Spike: Task19
Title: Message System

Author: Hayden Whiteford, 104001272

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
a. Send/Leave a simple message from one game entity to another (A to B) to change a state. b. Send/Leave a message with extra data from one game entity to another (A to B with data) c. Send/Leave a message response for the (A to B, B to A response)

Technologies, Tools, and Resources used:

- Xcode
- draw.io (for the diagram)

Tasks undertaken:

- Created a simple plan for a Post Office style messaging system
- Created a Component Abstract Class (as an improvement over my zorkish code)
- Created a PostOffice class that can read in and execute messages
- Created 4 tests in the main file:
    - The player sending a message to open a box
    - The player sending a message to a light switch, that then sends a message to a light bulb to glow
    - The player sending a message to an ATM to check the balance of a card number, and the ATM sending a message back to the player with the balance of the provided card number

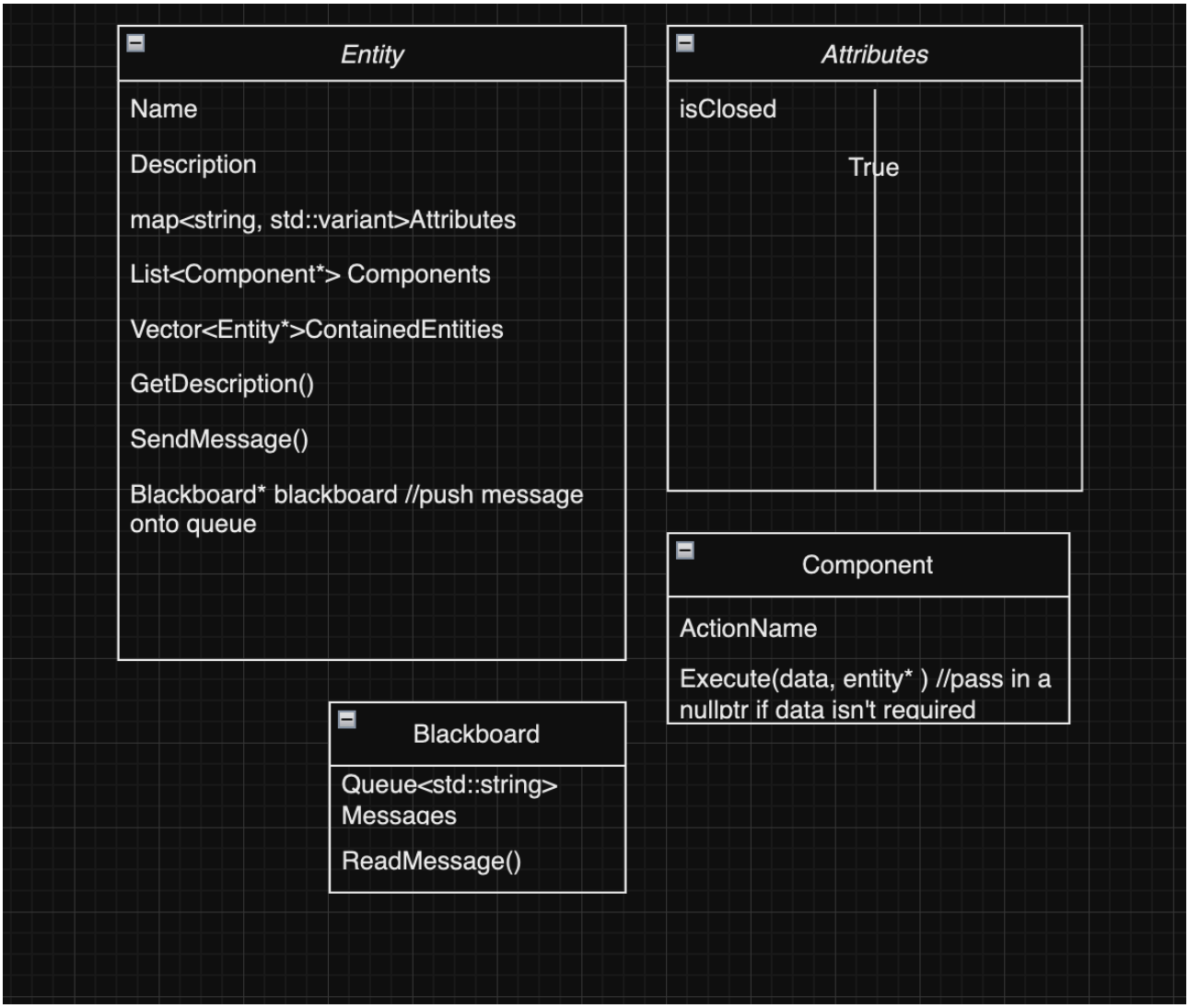
What we found out:

I originally had named the Post Office a “BlackBoard”. I Later found out this was incorrect due to how each system operates. For a BlackBoard system, each entity would need to be “polling” the blackboard for new messages, and be checking whether those messages were relevant to them. With a “dispatcher” or “Post Office” style system, the message is read and executed by the dispatcher, which I found simpler for this task.

Open issues/risks:

This task is a somewhat rudimentary version of what would actually be implemented in a zorkish game. A lot of the setting up of the entities, attributes, and post office is done in just the main file, which is honestly a little silly, and with more time I would build most of the set up into the constructors themselves.

Plan:


Entity
Name
Description
map<string, std::variant>Attributes
List<Component*> Components
Vector<Entity*>ContainedEntities
GetDescription()
SendMessage()
Blackboard* blackboard //push message onto queue
Attributes
isClosed
True
Component
ActionName
Execute(data, entity*) //pass in a nullptr if data isn't required
Blackboard
Queue<std::string> Messages
ReadMessage()