Ruud Peters 404765

Design Document Game Architecture

Introduction

For this assignment the student was supposed to improve the architecture of a simple Pong game. Different patterns had to be applied, in order to make the engine worth building upon. As everyone knows: a structure is as strong as it's foundation.

My personal goal

In two more weeks, I can call myself a third year student. Compared to the soon to be second year students, I've had a lot more experience with custom engines (like the custom c++ engine for 3D rendering). I've taken quite a bit of inspiration from both the c++ engine and Unity, as unity is a very easy platform to build upon. Functions like Start, Update, Oncollision and OnRender were fun features which I enjoyed implementing in the GaEngine. I've tried decoupling the GameObject from the components as much as possible, without going overboard.

Component

As I mentioned Earlier, the component has virtual functions which each can be overridden. The true magic lays in the background: barely any setup is needed to make these functions work. Adding a simple BoxColliderComponent enables the use of the OnCollision for example. Adding a ImageComponent automatically uses the OnRender function. Everything is taken care of.

EventQueue

The queue is a simple ring buffer queue, which takes advantage of the speed of an array. Different types of Events can be sent to it, which each can easily be resolved by overriding the Resolve function. The EventQueue is resolved after updating and collision checking, just before rendering.

ServiceLocator

The servicelocator uses template functions to easily access different types of classes. Simply using ServiceLocator.Locate<World>() is enough to parse the world. No more needed of unnecessary singletons!

World

When the update and render functions are called in world, it'll go down the hierarchy of children, all the way down to the components. Handy for managing GameObjects.

GameObject

The GameObjects has multiple handy template functions. Like Unity, it is able to use AddComponent<componentType>, GetComponent<ComponentType> and RemoveComponent<ComponentType>. Each GameObject can be given a Tag too, which can be used for searching them in any component via world.

Ruud Peters 404765

Statemachine

Although the statemachine is barely used (it only has two states), I chose for this solution because there are no more states in this game. I was thinking of adding A score and reset state, but those aren't states.. those are events. The Statemachine is easy to build upon though, features like menu's are simple to implement using the StateTransition dictionary.

Epilogue

Although I'm quite proud of my code, the diagrams are quite lacking because I had no chance to get feedback. In the 4th quartile we had an 8 week long project, which took all my attention. I hope that you can see that **I do understand how the basics of each of them work**, I just didn't have enough feedback to polish them. Hopefully the quality of the code really shows that I understand the patterns. A lot of thought went into them.

Thank you for reading, have fun looking through the sourcecode ☺ The diagrams are under diagrams/PDF