


Monday, September 12, 2005

## MVC and FrontController: QS 3-0.10

Quantum Star SE 3.0.0 - the tenth revision

During the next 2-3 weeks, QS3 will hit its first major milestone (set way back in January 2005). The tenth revision (v3-0.10) brings together numerous ideas, plans and test code which have been hanging in limbo for months. In short it shows the future face of v3.0.0 - and what sort of mutated beast it will be 

The primary idea to keep in mind is that QS does a fair job at separating data, business logic, client code, and presentation. The original pre-alpha back in April was not quite so clean - it integrated everything into a ad-hoc hodge podge. v0.10 however handles things quite different.

The primary "structure" being used is one based on the Model-View-Controller design pattern. For those not familiar with the concept - MVC describes an object oriented class structure which separates certain broad tasks - typically Controller, Views, Business Logic and Data Access.

The advantages of the above is that it maximises code reuse, and minimises code duplication (the major banes of QS2). In doing so it breaks down many aggregate tasks into their individual tasks - and sets each as a object method. Doing so allows far simpler maintenance and debugging. There are other advantages (but I'm trying to limit this entry!).

For QS3 the MVC implements a Business Logic Layer, a Data Access Layer, a Presentation Layer, and a Controller (the Controller is yet to be added - since all layers are independent, changes to one do NOT effect another - basically its made of a Front Controller, which makes calls on various Page Controllers). The Data Access Layer handles all SQL queries, and their results. The Presentation Layer (simple Smarty-Lite) prints parsed templates to the browser. The Business Logic (or Model) layer handles any internal logic operations - e.g. deducting credits, adding turns, checking if a ship belongs to a fleets, etc. The Front Controller handles incoming requests, and based on those requests accesses the Model Layer to perform the requested action.

As an example:

Player A wishes to Warp to SS#2 from SS#1. He clicks a url of the form (<http://www.example.com/index.php?action=location.navigate&starid=2>). The Front Controller will immediately parse the action value - in this case it refers to the Navigate command of the Location module. On this basis, it will search for the relevant Page Controller (/Modules/Location/NavigateCommand.php). The Navigate Class will contain the simple methods (common to all Commands) of execute() and render(). execute() will make calls to the Business Logic Layer to update locations of User and Fleet (in QS3 a Ship's location is taken from it's parent Fleet) while render() will parse the relevant Smarty Template and output the result to browser.

Its worth noting that the Business Layer contains delegating functions to the Data Access Layer (in effect, a Command class developer would only need to know the BL API - as the BL will pass off to the DAO layer methods, i.e. it handles any other related classes). I also note that data is passed between all layers as a Transfer Object (i.e. an object with properties populated from the data source with simple get/set methods for each property - TOs are passed to the Data Access layer for writing to the data source is needed).

Now to all reading, this may seem horribly complex (and it can be!), but one thing that should be noted is that all this separation and complex contortions results in very fine grained separation of tasks - leaving (of course) many more discrete files, but which are organised by task and module, and are open to editing without disrupting any other code. It

comes at the price of intuitive understanding - but I think its a solid step forward.

So what about the game!!!

The impact on QS3 play is not a huge one (at least compared to the 3-0.6 pre-alpha). The game still does the same things - but it does them faster and more efficiently. There's been added a certain openness which will allow complex plugins and game modules. Single entry points into the game allow centralised controls over input data, url validation, etc - adding security, and removing any alternate entry paths. It also allows a neater method of developer's authorising valid url's in response to any request - i.e. restricting pageflow.

You thoughts are welcome.

Posted by Pádraic Brady at 19:23