Task 21 - Spike (opt): Entity Component System

Context: An Entity Component System (ECS) is an architectural software pattern often used in games and game frameworks. It is used as an alternative to deep inheritance hierarchy that often don't fit well with complex or diverse game entities, and instead uses composition. Composition provides a flexible way to define game entity data, state or behaviour. The "system" idea, promoted to a first-class feature of the pattern, performs global actions on all components, and is disconnected from the entities.

Knowledge/Skill Gap: The developer wants experience implementing an entity component system pattern.

Goals/Deliverables:
[CODE] + [SPIKE REPORT]

Implement and test a basic entity component system. There are various approaches to implementing the essential ideas of an ECS design, but we want one that moves away from each entity having the components it uses, and to also (ideally) keep component data in a packed format. So, for this task demonstrate the follow key features:

- **Entities**: Simply an ID. Does not (essentially) contain anything. The ID is used as an index into a collection of components (although best if this contiguous memory – an array).
- **Components** as Plain Old Data (POD) -a simple struct with only the relevant data included. Again, ideally stored in contiguous memory (~an array). Suggest also giving your components a unique ID.
- **Systems**. The logic that operates on components. Each system has access to the components (or contains a list of all entities IDs to get to the components) the system will read or write to.

To support the creation, configuration and destruction of these, also demonstrate the equivalent of:

- **EntityManager**: Entities are very simple. There should be a way to wrap up making and removing entities. You can use the entity manage to work with the component manager to create entities with the components they need.
- **ComponentManager**: Looks after the allocation and removal of components to collections. This manager should allow use to get all the components of a certain type (or have access to the collection). It can also help the system manage have access to the components relevant to a system, without direct knowledge of the entities using the components
- **SystemManager**: There are a number of systems, each needing components of particular type. This manager looks after all the systems and directs them to operate on their known components at the appropriate time.

Your demonstration of the ECS pattern could be a simple console program, or a graphical (SDL) simulation. In either case, keep it simple for the needs of this spike. You can use an existing ECS library, or create your own, but you need to be able to demonstrate the features listed above.

Note: The objective for this spike is not to create an optimal and ideal ECS system. The key is to demonstrate the ECS parts – in particular the "system" is able to alter components without knowledge of entities. You only need to show a simple working example. Keep the work for this spike as simple as possible while demonstrating the ideas.

Ideally, components will not know about each other but there are times when they might need to. This can be coordinated via the ComponentManager, or you might allow some knowledge of components between components.

Recommendations:

- If you are not sure how to design and implement your own ECS, you could try an existing ECS framework. There are several excellent ones publicly available. However, they are typically not simple to understand! If you want to do this, take some time to study their designs first. You might find enough detail from their design to create a minimal working version of your own with the full optimal complexity.
- Only take on this optional challenge if you are comfortable with your other core work so far. You can always come back to this when you are better prepared.