So I have been looking into a few things, Note: this may change slightly once I get the SoGaCo source code.

Build Process:

Heres a rough diagram, I will update as I go along.
https://www.lucidchart.com/invitations/accept/7d1dabde-ee4f-42ba-880a-86865258fa47

I still have a question:

Do we want both a master and release branch? or do we want to use our master branch as the release. Note: I can use travis to compile and run both sets, but only the release branch would push to a live server. Please note that all testing should be done BEFORE pushing to master/release. The last test by travis is just a guarantee that by pushing your new section of code, no other tests break etc. This makes sure the entire application has been tested before release.

This build system should be complete by the end of next week, but to do that I need the start of everyones code, So I can at least get the baseline for the technologies everyone is using. What I want is for everyone to push some base code in their languages etc. So I can build.

Package Structure:

It is vital that we follow a uniform package structure across all parts of the application.

Since SoGaCo uses the maven package structure I feel all out code should be structured the same.

Heres a rough outline.

http://maven.apache.org/guides/introduction/introduction-to-the-standard-directory-layout.html

```
ProjectFolder:
        -src
                -main
                        -resources
                                -scripts
                                -html
                                -css
                                -fonts
                                -images
                        -java
                                -domain name (We probably should acquire a domain)
                                        -games
                                                -battleships
                                        -languages
                                                -blockly

                -test
                        -java
                                -domain name (We probably should acquire a domain)
                                        -games
                                                -battleships
                                        -languages
                                                -blockly
```

Build Specifics:

Main Build:
Will probably be something as simple as a makefile that simply calls the other 2 build scripts, and increments a version number

Java Build:
Look at the build scripts here, pretty much sums up why I use gradle.
http://technologyconversations.com/2014/06/18/build-tools/

Maven - SoGaCo uses this already. But it is extremely verbose and annoying, No one has really used any build tools other than myself, so there is no clear advantage from that.
I suggest we pull SoGaCo directly from their own build system/or build from source, still using it as a black box, which means we can happily run the latest SoGaCo build and integrate it. I'll try to integrate it so we can set a version based off their jenkins build system(I think thats what they use.) But I will look into this and talk to Jens.
I also suggest we build The language and game independently, which makes it easier to test.

The Java build order will consist of:
- -SoGaCo (Build from source, use as dependency for everything else)
- -Language
- -Game Logic
- -Web (Secondary build system)

FrontEnd Build:
Gulp - Gulp is a build system I have used very recently and understand it very well, I personally use it over grunt for a two main reasons:
- -Its simple
- -It's lightning fast

Basically it streams data, eg you specify a source (maybe everything in the javascript directory). Then you pipe it through plugins (Test frameworks/preprocessing/uglification etc.). Then output it to a destination.

We could potentially use a dependency bundler like requirejs/browserify/webpack to bundle everything into a js file (speeds up the server, reduces number of requests, caches site by hash of js file. Just all round incredibly useful.) But that would be another layer of complexity, It's definitely an option.

Continuous Integration/Continuous Deployment:

Why use it?

CI basically means when we push stuff to github it will pull the repository and build it, this way with every push we can monitor how healthy the codebase is, make sure all our tests are working etc. We can also auto-deploy to things like heroku/digital ocean very easily.

Why Travis-CI?

It comes free as part of the Github Developers pack, if you dont know what that is, check it out here: https://education.github.com/pack
It's free for open source, but this allows us to build from our private repository, Travis is probably the largest CI aside from Jenkins (Which you have to run on a personal server, which has a larger overhead than you realise). Travis also comes with inbuilt support for both heroku and Digital Ocean(See next section), making deployment effortless.

Deployment/Hosting:

There's a few places we could deploy to.

Heroku: Free https://www.heroku.com/pricing
I've used it before, The server actually sleeps after 30 mins of inactivity, thats how they keep it free.

Pro's:

-Don't have the overhead of having an entire VM, you just deploy the "Application"
-Free
-Incredibly simple to setup. You simply use git to push to heroku
"git push heroku master" and your done.
-Has free database access as well (to an extent, may not be what we need)
-Great for pushing something to live quickly
-Takes care of pretty much everything

Con's:

-Database pricing is expensive(After the first 10,000 lines)
-Free server takes around 30 seconds to start up after sleeping
-Free server only 512mb ram

Digital Ocean: Paid, $100 credit through the Github developers pack. Servers we would use are anywhere from $5 - $20 a month.

Pro's:

-Can resize the server(Increase required resources eg. For large scale testing increase the capabilities, but during dev decrease it)
-Full CLI, can install anything, control everything
-Can run the database on the same server, regardless of size.

Con's:

-Harder to maintain

AWS/Azure/Google cloud etc:

All cost, some have free trials, others(such as AWS) offer free servers for education (Jens would have to make an application to them). I don't mind using these but The trials are usually only 1 month, and unless we get the free server for educational purposes, I suggest we use digital ocean instead.

In my head, we could potentially leave both options, For now I honestly think we can suffice with heroku, and maybe switch to digital ocean down the track when we start testing on a larger scale.

Heroku is live already, we can spike off and explore other options later.