

Task 19 - Spike Summary Report



Spike: Task_19

Title: Messaging and Blackboards **Author:** Thomas Horsley, 103071494

Goals & Deliverables

Aim: Develop a messaging system with either a dispatch or blackboard architecture. This system must be extensible enough to be implemented into Zorkish.

Deliverables:

- Functioning message dispatch system
- Spike summary report
- Git commit history

Technology, Tools and Resources

Tech and Tools



The project was scripted in C++ 17 using Visual Studio Community 2022.

UML's and charts are made with www.Lucidchart.com

Source control is handled using Git.

Resources

• Echo360 Lecture "Topic 7.1 - Messaging Systems and Architectures"



Task 19 - Spike Summary Report

Class Descriptions and Notes

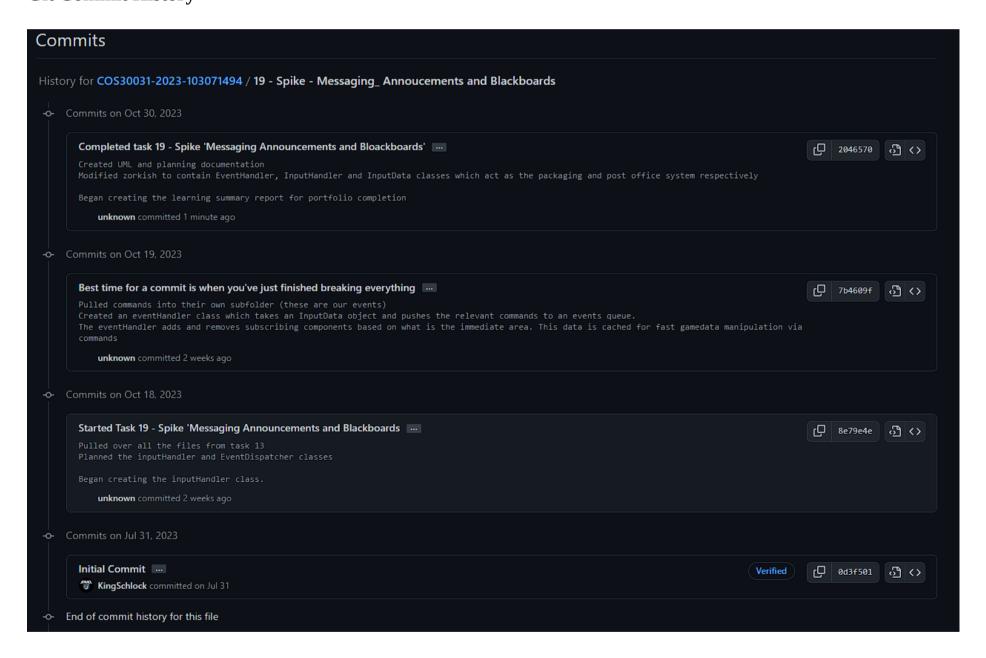


For the "post-office" structure, I've chosen to implement a dispatcher for the messages rather than a blackboard as I believe this will suit my pre-existing code better.

Three notable extensions to Zorkish are the InputHandler, InputData and EventDispatcher classes. These objects provide us the tools to take input, format the input into a message object capable of being used as a context for the commands, and dispatch events when necessary.

Implementation

Git Commit History



```
pclass EventDispatcher {
             GameData* _game_data = nullptr;
            MoveCommand* _move_command = nullptr;
             TakeCommand* _take_command = nullptr;
            LookCommand* _look_command = nullptr;
             ShowCommand* _show_command = nullptr;
            QuitCommand* _quit_command = nullptr;
            EventDispatcher();
             ~EventDispatcher(); // Delete commands
             void setGameData(GameData* game_data);
             void filterLocalComponents();
             void resetComponents();
            // Call the onEvent() method for the relevant commands parsing the relevant args
std::queue<Command*> processEvents(InputData* input_data);
             void getEntityComponents(const char UEID);
             C_Render* getRenderer(const char UEID);
            C_Inventory* getInventory(const char UEID);
std::vector<C_Portal*> getPortals(const char UEID);
             std::string getExitUEIDFromDir(std::string direction);
77
```

EventDispatcher acts as the post office for our messaging system.

The message context is pre-formatted and contained within the args. With other data being stored independently.

Additionally, each command has been modified to take an InputData message to provide context for it's triggerEvent() method.

2

The InputHandler object takes the user input and is responsible for formatting the input appropriately.

Once formatted, the InputHandler will provide the EventDispatcher with the InputData.

```
#pragma once
     ⊟class InputHandler {
      private:
          InputData* _input_data = nullptr;
      public:
          InputHandler(InputData* input_data = nullptr);
          ~InputHandler();
          InputData* handleInput(std::stringstream& raw_input, GameData* game_data);
          void resetInputData();
          CommandType validateCommandType(std::string raw_c_type);
          std::vector<std::string> formatArgsForType(std::string raw_c_type,
              std::vector<std::string> args, GameData* game_data);
          std::string getItemName(std::vector<std::string> args, std::string safety_word = "");
          bool takeOrDrop(std::string take_modifier);
          std::string formatTakeOrDrop(std::string raw_c_arg);
          std::string extractUEID(std::string ucid);
27
```

The GameplayState update() method utilizing the messaging system.



Task 19 - Spike Summary Report

What was Learned?



This Spike involved understanding the "Post Office" pattern and implementing a messaging system robust enough to support the basic ECS and command pattern structure within my Zorkish implementation.

Throughout the duration of the project, I found that similar messaging systems are extensively used in both games and game engines for allowing communication between largely decoupled components and systems. I plan to use a more robust equivalent system within my own custom project.