

Tuesday, January 30, 2007

Factions: Social Control on Newbie Bashing

A recent discussion was sparked on the Astrum Futura forums regarding player retention and new player protection. It was one of those times when the discussion branched to a higher view - a wider look at how players interact on a large scale.

The current suggestions called for players to be penalised for undesirable actions, e.g. attacking a new player who is incapable of defending him/herself. This remains a long standing issue in many online games. AF could resort to enforced levelling - limited who a player can attack, and be attacked by. But this is all artificial - in the real world it's open season on anyone.

Cyberlot (Richard) raised the question of penalising through a Karma rating. As a player attacked newcomers they would attract negative Karma marking them out to other players. I can see the rank of "Newbie Basher" emerging as an undesirable outcome for many players. But Karma while adding some measure, does not address enforcement. Exactly who enforces what and when? Who sets the standard for interacting with other players?

From there we hit the subject of Factions. A Faction would be a specific group of players sharing a common interest. This obviously includes Races, but is also extendable to a Merchant Guild or a Pirate Clan. The point is to give players an instant group they are a member of. Now where it gets interesting is how the player and game logic would respond in the presence of Factions.

Say you have a Faction, Earth Humans United (go Terrans!). What happens if a member of this Faction attacks a new player who is joining the Earth Humans United Faction? Can't have allies killing each other off (even if it might reflect reality). The answer is in the Karma. Killing a new member of your own faction is stupid, damages the Faction, and makes other players less likely to join it. That member should start paying tax to the Faction, which is needed. Such a player would immediately get negative Karma.

So we have Karma, and we have Factions - how do they relate?

Karma is a measure of one's standing within a Faction. If your Karma falls below a minimal level you'll be declared an enemy of the Faction, and then it's time to start running before you're hunted down by your peers. Where do you flee? If you're lucky, some other Faction still has you as Neutral.

But this is where Newbie Protection gets interesting. It's not Faction specific - your bad Karma is classified when killing a new player as a "Stigma". It decreases Karma for ALL Factions. It's the ultimate punishment - imagine playing when nearly all the Factions refuse to grant you trade access, when your planets are penalised with "special tax charges", when other players are unable to maintain an Alliance with you once you become Factionless (Outcast). In short, if you gain a reputation for killing new players, you'll be kicked, booted and sent to Hell with a one way express ticket. No Admin interference or funny AI fleets required. Society will put you straight or else...

More on this later as details are worked out, but it looks like it's a feasible and supported suggestion.

Posted by Pádraic Brady in Astrum Futura, PHP Game Development, Quantum Star SE at 16:37

Monday, January 29, 2007

Thoughts on a Unit Testing and Test-Driven Design Experience

Creeping closer to Spring; it's 1 February if you follow my calendar. One of the things which stand out in my online activities as it touches on PHP is the 3 month period spent working on the QGL (Quantum Game Library). The library is just reaching its 200th commit, and should pass that barrier later this evening when I get around to a small class naming change.

Sticking with the QGL, it's a small side project which sprung up from a vague idea. Those of us working on the related Astrum Futura game project, knew a standard game library of some description would pay dividends for future projects by centralising a general set of useful classes. It was shortly after this concept was solidifying when Jacob Santos joined the team with his imaginative vision of implementing AI algorithms.

The one thing that really stands out from the usual project hustle and flow, was our shared wish to use Test Driven Design and Unit Testing. It's very difficult to use it on anything more than personal projects at work where developer resistance (though falling) is still an obstacle. So the QGL was an all too rare opportunity to cooperate in a team led effort focused around the test-first approach.

So how did we fare?

Personally, I think a lot of what we accomplished to date (bearing in mind the time limitations) was a result of TDD, and of course Unit Testing's benefits in general. There were times where we were working on complementary components such as Jacob on `Ai_Pathfinding`, and myself on `Quantum_Map_Measure` where TDD led to simpler more manageable classes. Of course the unit tests supported our efforts with style. Throughout the process we were both refactoring code, and tweaking interfaces. Without the unit tests we would have reached deep treacherous waters as our respective efforts slowly drifted apart.

I also think poor Jacob may have become test infected. He seems to be recovering, so I'll have to wind him up later over the current two failing tests he introduced! I haven't heard any rumours of a bald Jacob Santos so presumably that odd side symptom has yet to rear its ugly head.

Back on track... The most visible benefits of the TDD + UT approach during the last three months can be summarised as:

- Easier Refactoring (perform a refactor, check tests still pass, rinse and repeat)
- Immediate Feedback when code goes wrong (big fat red bars in the testing results. Hard to ignore!)
- Enforcement of Standards (tests discourage questionable hacks/shortcuts)
- Regression Testing (if something goes wrong, we add test(s) to prevent repeats)
- Force Feedback (not just a console controller type, failed unit tests spark attention from other developers)
- Code To An Interface (TDD and unit testing teach you why if you're a quick study)
- Testable Code is Better Code (adherence to practices which generate clean, focused, flexible classes)

I'm sure there are others, but these were the most obvious over three months of development across 200 commits to subversion from two developers. If the immediate benefits are not a motivation to become test-infected the future benefits might. The one thing I know from past experience is that unit tested code tends to be easier to maintain and generates fewer bug reports.

Now if only someone miraculously removed the initial pain a developer experiences when starting to learn TDD and Unit Testing (the pain that explains why many quit before the summit of the learning curve is reached) we could get more folk on the TDD bandwagon. One unfortunate fallout from following a test-first approach is that it necessitates any new developers becoming educated. No easy task when dealing with a vast population of wannabe programmers taking their first few running jumps with PHP by showing interest in open source projects.

Posted by Pádraic Brady in Astrum Futura, PHP Game Development, PHP General at 16:58

Blog Export: Maugrim The Reaper's Blog, <http://blog.astrumfutura.com/>

Thursday, January 25, 2007

Time to tease...

A cryptic message was posted to lamsure's blog earlier: <http://kabal-invasion.blogspot.com>

I'm not sure what's next, but will we have to wait another month to find out?

Posted by Pádraic Brady in Irishisms at 11:31

Tuesday, January 23, 2007

Doing the Poka-Yoke

After getting a little impatient looking at PHP code from however many projects and seeing the typical approach of making input filtering, sql and output escaping the responsibility of the human error-prone developer, I'm now making it standard practice on any of the projects I run to dump this sorry mess. And it is usually a mess.

The fact is you cannot trust a developer to manually secure source code - it's like God relaying the Ten Commandments to Moses. No matter how many burning bushes, prophets, unnatural disasters, signs and miracles you use to emphasize those ten simple rules, you can still wager there will be a bunch of folk breaking them (including myself!) left, right and centre. It's human nature to err. It's my nature to make the verb "err" redundant.

Since the developer is not to be trusted with security (at least not on a detailed manual basis), I have over the months integrated a few simple practices into some of my code, including Zend Framework subclasses where appropriate. I say simple, because when implemented they become automatic and require no developer decision making (i.e. no stupid errors).

Here's one example.

For HTML output with templates, escape all variable data assigned to the template engine. Why? Because expecting the template author to call that hanging `escape()` method where requires allows for that author to make a mistake. Mistakes that can introduce security vulnerabilities nobody else might catch before an application is out the door and some scriptkiddie is doing the samba at their PC...

That may sound a bit over the top and strange (so the emails go), but is relatively simple to implement. You just find the template engine's setter method, for example, the Smarty or Template Lite `assign()` method, and modify/subclass it to escape all assigned data by default. The default bit is important - force the developer to manually disable it if absolutely required.

If you use the Zend Framework, this type of modification requires subclassing `Zend_View` with a class containing the following method:

```
public function __set($key, $val){ require_once 'Zend/View/Exception.php'; if ('_' = substr($key, 0, 1)) { throw
new Zend_View_Exception('Setting private or protected class members is not allowed'); } if(is_array($val)) {
$clean = array(); foreach ($val as $index => $value) { if(is_object($val)) { throw new
Zend_View_Exception('Objects may not be directly set as template variables, only arrays of, or standalone, values.');
```

```
    } $clean[$index] = htmlentities($value, ENT_QUOTES, 'utf-8'); } } else { if(is_object($val)) {
throw new Zend_View_Exception('Objects may not be directly set as template variables, only arrays of, or
standalone, values.');
```

```
    } $clean = htmlentities($val, ENT_QUOTES, 'utf-8'); } $this->$key = $clean;}
}
```

I completely ignored the `Zend_View` ability to define an escape function as well as the character encoding to use - so lump that in if you wish. The main limitation of the approach is of course that values are escaped under the assumption they are strings, so assigning objects has been made an Exception though after a single dimension that detection will falter - unavoidable with the code above without implementing a recursive check. Personally I see no harm in this - I won't always have total control over who is writing templates so why should I give them additional rope to hang themselves by handing over objects instead of simple value arrays?

The Exception handling isn't pretty either, but if subclassing it's likely the subclass would have its own separate Exception class anyway. It would be nice if the ZF put includes in a predictable placement but the `__autoload()` debate on the mailing list and other preferences are sometimes pushing Exception file includes to differing places depending on the component.

If special treatment is needed, i.e. some object absolutely must be passed, then a second custom method may be added to handle such special cases. The point of course being to force a template author/developer to deliberately override the default behaviour as needed which presumably means they will think about why they are doing this first.

As for the entry title, Wikipedia has this definition of Poka-Yoke:

Poka-yoke (ポカ・ヨケ, pronounced "POH-kah YOH-keh" means "fail-safing" or "mistake-proofing" - avoiding (yokeru) inadvertent errors (poka)) is a behavior-shaping constraint, or a method of preventing errors by putting limits on how an operation can be performed in order to force the correct completion of the operation. The concept was originated by Shigeo Shingo as part of the Toyota Production System. Originally described as Baka-yoke, but as this means "idiot-proofing" the name was changed to the milder Poka-yoke. One example is the inability to remove a car key from the ignition switch of an automobile if the automatic transmission is not first put in the "Park" position, so that the driver cannot leave the car in an unsafe parking condition where the wheels are not locked against movement. Another example can be found in a normal 3.5" floppy disk: the top-right corner is shaped in a certain way so that the disk cannot be inserted upside-down.

Edit: Altered code to reflect Zend_View_Abstract no longer extending ArrayObject (I need to update my source after the January bustle!). Thanks to Waldemar Schott for pointing this out. Took a time out to elaborate a bit more as a result too.

Posted by Pádraic Brady in PHP General, PHP Security at 17:47

Sourceforge Ranking Review

Sometimes I find it useful when measuring what's being downloaded in terms of open source PHP games to look at the top 10 rankings on Sourceforge for each gaming category. I say sometimes, because it tends to be fairly unreliable and doesn't really reflect real usage online.

The list of top ten Turn-Based Strategies (as filtered for PHP use) include:

1. phpDiplomacy (268)
2. Some Chess (1077)
3. Quantum Game Library for PHP (3196)
4. Solar Imperium (3860)
5. Solar Empire (4032)
6. REMLAB Web Mech Designer (4202)
7. Alien Assault Traders (5272)
8. PHP RISK (7156)
9. Star Trek: Allegiance (7351)
10. GameServ IRC Services (7417)

This shows the first place with a total showup is phpDiplomacy. The numbers in brackets represent overall Sourceforge rank against all other projects of all categories. 268 is remarkable. The Quantum Game Library, likely reflecting its status as a library as well as still hosting the current 0.7x series of the QS game, is third with a total rank of 1077.

QS is a really odd one to be honest. We've registered 36,000+ downloads compared to Solar Empire's 21,000+ and Alien Assault Traders 12,500+ - I can't explain it... Most of the downloads peaked over a year ago.

As for what the rankings tell me - all they really show is there is a fairly wide gap between each placement. The number one and two spots are widely separated from subsequent rankings. Overall the QGL has leaped up the rankings since last year. Our ranking on SF has moved from 16,808 in September 2006 to 1077 as of 23 January 2007. This marked shift coincides with the transition of the project from the QS game to the QGL library and marks the project's highest average monthly ranking since June 2005.

Although SF ranking as a measure of project value is undoubtedly suspect. It's still a large driver of publicity so it's worth improving. Astrum Futura is nowhere in sight, which is fine by me - it's only getting started and doesn't warrant much exposure yet.

Posted by Pádraic Brady in PHP Game Development at 11:52

Now broadcasting from astrumfutura.com

In case you haven't noticed, the blog is now hosted on the astrumfutura.com domain. I've added a set of HTTP headers on the quantum-star.com domain which should transparently redirect many of the old links to the QS blog to the correct location here.

Blog Export: Maugrim The Reaper's Blog, <http://blog.astrumfutura.com/>

Since the blog has moved, I'll explain why. Back in December the blog fell offline when a series of bad PHP files were written into all the writeable directories. Since I was using Hostrocket, these are writeable only under a 777 chmod. It's a pain in the ass, but unavoidable because of how Apache is set up for Hostrocket's shared servers. Unfortunately, this keeps happening month after month regardless of changing passwords, updating version, etc.

Astrumfutura.com is on the other hand, hosted on Dreamhost. Dreamhost have this nifty thing going where Apache runs under the local domain user account. Result? Apache (and therefore PHP) need no special permissions to write to the server by default - which of course means no 777 permission anywhere. It has it's own issue (PHP apps have default write permissions) but it avoids the pain in the ass world-write situation.

Anyways, enough of my moaning. Back to blogging!

Posted by Pádraic Brady in Irishisms at 10:00