**Extension:** Task 19

**Title:** Messaging Extended

**Author:** Sam Huffer, 101633177

# Goals / deliverables:

* Extend the previous spike to include one or more of the following:
  + Broadcast messages (specified by the sender)
  + Filtering of messages before delivery / pickup (by the blackboard / dispatch system, not the sender),
  + Scheduling of messages for the future
* You need to produce:
  + Updated design documents as applicable, clearly showing what you have had to add to support your additional features
  + Updated working code demonstration within Zorkish
* Notes:
  + You may like to include the ability for senders to cancel messages to support the above features.
  + Message filtering could be based on game entity values/types, or locations.
  + A message system is often a key part of any combat system. This might be a good target for you.

# Technologies, Tools, and Resources used:

* Visual Studio 2019
* Microsoft Word
* Draw.io

# Tasks undertaken:

* I copied the “Zorkish Adventure” project and the task 18 spike report into the task folder, stripping out the spike report’s original content and replacing it with goals and resources pertaining to the task at hand.
* I had a look through the task instructions and considered what I could build to demonstrate the suggested features. I put together a UML class diagram for the required classes that would need to be added or changed, and planned what order I would tackle required changes in.
* I removed MessageManager’s existing one-size-fits-all Subscribe(), Unsubscribe() and subscribers members, and replaced them with members suited for storing and handling Players, Worlds, Commands, Locations, Paths and Items separately (IMAGE). I then updated SendMessage() to distinguish between types of Message recipients through an enum-using switch statement, and pass the message to objects in the appropriate list (IMAGE).
* I went through World.World() and found all now erroneous calls to MessageManager.Subscribe(), and updated them on a type-by-type basis to use the correct, overloaded Subscribe() method for that type. I removed the Subscribe() call for items, and added Subscribe() and Unsubscribe() calls to Container.AddItem() and RemoveItem() respectively to ensure Items are subscribed under the correct container (IMAGE).
* I went through each class that had a Notify() method and made it or its parent class inherit from Notifiable, with Notify() being a virtual method of Notifiable that just returns a nullptr unless overridden. (IMAGE)
* TODO: update GO to take string id of container game object, update C to set string id to game object when adding item and removing item. Same for paths? Nah, they stay with the same location.

Need to fix one call to SendMessage() that wants to send a message to an item but isn’t sending the id of its container; all such calls should also now use the new field of game object to specify the container MessageManager needs to check in.

* TODO: update commands take, put and drop to unsubscribe and resubscribe items in relation to their new container.
* TODO: assess CommandOpen to see if it can go hand in hand with filtering changes to remove needing fields of a particular object.
* TODO: add Button, Landmine and Flammable Component classes
  + Skeletons
  + Text file specification
  + World.World() handling
  + Internal behaviour

# What we found out:

* Filtering is good for restricting messages to being send-able only to game entities at a particular Location and/or of a particular type.
* Filtered broadcasting is good for triggering behaviours of all game entities of a particular type at a particular Location.

# Task 19 – Messaging Extended – Design Diagram