Task 28 Spike: Observer Pattern

OPTIONAL

Context

Changes in one part of a game often require responses from other components. For various reasons*, the changed components may not be able to keep track of all components that should be notified when it is updated. The Observer Pattern allows components that require notification of a change to observe other components and respond when they update.

Knowledge/Skill Gap:

The developer needs to know how to allow Game Objects to watch other Game Objects for changes so that they can respond.

Goals

Building on the work of earlier Zorkish or SDL Spikes, create a class that observes another one and responds to a state change on the target class.

*You should comment in your spike report on the reasons one might use an Observer Pattern instead of directly coupling objects.

Expected Output

Repository

1. Code
2. Spike Report

Canvas

1. Spike Report

Notes

- You will have to decide whether the Observer Objects register themselves with the Observed Objects directly, or with some intermediary that controls the observing process.
    - Hint: less coupling is better
- In the case of Zorkish, a Trap class (per the Zorkish Part II specification) is an ideal use case. The trap class can observe what Location the Player is in and has a chance of activating itself when the player enters its Location.
- In an SDL game, a collision activated object such as a projectile could also benefit from this use case.
- Other cases could include updating a score or achievement counter.