Rethinking Game Architecture with Immutability – Milestone 6 (April 16th)

Student: Jacob Dufault (jacobdufault@gmail.com)

Faculty sponsor: Professor Bernhard

Progress Matrix

Task	Completion
Work on XNA/MonoGame bindings	100%
Demo a simple game running in both XNA and Unity	100%
Continue work on Unity integration	100%
Writer user manual and documentation	100%

Task Summaries

Work on XNA/MonoGame Bindings

The XNA bindings work well right now. All gameplay logic can be shared between renderers, and there is a compatibility framework so that some renderer-specific data can also be shared. The XNA bindings are relatively minimal as more development experience has shown this to be a more optimal development method; more can be added in the future as pain points are discovered.

Demo a simple game running in both XNA and Unity

The simple game has been created and works; it runs in both XNA and Unity. The gameplay sample successfully demonstrates object creation, object updating, input, object destruction, and template usage. The focus is on demonstrating how to use forge and key concepts, not on any actual gameplay mechanics.

Write user manual and documentation

The user manual has been written and is included in the github page for forge. There is also documentation generated via Sandcastle for all of the extensive XML comments on all of the types within forge.

Lessons Learned

Immutability is extremely useful

Immutability prevented a number of bugs, such as those related to update order.

Favor simplicity over performance

Large amounts of time have been spent on optimization, which is likely unneeded. Instead, it would have worked better to have good levels of abstraction where the implementation could be changed to give much better performance.

Don’t optimize until profiling

It may take very little time to write something that works but the implementation may be slow. However, if the feature is used very rarely this does not matter and the developer can move onto more important things.

Multithreading introduces subtle bugs

It’s easier to just ignore multithreading until it’s necessary. Of course, the initial architecture needs to be designed such that it can be multithreaded. In terms of games, the actual game logic likely does not need to be multithreaded. Instead, core systems, such as pathfinding, physics, rendering, etc, can all be submitted as jobs and they instead are the multithreaded logic.

Unit tests are awesome

Unit tests caught a large number of simple programming mistakes and helped maintain application stability.

Dual content/runtime implementations are tricky

Lots of serialization complexity to support dual content and runtime implementations, including a decent amount of duplicated code. This would be easier if there was an explicit serialization format/library that handled these issues. Then the content engine could be completely separate from the runtime engine.

Sponsor Feedback

Signature and Date: _____

Feedback:

Sponsor Evaluation – Rethinking Game Architecture with Immutability – Milestone 6

Jacob Dufault

Score (0-10):

Signature & Date: