**Impacts of FSM (Finite State Machine) In an ECS (Entity Component System)**

Executive Summary

In this article, I will go over the negative and positive impacts of adopting the ECS and the coupling of FSM into the ECS pattern. I will cover topics such as **Migration of Code, Complexity of Code, Costs of Re-Training Developers** to showcase these impacts.

Associated costs

If we consider a standard monthly salary of a senior game developer to be at **$9000** and if the amount of developers working are 50, a video game company that wants to create a game, with a complex ECS that handles memory management and component filtering with an intricate FSM coupled into the it, the time it will take to develop an ECS that is coupled together with an FSM, that is easy to understand for other developers and follows software practices such as SOLID principles, is hard to estimate, as a lot of work will be done in the designing of the ECS paradigm and the coupling of FSM with ECS. A best estimate of the time to take, would be around 2 months to design and 4 Months to actually develop. If we take our standard salary into account and 6 months total development time. The costs of developing this for 6 months for one developer is ($9000 \* 6) **$54,000** and for 50 developers the cost will be (54,000 \* 50) **$2,700,000.** This could have an impact, as the company might not have the capital to spend on 6 months of development time on something that the end user might not notice (They might want to put that capital into the advertising or art department that will make their game stand out more).

OO (Object-Orientated)

For small-scale games/projects, OO can be very useful to create these games/projects. This is due to the fact that OO is very fast to develop in. This will reduce the cost of the production of small-scale games and less overhead for game designer’s, which will leave more capital to use in other departments. Even in AAA games designers and developer would work to create early prototypes of the game they want, and this could be done in OO to test out the fundamental game mechanics and make sure that the game they are making is immersive to the user. But this will fall short when you want to develop big scale AAA games, which will lead to large inheritance trees that will cause problems down the road in production. These problems would be issues where certain classes inherit in such a way that could lead to a problem known as the “Diamond Problem”. These issues can be solved if you use the ECS paradigm, that will get rid of inheritance all together and will make all the code decoupled leading to a cleaner codebase.

ECS And FSM Coupled Together

The idea of the ECS paradigm is to completely get rid of polymorphism and inheritance tress, which in turn will lead away from OO and into DDD (Data-Driven-Development). The reason why ECS goes this route is that OO does not scale easily into AAA games and will lead to problems that developers will need to get around. ECS is a very different way of coding. Instead of thinking characters/npc’s as objects, in ECS we think of them as data. This type of programming is hard to understand and even harder to implement, as ECS doesn’t use polymorphism or inheritance. This will lead to an impact, where the developers will need to design out the entire ECS system and make sure that they have good memory management and really concrete component filtering, which will make it very easy for other developers to create/use different components together.

In OO, it is rather easy to implement an FSM into a game, as you can use polymorphism. All the states can inherit from a base state. This will lead to a problem where I have mentioned before, to a large inheritance tree. ECS doesn’t use polymorphism at all, so coupling together an FSM without inheritance will be very hard and time consuming. The biggest impact of this will be, time taken out in development of the game you actually want to create and into a concrete implementation of ECS with the FSM coupled together. One approach I would take to couple FSM into ECS is by using tags in the ECS. So, for example you could have different tags for states (i.e., “Walking”, “Jumping”).

This implementation can be done, but the problem of this approach is maintainability and speed. This is due to the fact that if you want to check if a particular entity has a state, then you would need to check it like this (if(entity.hasState<Walking>()). And you can add/remove states like this (entity.removeState<Walking>();). This is not maintainable, because if another developer wants to add a new state, then they would need to add to the existing codebase. Also, if you want to get the current state of an entity, you would need to query this in your ECS systems with something like this (query<Walking, Jumping>). With this approach the more states you have, the less maintainable and slow (i.e. increase in cache misses) your ECS will be.

Another approach you can take is the archetype approach, which stores entities with the same sets of components/tags in a table. This will lead to solve the slow issue of the tag approach I mentioned above, because the archetype implementation leads to a cache-efficient component iteration, where all the components will be tightly packed in contiguous memory. The issue with this approach is that if you add/remove tags from entities, which in turn you will need to update the map between entities and components/tags. This if not done properly can slow down your ECS.

A solution that I would recommend is the tag and archetype approach with the addition of having a linked-list per state, that is stored adjacent to the FSM component array with the different state identifiers stored within. So, by this approach all you will need to is go the head node of the corresponding linked list and follow all the nodes through the linked list and get the states that you need. The linked list can be stored as another adjacent array of integers where the integer will point to the next node in the array. Also, because states are mutually exclusive from one another, the lists for different states can be joined together in a single array which can be stored alongside the component array. This approach is slower than the component iteration, but it will not increase the overhead of inserting and removing states (remove + insertions in linked lists are constant time). This approach is also very fast in doing many queries to find a particular state in an entity. This approach done probably will give you a good FSM that is coupled to the ECS paradigm.

Summary

In conclusion, ECS with FSM coupled together, will lead to a better development environment, where the developers will be able to create and re-use different components without the risks that OO brings such as the “Diamond problem” and large inheritance tress. In my personal opinion, ECS with FSM is not warranted, unless you have the capital to spend for 6 months or more development time to create an ECS paradigm with FSM coupled in and also, if you want an engine that has the capabilities of re-using different types of components together.

Recommendations

In my opinion, I would recommend companies that are developing big scale AAA games that have already developed a multitude of games with OO with their own custom engine to integrate ECS with FSM coupled together into their engine. The reason for this, is by developing this paradigm, you will create a pattern that any game can use and will cause less overhead while you are creating multiple different games.

If you are a small-scale studio that wants to enter the game development industry. I would recommend the use of OO to get started, as you can implement prototypes and games in general pretty fast with OO whereas with ECS, there will be a big overhead on the initial development of the ECS paradigm that small-scale studios won’t have the capital and time for. As re-writing your entire engine to use ECS and FSM is very time consuming and hard, as it’s a different way of thinking.But over time, after a small-scale studio as created a video game that the public audience like.