Task 18 - Spike: Message Systems

CORE SPIKE

Context: Objects in games often need to communicate with a wide range of other objects. In order avoid the maintenance nightmare that is coupling every game object to every other game, a more general-purpose messaging system can used to help game objects communicate.

Two general message system architectures are “blackboard” systems, where game objects “post” and “check” for messages, and “dispatcher” (think “post office”) approaches that encapsulate and manage collection and distribution of messages.

Other common features of messaging systems include beibg able to **broadcast** from one to many entities, **delay** or **schedule** of message delivery, and **filtering** delivery of broadcast messages based on criteria (such as world location or entity properties). Note that messages can also be sent from a game entites to itself as a way of scheduling its own behaviour!

Knowledge/Skill Gap: The developer needs to be able to send messages between the components in the Zorkish game context, using a flexible and robust messaging system. Several practical message system features need to be demonstrated.

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

You need to develop a messaging system to facilitate communication between game objects (entities) in your Zorkish implementation.

You need to produce:

1. A design for the message system (overall architecture, blackboard or dispatcher), expressed as a class/module/sequence diagrams (or equivalent), including a clear description of your message details. (See notes below). Sumitted as a PDF doc with all diagrams/figures you create.
2. A working Zorkish demonstration that shows how a message can be used to change a game entity. For example, change the state of a “box” in the game that is initially “locked”, and using a message, is changed to “unlocked”. (Other ideas: on/off, running/stopped, open/closed, fast/slow/stopped).

Note: Do NOT use a message system for the command processor (at least for this spike), however you will probably create a new Command that “sends” a message to an entity.

Note: Do NOT implement broadcasting, filtering, or schedule/delay behaviour in this spike. If you wish to do this, look at the next spike extension that is for these options.

Recommentations:
You’ll want to put some thought into this before you start coding - design an **architecture** to handle messagesa, and a **specification** for your message details.

You will need to consider:

- How messages are sent ,received and acted upon
- How messages are addressed
- What content is included in a message
- How objects “register” (connected to post office?) to receive messages
- Whether a message contains information about who sent it (is it needed?

For implementaiton, keep it simple. Simple types and strings are fine for message “details”.