General architectural guidelines

Strive to follow the SOLID principles; * Single responsibility principle * Open/closed principle * Liskov substitution principle * Interface segregation principle * Dependency inversion principle

Furthermore we have decided to follow these slightly more concrete guidelines; * Do not use singletons. Singletons make testing much more difficult, and are generally an easy way out to compensate for poor architecture. * All concrete model classes should implement a superinterface. Doing so provides more extensibility, which should be strived for. * All subpackages should have a corresponding *Environment-class.

Naming conventions

- **Interfaces**: The name of the object. `PlayerShip`
- **Abstract superclass**: *Simple* before the name of the object. `SimpleEnemy`
- **Concrete class**: The name of the object, followed by *Impl*. `PlayerShipImpl`
- **Data holder/config files**: The name of the object the config is for, followed by *Definition*. `WeaponDefinition`

Documentation

At minimum all public methods should be documented with javadoc. All classes and interfaces should also have an explanation for what it does. Documentation inside of methods should only be provided if deemed necessary to understand the code and should be kept to a minimum. If you find yourself writing a lot of comments, you should most likely rather be restructuring the code than trying to comment it.

Git

The git model to be used is the one outlined in the blog post A successful Git branching model. This means using feature branches for new features and having the master branch working at all time, among others. Please read through the entire post to get a full understanding of the proposed git workflow.

Testing

For models, every public class should have a JUnit test associated with it. Testing of view and controller code is to be carried out graphically if unit testing is not applicable. The entire JUnit test suite should be run before pushing to the master or development branch.