Task 15 - Spike: Composite & Component Patterns

CORE SPIKE

Context: A text-based adventure game can create an immersive experience where entities of the game can be compositions of other entities. To do this, player commands need to act on "entities" of the game, some of which are composed of other entities. This can lead to heavy use of inheritance, which *can* be appropriate for games as simulations (however stylised) of the real world. In many instances though, this can lead to unnecessarily deep class 'trees' and many abstract objects intended to represent specific properties an object deeper in the tree may have. Unlike compositions, the component pattern de - emphasises inheritance as the source of object attributes by creating objects out of components, each one contributing some attribute or function to the complete object.

Knowledge/Skill Gap: The developer needs to know how to create and modify, for a text based adventure game, game entities that are composed of other game entities. They also need to be able to create and modify, for a text - based adventure game, game entities that are the sum of their parts, receiving attributes from components rather than inheritance.

The developer lacks an appropriate understanding of how the patterns are implemented and where they are best applicable.

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

Building on the work of earlier Zorkish related Lab and Spike work, create a game world to support game entities composed of other game entities as well as game entities that have components which contribute properties and actions.

Create part of the Zorkish game that demonstrates the following:

Composite Pattern features:

1. Adventure (world) files that include the specification of game entities, their properties, and any nested entities (composition) they may contain.
2. Players are able to observe and modify entities (what they contain, and their location) i.e. "look in", "take _ [from] _", "put _ in _", "open _ [with] _"

Component Pattern features:

1. Game objects that receive attributes (damage, health, flammability, etc.) from component objects rather than inheritance.
2. Game objects that receive actions (can be picked up, can be attacked, etc.) from component objects rather than inheritance

Recommendations - see next page.

Recommendations:

- To avoid confusion (seriously!) put designs and plans on paper. Think as much as possible before you code! (If you do this, be sure to include your paper design with your outcome report.)
- Don't try to be too clever. Get a simple version of each **separately** before you attempt anything interesting… then get creative
- The game location graph can be extended to support entities that are collections of entities – this is the essence of the OO composite pattern!
- Read the game specifications again.
- Research dictionaries – collections that can access contents using string keys. (STL)
- Create a new adventure file that contains a minimal game world description and some entities that also contain other entities that you can use later for testing.
- Update the adventure loading code so that your game world (graph) supports the entities and the composite pattern
- Implement the commands, test… extend… until done.
- Test. Check for memory leaks… (Seriously!)
- This is a good option to start learning about the component pattern: http://cowboyprogramming.com/2007/01/05/evolve-your-heirachy/
- Also have a look around here: http://gameprogrammingpatterns.com/
- The visitor pattern is has some similarities to the component pattern, and tutorials on implementing the visitor pattern can provide inspiration for implementations of the component pattern.