**Spike:** 10

**Title:** Sprites and Graphics

**Author:** Ryan Chessum, 102564760

**Goals / deliverables:**

* Code, see /19 – Spike – Sprites and Graphics /Messaging/
* Spike Report

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

List of information needed by someone trying to reproduce this work

* Visual Studio 2019
* C plus plus reference (<https://www.cplusplus.com/reference/>)
* Zorkish Specifications

**Tasks undertaken:**

* Download and install Visual Studio
* Create a new C++ project
* Plan out a design for a messaging system
* Import necessary classes
* Create a message class
* Create a blackboard class
* Add update function to entities
* Add code to entities so that they can send/receive messages

**What we found out:**

So far in Zorkish related Spikes I have been using a direct approach to adding interactions between entities by calling member functions in commands or in main. In larger games with many more entities a messaging system of some kind is used to avoid having to couple every game object you want to interact.

One approach to a messaging system is a blackboard. Objects in the game can have a reference to an instance of this blackboard in the main game. Each object with a reference to this will be able to send messages to the board that contain 1) who they are from, 2) who they are addressed to, 3) the message or a message id, and optionally 4) any data they want to send/receive.

I made a demo to try and create a simple blackboard that could be used for Zorkish.

First I came up with a desing for how this is going to work.

Diagram, timeline

Description automatically generated

The Blackboard will hold a vector of messages. Objects that have a reference to it can call the AddMessage() function to add a message to the vector. Messages contain 3 strings for the sender, receiver, and message as well as a void pointer to send data. To look for messages, objects with a reference to the blackboard can call GetMessage() and pass in a vector of strings. The function will iterate through the vector to find any messages that contain one of the strings in the vector for the ‘to’ string. Then it will return a vector of messages that match. There is also a function to remove messages which objects can call to remove a message after they are done with it. The clear() function is for use when the program is closed. It frees all of the memory for the pointers in messages just in case the game was closed before the memory could be deallocated.

Text

Description automatically generated

Text

Description automatically generated

I then created three entities to use the blackboard. A player, a chest and an NPC.

Text

Description automatically generated

Every entity has an update and render function which is called in the game loop.

Text

Description automatically generated

The players update function has some basic commands that they can use. Open will send a message to open a chest. If the player uses say and writes hi it will send a message to all npcs that the player said a greeting and pass in the player name as data. The purchase command will request to “purchase” an item from the shopkeeper greg. It will send a string for the item the player wants to buy as data.

Text

Description automatically generated

The chest class will search for messages addressed to it. If it finds a message to open it will set its status to open and remove the message.

Text

Description automatically generated

The npc will also search for messages addressed to it. If it finds a greeting addressed to npcs it will send a reply in the text output using the players name. The void pointer to the data is cast as a string pointer. Just in case we somehow sent the wrong datatype in the message, there is a try and catch statement to handle errors for us. Ideally it should always work as intended but just in case it somehow fails it will catch the error for us and stop the program from crashing.

Purchase also works similarly. It casts the void pointer as a string to use. If the player asks for a sword it will create a new Item and send a message to the player to receive the item with the sword passed in as data.

Text

Description automatically generated

Back in the player main function if the player gets a message to receive an item it will cast the pointer as an item class then put it in the players inventory. If it fails, the data will be deleted.

Heres what happens when we try and use all the commands.

Text

Description automatically generated