Task 13 - 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.

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

Building on the work of earlier Zorkish related tasks, modify the game world to support game entities **composed** of other game entities, as well as game entities that have **components** which contribute properties (health, strength, etc) and actions (shoot, switch on, explode etc.).

Create part of the Zorkish game that demonstrates the **basic features** of the following:

Composite Pattern (think "tree" of nodes that can contain other nodes):

- Adventure (world) files that include the specification of game entities, their properties, and any nested entities (composition) they may contain.
- Players are able to observe and modify entities (what they contain, and their location) i.e. "look in", "take _ [from] _", "put _ in _", "open _ [with] _"
- Composite details are best loaded from the adventure text file (not hard-coded).

Component Pattern (think "entity" that has a list of attributes/actions from its "components"):
At least one (maybe both) of the following:

- Game objects that receive **attributes** (damage, health, flammability, etc.) from component objects rather than inheritance. i.e. The "health" attribute is in the collection of object "attributes". When you "look at" an entity it should list it's attributes.
- Game objects that receive **actions** (can be picked up, can be attacked, etc.) from component objects rather than inheritance. i.e. The "shoot" action is available because it is in the collection of action "attributes". You will want a way to list what actions an entity has or include that in the "look at" list to keep it simple.
- For this spike you can hard-code the setup of an entities attributes or actions, or load from a file. Up to you.

Note: For the component pattern (in particular) you'll need to update the command manager to be able to "ask" an entity about its attributes or ask it to "do" a certain action. Some general examples:

- `open mailbox` == the "open" action is something the "mailbox" entity has, and it will be called and then change the mailbox (its owner) attribute of "state" from "is closed" to "is open".
- `use potion on dog` == the "potion" has a "use" action that would modify the target "dog" by checking if the dog has a "health" attribute, and then changing it (by, say, +2).

Recommendations - see next page.

Recommendations:

- To avoid confusion design first! Think as much as possible before you code. (If you do this, be sure to include your paper/diagrams/notes of your design with your outcome report.)
- Don't try to be too clever. Make a simple version of each **separate** key idea (mini demo) before you attempt anything interesting with it all combined.
- 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.
- 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 has some similarities to the component pattern, and tutorials on implementing the visitor pattern can provide inspiration for implementations of the component pattern.