Final Homework
v1.0
Cpt S 489

Choose one (and only 1, there are no options for extra credit) project from the list below. Submit information about which project you have chosen and who you want to work with (if applicable) as well as when you would like to demo during dead week by using the online TODO
All of these options will have been discussed in class, so below are the "short" explanations. You must submit all project code to Blackboard by the time of your demo, if your project requires demoing. Otherwise (for non-demo projects) all code is due Friday of dead week by 11:59 PM (just like previous assignments).

Option 1: Location tracker with data visualization using Google or Bing maps
Group size: Can be a solo project or a group of 2 (no more)
App(s) produced: In-browser client-side app
Requires in-person demo? YES, on MONDAY of dead week
Details:
Produce a JavaScript app that tracks your location and stores it every few seconds. Transfer this application to the phone of each group member and activate it while you walk around campus or town. Make sure the app stores the data so that you can retrieve it later.
Take the location data and use an online maps API, such as Google maps or Bing maps, to render the paths you traveled. The Google static maps API is one option that may suffice. Be prepared to show at least 2 significant length paths rendered in the maps API during the demo, and of course be ready to field questions about technical aspects of the assignment.

Option 2: Remote Camera
Group size: Can be a solo project or a group of 2 (no more)
App(s) produced: Involves creation of a server-side Node.js app as well as a 2 client-side apps
Requires in-person demo? YES, on WEDNESDAY of dead week
Details:
(This will have been discussed in class, so following is the "short" explanation.) Produce a client-side JavaScript app that uses the webcam (documented on MDN under "WebRTC API") to take pictures and upload them to a server. The server-side app will be written in Node.js and will store the latest uploaded image in memory. It will also provide the ability to download this image. Make an additional client-side app that requests the image from the server repeatedly and displays it on the page. You should be able to achieve smooth video when running these apps on a local network. All together this creates the potentially for one machine to serve as a camera and the captured video stream can be viewed from other machines.

Produce a video of the app in action, because you'll need to run this on a home network (the WSU network will likely block it). Show the video during the demo and be prepared to field questions about technical aspects of the assignment.

Option 3: Simple 2D Game with Gamepad Support
Group size: (discuss with professor during project proposal)
App(s) produced: In-browser client-side app
Requires in-person demo? YES, on FRIDAY of dead week
Details:

Because you will get some choice in what type of game to produce, you must get professor approval before starting this project! Don't bite off more than you can chew. This game should be very simple and you shouldn't waste time on details that can become very time consuming. On the flip side, it must be a game that at least demonstrates basic playability. It can't be a "you can move a circle around with an Xbox controller" game, it needs to have some aspect of challenge to it.
Examples:

- Arkanoid
- Asteroids
- Pac-Man (probably too complex, so not recommended)

Implement controls only through the gamepad API. Bring a gamepad to the demo. You can add keyboard/mouse support if you wish, but this is only if you want it for testing purposes it's not worth any credit.
Implement basic gameplay first. There should be a way to win and a way to lose at a minimum. That's really all that's required for credit. You do NOT need, and in fact should avoid (because of limited implementation time), the following:

- scoring
- multiple levels
- instructions
- title screen
- music
- sounds
- (any other similar things)

BUT, you definitely do need a concept of win + lose.
Use canvas for rendering.

Option 4: Sorting with web workers
Group size: Solo project only, no group work allowed for this option
App(s) produced: In-browser client-side app
Requires in-person demo? NO, submit to Blackboard only
Details:
Use web workers to implement two custom sorting implementations and compare multithreaded vs. single-threaded sorting times. Use up to 16 web workers concurrently for the multi-threaded sorts. You'll need to figure out how to ensure that these sorting algorithms don't exceed 16 workers but do indeed use up to 16. You'll also need to figure out how to share/transfer data between the workers. On a 16-core machine, the multithreaded (i.e. web worker) sorts should happen approximately 16 times faster than the single threaded versions.

The HTML page included in the HW .zip file has the setup for the sorting algorithms. You will be implementing your own sorting functions and not using the built-in sorting algorithms. The reason for this is to ensure that you have full control over how the web workers split up the sorting task.

You will need to implement merge sort and quick sort. Recall from Cpt S 223 that both algorithms split the list into 2 pieces and recursively sort each part. You will implement a single-threaded, synchronous version of each to start. These should be pretty much "textbook implementations", in the sense that they don't require any special alterations. Then you'll implement a mult-threaded, asynchronous version of each, wherein each

recursive sort happens on a new web worker, provided you don't already have 16 total workers active. If 16 workers are active, then the two recursive calls are made on the same thread/worker. The HTML page has a timer for the single and multithreaded sorts and on a machine with X cores, where X <= 16, you should see a speedup factor of about X for the multithreaded versions. It may be a little less, given that certain operations like the partition for quicksort and the merge for merge sort cannot be done in parallel, but it should be close.

Your solution must work on Firefox and Chrome. Things that aren't supported on both browsers, like shared array buffers, must NOT be used. This is where the challenge comes in. You cannot just pass a pointer to the same array to several different web workers. Objects either get cloned or transferred when using postMessage.

The sorting algorithms must be implemented correctly, produce correct results, work on both Firefox and Chrome, and have appropriate speedup factors in the multithreaded versions for 2 of the 3 points. For the 3rd point, you must beat the speed of the built-in sorting consistently in both browsers with your multithreaded sorts. You may assume a quad-core processor will be used when grading.