Collaborative Artware

MAAD 23655 1 (Winter 2022)

> *The term ‘software art’ acquired a status of an umbrella term for a set of practices approaching software as a cultural construct. Questioning software culturally means not taking for granted, but focusing on, recognising and problematising its distinct aesthetics, poetics and politics captured and performed in its production, dissemination, usage and presence, contexts which software defines and is defined by, histories and cultures built around it, roles it plays and its economies, and various other dimensions. Software, deprived of its alleged ‘transparency’, turns out to be a powerful mechanism, a multifaceted mediator structuring human experience, perception, communication, work and leisure, a layer occupying central positions in the production of digital cultures, politics and economies.* — Olga Goriunova (2007)

Course Description

In this course we'll be working together as an open source arts collective. We'll produce an online app which explores the expressive space between software as a tool and software as art. We'll learn the processes and tools that professional creative technologists use when working together to produce "software art" projects. This is an intermediate level coding course with work being predominantly written in JavaScript (server side and client side). While proficiency in JavaScript is not required, it's recommended that students have a background in basic programming concepts (data types, variables, functions, conditions, loops, etc) as this course will build on those to introduce more intermediate level concepts and programming paradigms.

where:	Cobb Hall 310
when:	Mon, 01:30am - 04:20pm ;
professor:	Nick Briz ;
ta:	netnet ;
email:	nbriz@uchicago.edu ;
office hours:	*by-appointment-only* ;
class website:	http://netart.rocks/uchicago/artware ;

Learning Goals

- Develop a working knowledge of core web technologies and coding languages with a focus on building web applications (as opposed to web sites) with standard web APIs (as opposed to application frameworks like React, Angular or Vue)
- Foundational understanding of what it means to work collaboratively on online projects including a working knowledge of open-source collaboration tools (specifically Git and GitHub) and methodologies (Agile, Scrum, etc)

Class Materials

In order to participate in this course you will need to have a decent computer (desktop or laptop with 8-16GB of ram or more) and a modern **Web browser** like Firefox, Brave, Chrome, Vivaldi or others (**do not use** Internet Explorer or Safari as your primary browser, those are subpar browsers).

I highly recommend that you download more than one, because some of our code may render slightly differently on different browsers. You never know which browser your audience will be using to view/experience your work online and so it's important to test your work across the most common/popular browsers, these days that includes: Chrome, Firefox and Opera as well as Edge (the default browser on Windows) and Safari (the default browser on MacOS).

You will need a **code editor**, I will be using Atom but you could also use Sublime or VSCode (these are all very similar). If you have a preferred code editor that is not one of these, run it by me first for approval.

We'll be using standard **open-source development tools** in order to work collaboratively on our class project. For this you will need to download and install the git command line tool as well as create a free account on GitHub (if you're new to GitHub you should also checkout their Student Developer Pack).

Class Structure

Our primary goal in this course is to create an experimental web based drawing application together as a class. The structure of this course is loosely modeled on the same process a group of creative technologists would follow while working on professional/commercial projects. This means that the specific tasks (assignments) will be collectively discussed but individually self-assigned.

The first couple of weeks this quarter will be predominantly focused on introducing the core concepts and tools required to complete these tasks. This means ensuring everyone has the necessary technical background (familiarity with JavaScript, code editors, browser developer tools and git/GitHub) as well as an understanding of the collaborative open-source development process. The rest of the quarter will be spent working on our assigned tasks. Each student will be working at their own pace and so the number of tasks will vary depending on the student's technical level as well as the difficulty of each particular task.

Evaluation

Loosely following the processes of development methodologies like Agile and Scrum, students will self-assign tasks after ideas and the general direction have been discussed and agreed upon as a group. Students will be expected to move their tasks through the various columns of our project's "kanban" board as they progress through their self-assigned tasks. Each task is worth a different number of points depending on the type of task it is (see assignment section below). Your grade will be directly related to the number of points you acquire (max of 100). I will use this formula: `grade = points * 4 / 100` to determine your grade.

A task (ie. an individualized class assignment) is considered submitted when the student opens a "pull request" (PR) and is not considered complete until it gets "merged" into the class "repository" (repo). Evaluations will be conducted in the form of a "code review" on GitHub. Contributions to the class repo will only be merged if/when the assigned task has met the following criteria:

- the code must accomplish the intended goal (as defined the corresponding GitHub project's task)
- the code must be as error free as possible (this means any console errors should be addressed before being merged)
- the code must conform to the standard JS coding style as well as other conventions established in class (indentation, naming conventions, line lengths, etc)
- any new files created in accomplishing the task must be contained in the proper directory.

Assignments

(aka “tasks”)

The vast majority of the tasks will be “module” contributions, these will be either additional tools, filters, options or functions (the details for each will be explained in class). These tasks will be initially discussed/proposed in a class discussion and, after receiving some feedback on the idea, will be created by the individual artist working on that task. A “meta task” is an idea that goes beyond the constraints of a module, this could mean making modifications to the framework itself like design changes, architectural changes or otherwise extending the functionality beyond what it’s capable of. These tasks will be created by the individual artist but must be pre-approved by the professor before it can be self-assigned.

The remaining two categories are not created by the artist, but rather by the professor. A “bonus task” is something small which might come up in a class conversation (updating a logo, reorganizing files, etc), the points for which will vary depending on the time/difficulty of the task and will be determined upon the creation of the task. The same is true for “bug fix” tasks, which have a minimum of 10 points (but may be more). Any artist in the group can assign any of these tasks to themselves, so long as they haven’t already been taken by another artist.

- `tool module:     20`
- `filter module:   20`
- `options module:  30`
- `function module: 40`
- `bonus task:      ??`
- `bug fix:         10+`
- `meta task:       50-100`