**Spike:** 15

**Title:** Messaging Systems

**Author:** Ben Holmes, 103024841

**Goals / deliverables:**

1. Design details for the message system (overall architecture, i.e. blackboard or dispatcher), expressed as class/module/sequence diagrams (or equivalent). Include a clear description of your message details.  
   (See notes below). Include the design details in your spike report.
2. A working demonstration that shows how your message system can:
   1. Send/Leave a simple message from one game entity to another (A to B) to change a state.
   2. Send/Leave a message with extra data from one game entity to another (A to B with data)
   3. Send/Leave a message response for the (A to B, B to A response)

**Technologies, Tools, and Resources used:**

* Visual Studio 2022
* Draw.io
* Word
* The Discord

**Tasks undertaken:**

* Use ComponentTest.json to test out the messaging system as it is the one with the components done
* Develop message board
* Create message board functions
* Create message class
* Create message use functions
* Adapt commands to use messages instead

**What we found out:**

I ended up doing the design late as a result of developing spike 14 and 15 simultaneously which was a minor mistake but did not cause me any problems.

Here is the UML diagram for the MessageBoard and the Message classes:

**A black screen with white text

Description automatically generated**

The MessageBoard being used is stored in the Adventure class



And then in the gameInput function it runs the doMessages if areThereMessages() is true

A black screen with white text

Description automatically generated

A computer screen shot of text

Description automatically generated

It loops through all the messages in the vector (if there is more than one) and either runs the output, or sends them off to the entity and deletes them from the list.

A screen shot of a computer code

Description automatically generated

This is the add message function that is used by the entities and commands:

A black background with white text

Description automatically generated

Simply adds a message to the list to be run in doMessages

And here is the entites receive message function:

A screen shot of a computer code

Description automatically generated

Message:

A screen shot of a computer

Description automatically generated

The operrator is required to use the erase(remove()) part of the vector.

Currently 5 types of messages are used:

Success/failure (they are only different for debugging/code readability purposes atm):

From: “entity.name”

To: “output” // sometimes isnt ouput, the check is on type not to

Type: “success” or “failure”

Message: “error or success message”

A success or failure message is always sent back to the msgBoard after an entity runs the final part of a Command (Use has a message sent from one entity to another). The message of the success or failure is outputted to console.

Open:

From: “CommandOpen”

To: “entity.name”

Type: “open”

Message: “Opening entity.name”

Additional: “key” (this is only done when WITH is used, otherwise null)

This is used in the Open command, command code update is here:

A screen shot of a computer program

Description automatically generated

And here is the open component using the message:

A computer screen shot of a program code

Description automatically generated

This was the simplest of the 3 to implement, with no real difficulties.

And here is a example:

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer screen

Description automatically generated

This is deliverable 2a and b

Use:

From: “CommandUse”

To: “entity.name”

Type: “use”

Message: “using entity.name”

Additional: “targetEntity.name”

This is used in the UseCommand:

A black screen with text

Description automatically generated

The message is sent off to the entity with component use and does:

A computer code on a black background

Description automatically generated

Where it grabs the heal or attack components value and makes that the additional before sending it off via the Health message.

A screenshot of a computer

Description automatically generated

A black background with white text

Description automatically generated

This is deliverable 2b and c

Health:

From: “entity.name”

To: “targetEntity.name”

Type: “health”

Message: “entity.name used to attack targetEntity.name” (attack is interchangable with heal

Additional: “value” (currently either 2 or -5)

Health is used in the health component here:

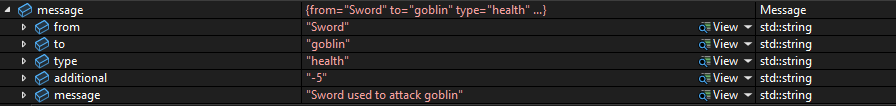
A computer screen with text on it

Description automatically generated

There is a try catch as a just incase (even though it should never be needed in current state) as the string needs to be able to be converted to int for health adjustment.

Health component handles the adjustment of health from the additional value and changes the monsters description to add dead if health drops to <=0 after the change, also prevents the change if health is already <=0.

Example:



A black screen with white text

Description automatically generated

A black screen with white text

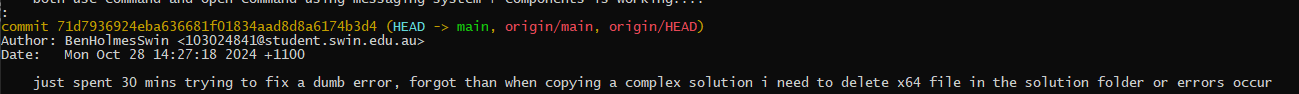
Description automatically generated

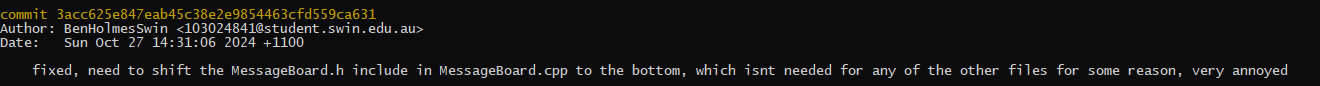
Dificulties:

I ran into a number of build issues with the messageboard with entity not defined.

I tried to fix it for ages and it was all because of 1 circular definition somewhere but it stated so many different error locations for one or two small errors.

This also became a problem when I copied the working in spike 14 version across to spike 15 and ran into it again, all because I forgot to use rebuild/remove the x64 folders.





(the second 1 was because of some incorrect/poorly done forward definitions in enitty.h)

I have tested on another machine transferred via git and it builds correctly, and works once you change the command line argument to use the ComponentTest.json