment (some Kindles, for example, lack the processor speed to achieve the goals of several of our projects).

In addition to building the applications, you are asked to post to the course blog once a week (see related material below). From time to time, you may be asked to prepare for class by reading one of the articles (see related material below).

You will also need to set up a free account on GitHub:

`http://www.github.com`

Finally: This course is now part of a grant provided by the University to encourage faculty to research the use of mobile technologies in the classroom. As such, we will be making use of a badging system that is linked to our blog. Your participation in this process is largely passive (you are awarded badges for various accomplishments). You will, however, have the opportunity to award badges to your colleagues to acknowledge accomplishments big and small, and to celebrate otherwise irrelevant achievements. (Predictably, perhaps, I am interested not only in how these badges can improve our work, but also in how we can undermine their pompous and arbitrary authority over us). In effect, this will require very little effort on your part, but your participation will ensure the success of this research.

If you would like to read more about the University's technology initiative, begin with this letter from the Provost:

`https://itel.georgetown.edu/announcement/`

If you would like to read more about badges, the Mozilla Foundation, supported by a grant from the MacArthur Foundation, has launched this website:

`http://openbadges.org`

Additionally, this is one of the early drafts of a video that I submitted to the University ITEL grant committee:

`https://vimeo.com/78317976`

Material Requirements

(NB that we will review the following requirements in class.)

Required:

- A text–editor (e.g., *Sublime 2*; *TextMate*; *NotePad++; TextWrangler*)
- The Corona SDK (the free package will suffice; requires registration)
- A laptop capable of compiling .lua code via the Corona SDK

Optional:

- An Apple Developer license ($99)
- Access to the Google Developer Console ($25)
- One or more iOS and/or Android devices

Suggested Reference Material

There is no need, *per se*, to purchase any texts for this course, as there is a wealth of freely–available, authoritative reference material online. The codebase of the Corona SDK is in very active development (see, e.g., http://coronalabs.com/blog/), and it changes – sometimes substantially – on a daily basis. It is a commonplace that even incremental modifications to Apple's iOS or Google's Android sends developers scrambling to update their code as well. And it is fairly certain that during an otherwise unremarkable trip to the bookstore in the 2017, you will find a 2014 reference text on Lua – as well as dozens of books about other now–contemporary programming languages – piled haphazardly into the $0.99 Remainder Bin.

Still, we sometimes feel the urge to leaf through something physical, something that offers *resistance*. Further, you may find (as I have) that having reference material in a paper format means less clutter on–screen. With that in mind:

This text is the "official" Lua reference guide, well–received by the community, and published by Lua's authors themselves.

Ierusalimschy, Roberto, Luiz Henrique de Figueiredo, and Waldemar Celes. 2006. *Lua 5.1 Reference Manual*. Rio de Janeiro: Lua.org.

Note that an updated version of this same text is available on the Lua.org website: http://www.lua.org/manual/.

The other Lua-oriented text that you may find useful is less technically–oriented, focusing instead on how to make the best use of some of the features of the language. This is a far more common purchase among Lua coders.

Ierusalimschy, Roberto. 2013. *Programming in Lua*. Rio de Janeiro: Lua.org.

With regard to reference guides to the Corona SDK: While I own several books about Corona, there are no published guides or manuals to this language that I have found useful. As is typical of many of the hastily–assembled texts in the fast–moving tech publishing industry, the books are often poorly written and badly designed. The online Corona SDK API reference,

```
http://docs.coronalabs.com/api/index.html
```

remains the best option. *Nota bene* that it is available as an offline document, too:

```
http://docs.coronalabs.com/CoronaApiDocs.zip
```

We will post links to additional guides and reference materials throughout the semester. If you find something invaluable, please let us know about it via the blog.

Finally, a word about ebooks and reference guides: While I make daily use of various ebook platforms across several devices, I no longer purchase tech-oriented publications in this format. E–books offer many advantages, but publishers seem to spend little effort to ensure that their manuals' contents, newly deployed to this tiny medium, are still legible. A reader typically reaches for a reference manual after some threshold has passed: Out of frustration, for example, as she becomes exasperated with some difficult chunk of code or some mysterious crash; or in a hurry, perhaps, when she realizes her code is due in the morning and she no longer has time for random experimentation. These contexts offer challenges for which contemporary publishers largely fail to account: When the guide itself becomes an *additional* source of frustration, or when the layout on the small, low-res screen makes reading a slow and laborious chore, we are unlikely to solve anything.

Course Blog

You are required to contribute to our class blog on a weekly basis. These contributions *are not graded*: Instead, you will simply receive credit for their completion. Your blog post should appear on the same day every week: You can sign–up for your preferred day of the week during class.

In some university courses, blogs are essentially online assignment repositories that are bolted-on to the class to little effect. Whereas many of us may visit favorite blogs fairly regularly, the course blog gets the same 31 visitors at the same time every week — the night before the class meets.

We want to foster a course blog that is worth your visit. While we welcome thoughtful essays and interesting stories related to our subject matter, these are not required. Instead, we ask that you help your classmates (and classmates in similar straits throughout the world) become better programmers. Share what you have learned during the past week: Brief insights you have had into the process of coding, back-of–the–envelope pseudo–code scribbles (literally! scan it in and post it!), clarifications of confusing documentation that you solved (or failed to solve) through trial and error, screen–caps or videos of the project you are currently developing, examples of favorite User Interfaces you want to emulate (or deconstruct!), or links to exceptional tutorials and other useful online resources.

In terms of format, remember that this is a graduate program, and thoughtful, thorough articulation that sparks conversation is what we strive for. While his site is (justly) revered by the tech community, for example, I would steer clear of adopting the copy+paste style lately demonstrated by blogger John Gruber:

`http://daringfireball.net`

This style points us to interesting bits of information, but has little interest in engaging us in dialogue. Instead, look to development-related blogs like these for styles worth emulating:

Brett Terpstra: `http://brettterpstra.com/`
Michael Yoshitaka Erlewine: `http://mitcho.com/blog/`
Ian Bogost: `http://www.bogost.com/blog/`

Suggested Readings

As a practical matter, meeting the requirements of this course is a significant challenge. After some consideration, I have decided that assigning published scholarship and research reports — in addition to reserving class time for their consideration — is simply unrealistic. But I want to emphasize that our course, while probably technical at any given moment, is effectively an act of what Ian Bogost calls "philosophical carpentry." It is vital, therefore, that we invite other scholarly voices into our conversation.

Below I have included a list of *strongly suggested* readings for the course. *At this point, all of them are optional*. If I believe that we can find time for their consideration, I may occasionally ask you to read one or more of these texts for the following week; alternatively, I may occasionally suggest that some are relevant to that week's in–class and on–line conversation. In any event, should you elect to read any or all of these texts, I believe that you will find that they enhance and deepen the conversation surrounding many of the issues we address.

Cassell, Justine. 1999. "Storytelling as a Nexus of Change in the Relationship Between Gender and Technology: A Feminist Approach to Software Design." In *From Barbie to Mortal Kombat: Gender and Computer Games*, ed. Justine Cassell and Henry Jenkins. Cambridge, Mass: MIT Press.

Dholakia, Nikhilesh and Detlev Zwick. 2003. "Mobile Technologies and Boundaryless Spaces: Slavish Lifestyles, Seductive Meanderings, or Creative Empowerment?" Unpublished MS. Home Oriented Informatics and Telematics. April 6-8.

Filreis, Alan. 2006. "Kinetic Is As Kinetic Does: On the Institutionalization of Digital Poetry." New Media Poetics: Contexts, Technotexts, and

Theories. Adalaide Morris and Thomas Swiss, Eds. Cambridge: MIT.

The complicity of university in the maintenance of conventional poetics / interface with thought. Raises questions that hint that our MOOC madness is going to be short–lived.

Geser, Hans. 2004. "Towards a Sociological Theory of the Mobile Phone," release 3.0, May. In Sociology in Switzerland: Sociology of the Mobile Phone. Online Publications. Zurich.

Graham, Paul. 2003. "Painters and Hackers." Address delivered as guest lecture at Harvard in May. Available at http://www.paulgraham.com/hp.html

Well-known lecture by author, entrepreneur, and Silicon Valley–favorite Paul Graham, one of the founders of startup incubator Y Combinator.

Hayles, N. Katherine. "Metaphoric Networks in Lexia to Perplexia." electronic book review. dtd April 19, 2005. http://www.electronicbookreview.com/thread/firstperson/creole.

nb: Preferred text is via PDF, as host website's outdated code is becoming increasingly problematic.

Memmott, Talan. 2000. Lexia to Perplexia. Via http://collection.eliterature.org/1/works/memmott__lexia_to_perplexia.html

nb: Largely inaccessible to contemporary readers because of increasingly outdated javascript and DHTML dependence. However, hosted exclusively as an electronic text.

"Talan Memmott's *Lexia to Perplexia* is a rich and complex exploration of the relationship between human consciousness and network phenomenology. Alluding to traditions ranging from ancient Greek and Egyptian myth to postmodern literary theory, using a creole of human language and code, Lexia is a work in which the functioning and malfunctioning of the interface itself carries as much meaning as the words and images that compose the text."

> Morris, Adalaide. 2006. "New Media Poetics: As We May Think/How To Write." New Media Poetics: Contexts, Technotexts, and Theories. Adalaide Morris and Thomas Swiss, Eds. Cambridge: MIT.

"New technologies not assimilable in terms of contemporary cultural codes." It isn't merely a matter of writing poems in new media. These categories themselves become suspect, conservative.

> UC Santa Barbara. nd. "The Agrippa Files: An Online Archive of Agrippa, a book of the dead." http://agrippa.english.ucsb.edu/.

Agrippa is (was) (is/was) a 1992 poem / e–poem / performance / commodified artifact / hacked code / scholarly obsession produced by science fiction author William Gibson, one of the founders of the cyberpunk genre (see, e.g., "Burning Chrome," 1982). As conceived by Gibson and two partners, the *Agrippa* project itself was a very cool thing. But what happened after publication was even better.

> Watten, Barrett. 2006. "Poetics in the Expanded Field: Textual, Visual, Digital." New Media Poetics: Contexts, Technotexts, and Theories. Adalaide Morris and Thomas Swiss, Eds. Cambridge: MIT.

Reflects on poetics as a category. Emphasis on Memmott's *Lexia to Perplexia*.

Assessment

In sum, there are seven projects due throughout the semester. The *raw grade* for this course is the average of your weighted project grades (see below).

The *final grade* for this course (*i.e.*, the one that appears on your transcript) is equivalent to this *raw grade* plus or minus up to 5% in order to reflect your attendance, participation in class, fulfillment of responsibilities on the blog, and/or missed conference appointments (with myself or the course TA). Your excellent participation in class, for example, could shift an A– to an A, while several missing blog entries may shift an A– to a B+.

The categories by which this shift can occur are often intangible, and admittedly subjective. *If your participation or attendance threaten to lower your grade, I will alert you to this fact with sufficient time to remedy the situation.* At the same, if you have any questions about attendance, how I perceive your participation in class discussions (on-line and off-), or concerns about your grade for the course in general, please let me know.

Project Grading

Projects will be assessed on a scale from 0 to 10. In practice, these will typically be 7, 8, or 9.

While all of the projects will share portions of a single assessment rubric, each project will usually include a few unique requirements. Both the general rubric and the project-specific rubrics will be posted to the blog.

You will grade your own project before you turn it in, noting what worked and what did not. After I have assessed your work, we will take time to discuss the particulars of your grade. You will have until the end of the semester to revise and resubmit any projects that do not score as well as you would like.

Collaboration

On any project, you are invited to collaborate with up to two of your classmates. Working to design, code, and debug collaboratively introduces a host of new options (and a wealth of new challenges) to each assignment.

My suggestion: collaborate first with your potential cubicle-mate on a small, 2-hour project (we'll build a small library of these during the semester), and see how it goes. You will both have a reasonably clear idea of what to expect at the end of those two hours.

Conditions of Your Collaboration

If you intend to work with others on a project, please read the following material carefully.

- If you collaborate on one of the 7 course projects, each member of your group must explicitly declare their intention to do so — via email — to both myself and the class TA.
- Each new project dissolves any previous goups. If you collaborated on the first project, for example, and you wish to work together on the second project, all members of your group must still declare their intention to do so.
- This notification must occur during the first week of our work on a project.
- Any particular group configuration may collaborate a maximum of three times during the semester. Adding or subtracting a member from a group effectively constitutes a new group, however.
- There is nothing especially complicated about this process, but it is imperative that we be clear about how it will work and what you can expect. Here are two brief examples.

Example

Steve and Bill wish to work together. They collaborate on projects 1, 3, and 4. They then decide that they want to work together on the final project, but it is too late: They've already collaborated three times. After adding Marissa to their group, however, the three are able to declare themselves as a new group and work together on the last project.

Example
Peter, John, Luke, and Matthew wish to work together on projects 4, 5, and 6, but they cannot: Groups can contain no more than 3 members. So they split the difference: For project 4, Peter and John form one group while Luke and Matthew form another. Then, for project 5, Peter and Luke work together, while John and Matthew each labor independently. For project 6, John, Luke, and Matthew form a new group, and Peter is left to fend for himself.

Nota Bene: There is no need to notify us if you intend to collaborate on smaller assignments or incidental code.

Assessing Collaborative Work
When you declare your intention to collaborate on a project, you agree to this stipulation: *For good or for ill, everyone in your group will receive the same project grade, regardless of one member's unforeseen personal circumstances; regardless of conflicting opinions among group members about the final code; regardless of any debate over the putatively vast differences in team-members' efforts, skills, commitment, attitudes, and so on.*

Note that a group cannot be dissolved before the project is delivered. Once you declare your intention to incorporate as a group, I will no longer accept individually–authored versions of that specific assignment from you.

We will use the same criteria to assess collaboratively-built projects as we use to assess independent projects: There will be no separate group rubric.

Final Grade: Project Weighting
Six In–Class Projects: 12.5%
Final Project: 25%

Plagiarism, Attribution, and Other Hackers' Code

This course takes seriously the *ad hoc*, socially-constructed norms that enrich and inform private software authorship today. These norms, rooted in the affordances of digital networks, the ergodic character of the code we run, and the (frequently) generous spirit of development communities worldwide, are often baffling to outsiders, as they may seem to promote intellectual fraud, the theft of IP, dishonesty in claims of authorship, and so forth. But the norms of contemporary code–sharing are vivid reminders that "plagiarism" is not an act of intellectual evil: It is, instead, just another arbitrary technological practice that happens to run counter to important social, cultural, and (most significantly) economic norms.

Early in the semester, we will discuss the ethics of code–swapping, cribbing, snippets, GISTs, hacks, cracks, and decompiles as they relate to amateur hackers like ourselves.

For the purposes of this course, here are two rules of thumb for stealing code, inferred from common practice within the public dev community, together with a corollary:

1. If you can find it online, and it is useful to your project, you may use it;
2. If your code contains others' work, it <u>must</u> give clear attribution, indicate the exact points at which the borrowed code begins and ends, and express gratitude for the contribution (seriously);

And a corollary, specific to our course:

3. Steal infrequently, steal thoughtfully, and always be sure that it is the code that you have written which does most of the work. When you can, do not use others' code verbatim, but learn from it instead and incorporate that lesson into your own work (though the rule of attribution still applies).

Course Schedule (Compact Version)

As it exists, the schedule for this course is long, detailed, and (given the de facto *length of this document) a bit overwhelming. For the purposes of this syllabus, this is a compact version of the schedule. I will share the details of upcoming sessions as it is useful throughout the semester.*

30 January

Express Yourself: Variables, Relationships, Maths, and More

Project One: "Hullo? World?" (A Non-Utility)

Turn to your smartphone for bad, occasionally incoherent advice.

1. Random sentence generator will choose from one of several pre-built sentences to display (stored either in the body of main.lua or in a subsidiary asset file) or may build sentences from fragments and word lists (à la Mad Libs).
2. Innovate: What does the UI for irrational, unpredictable outcomes look like?

6 February

Go On, Git: How To Use GitHub For Almost Everything

Project One: In–Class Workshop. We'll work together to tackle the problems you're having getting your app up and running. Share ideas with your colleagues about expanding, extending, and polishing their applications.

Due: Bring working code fragments to class

13 February

Again and Again (and Again): Control Structures

Project Two: Pocketful of Paramecia (Playing with Simulation)

Using Conway's Game of Life as a point of departure, we'll fashion a virtual–pet–in–a–Petri–dish, and afford the user some power over its well–being.

1. Reproduce Conway's simulation on a 25 by 25 grid;
2. The life cycles of the simulation should be variable, but for now, set the simulation to spawn a new generation every second.
3. Typically, the cells on Conway's grid are merely colored squares. If you have time, consider converting them to images of realistic or cartoonish cells. If you have lots of time, consider animating the cells as sprites (this sounds complicated, but it is actually surprisingly straightforward).
4. Add a layer of user interaction. Give the player the power to add cells or remove them in between generations.
5. If you have time, consider deepening Conway's model by introducing cells with more than two states. For example, instead of "Alive" and "Dead," depict cells that are "Healthy," "Unhealthy," and "Dead." Your algorithm for calculating a cell's specific state will need to be adjusted to reflect this new calculation.

20 February

The *Funk* in Function: Arguments and Overloads

Project Two: In-Class Workshop. We'll work together to tackle the problems you're having getting your app up and running. Share ideas with your colleagues about expanding, extending, and polishing their applications.

27 February

A Motion to Table the Matter; or, "OOPs, I Did It Again"

Project Three: Anxious Moments (Rethinking the Representation of Interval)

Our world teems with *becoming*: Some populations multiply at a staggering rate, while others hurtle towards oblivion with abandon. With this project, we will try to build applications that visualize the systems and cycles that our conscious mind works so hard to repress.

1. Before building this app, you'll need to obtain some useful data around which you will build your visualization. In this case, our desire is to visu-

alize the cumulative impact of one or more processes over time. The precise nature of those processes, or their impact, is up to you.

2. The app makes use of simple timers to trigger animated on-screen events.
3. Once the basic mechanisms of the application are working, embed your visualizations in a physics layer to animate them.
4. If you have extra time: Ensure that the on-screen objects are "reacting" to the tug of the Earth's gravity by polling the motion sensors on your phone and adjusting "up" and "down" accordingly.

6 March

Tables II: The Final Tabling ("This Time Its Personal")

Project Three: In-Class Workshop. We'll work together to tackle the problems you're having getting your app up and running. Share ideas with your colleagues about expanding, extending, and polishing their applications.

20 March

OOPs I Did It Again, Again: More Object-Oriented Code

Project Four: Demonic Photobomb (The Cottingley Girls & the Hidden Real)

With this app, your smartphone's camera will reveal a hidden world of creatures desperate for attention.

1. Build a working camera application (poll the camera and display the results; capture the image at the user's request).
2. Improve the camera's operation (allow for retakes; improve the UI; etc.)
3. Add one or more layers to the captured image, using assets that you provide as PNG files with transparent backgrounds.
4. Add custom filters for effect. Strategic blurs or overexposures, for example, will heighten the effect of the finished image.
5. Send the finished image to the operating system's photo storage.
6. If you have time: One of the keys to this application is not programmatic at all: The app will only be interesting insofar as your daemons, faeries,

angels, or sprites are interesting. Spend time acquiring, commissioning, or creating compelling graphics.

27 March

Advanced Topics and Techniques

Project Four: In–Class Workshop. We'll work together to tackle the problems you're having getting your app up and running. Share ideas with your colleagues about expanding, extending, and polishing their applications.

3 April

Advanced Topics and Techniques

Project Five: A Shuffle of Zombies (Augmenting Whose Reality, Exactly?)

For this project, we'll use your mobile's GPS to build a simple Augmented Reality game that has little interest in the tasks to which Augmented Reality technologies are typically put. Instead, the player tries to avoid wandering groups of zombies that are depicted on-screen as plaguing your neighborhood (the top-down view of which is provided courtesy Google Maps, for example).

1. Build your zombie agents and experiment with suitable AI.
2. Animate your zombies.
3. Add a GPS layer that visualizes the player's position in the world.
4. If you have time: AI is always challenging. Zombies are a favorite among game developers because their motivation is one–dimensional: Move towards the player. They seldom demonstrate strategic insight or a sense of alternative approaches.

10 April

Advanced Topics and Techniques

Project Five: In–Class Workshop. We'll work together to tackle the problems you're having getting your app up and running. Share ideas with your colleagues about expanding, extending, and polishing their applications.

17 April

Towards a Poetics of the Mobile

Project Six: Poetry Machine (Machinic Expression)

Historically, poetry has been integral to human existence: It served as chronicle and prophecy, curriculum and entertainment. During our brief moment, though, poetry is largely irrelevant, and is now the stuff of banal greeting cards, pop divas, and English professors.

This project is technically uncomplicated, but provides a significant opportunity for novel expression and insight. Your application will generate the 21st Century equivalent of "poetry." Self-evidently, the meaning of this term is non-obvious: Your code argues for some particular point of view, presents some specific ideas. Drawing on third-party data streams, user input, randomization, dynamic visuals, variable typefaces, and so on, create a platform for generating dynamic, machine-made poetry.

24 April

Sartor Resartus; or, The Coder Re-Codified

Project Six: In-Class Workshop. We'll work together to tackle the problems you're having getting your app up and running. Share ideas with your colleagues about expanding, extending, and polishing their applications.