Spike: Task 12
Title: Command Pattern

Author: Hayden Whiteford, 104001272

Goals / deliverables:

1. The **loading of an adventure file** (text format) from the “Select Adventure” stage (state pattern).

2. The adventure file format needs to include support for game locations (earlier spike) and extended to support NEW **game entities** (rocks, flowers etc).
    - You will need to add these new “entities” to the world graph work you did earlier.
    - Commands will look at, but will not move or change, these entities.
3. A robust **command processor** supporting a minimum of the following commands
    - GO to change location. (Use an alias later to remap “MOVE” to GO.)
    - HELP to lists known commands and their syntax details.
    - INVENTORY to list what the player has.
    - LOOK, LOOK AT (but not LOOK IN yet) to show details of location or entities.
    - ALIAS to remap commands to alternative command strings.
    - DEBUG TREE to show debug-useful details of the game graph world and entities.
4. A **UML diagram** that matches your final command pattern-related classes. Include this in your spike report.

Technologies, Tools, and Resources used:

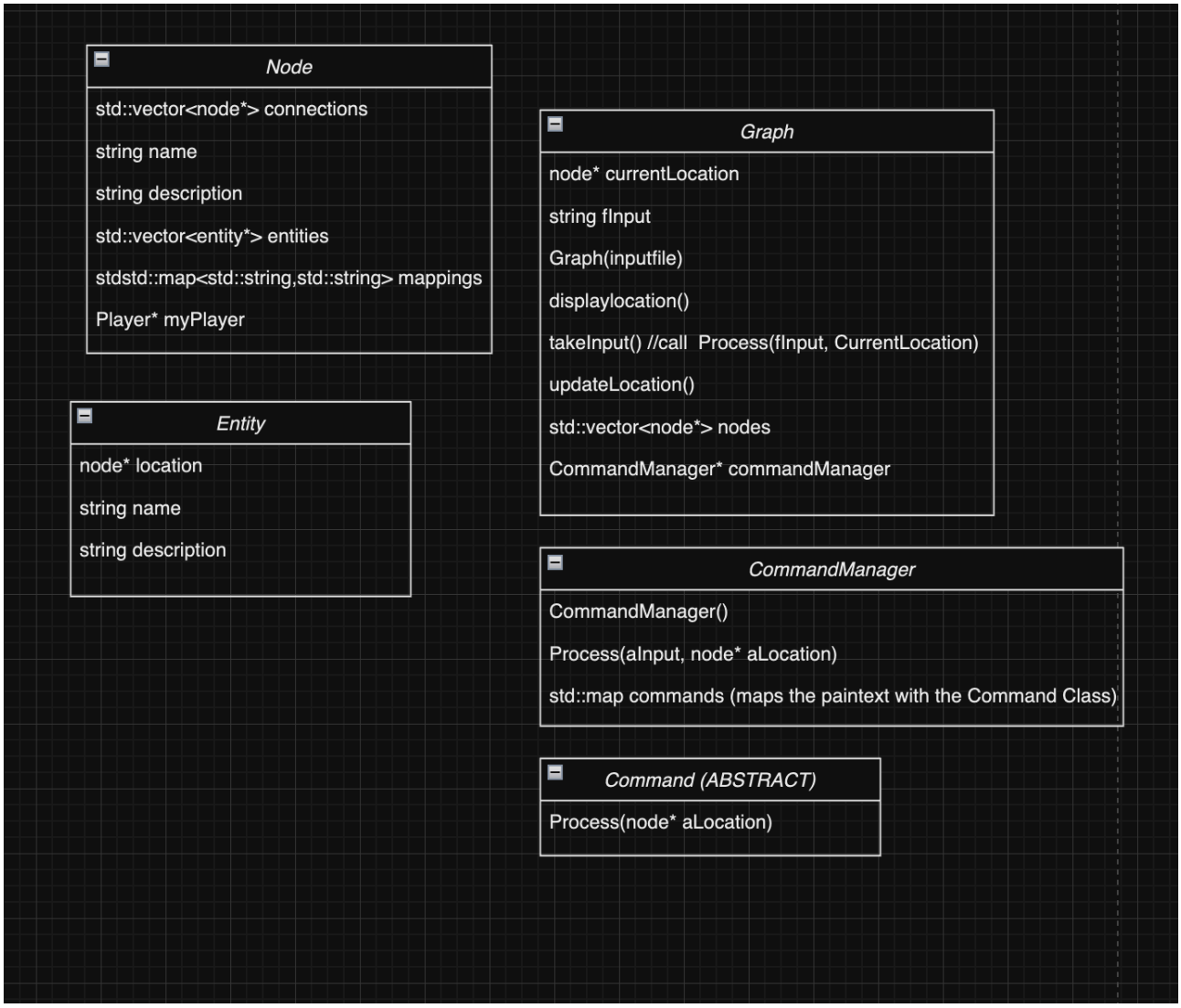
- draw.io **- UML Plan**
- Xcode

Tasks undertaken:

- Created a UML plan
- Created the Entities class
- Updated the node class to support entities, the player reference, and a new “mappings” map for translating directions to connected locations
- Updated the Graph class constructor to read in entities and mappings from the locations.txt file
- Created a CommandManager class
- Created a Command abstract class
- Created various other classes that inherit from the Command class (CommandMove,CommandHelp, CommandLook etc.)

What we found out:

The usefulness of the Command structure. Its similar to both the node and graph structures, and the game state structures, in that they are all a series of objects that dictate certain states, that are all being managed by a “manager” class. By considering all of the commands as objects, the code becomes far more modular and scalable for future coding endeavours. Neat!


Node
std::vector<node*> connections
string name
string description
std::vector<entity*> entities
stdstd::map<std::string,std::string> mappings
Player* myPlayer
Graph
node* currentLocation
string fInput
Graph(inputfile)
displaylocation()
takeInput() //call Process(fInput, CurrentLocation)
updateLocation()
std::vector<node*> nodes
CommandManager* commandManager
Entity
node* location
string name
string description
CommandManager
CommandManager()
Process(aInput, node* aLocation)
std::map commands (maps the paintext with the Command Class)
Command (ABSTRACT)
Process(node* aLocation)