Extension: Task 16
Title: Configuration Files
Author: Sam Huffer, 101633177

Goals / deliverables:

- An expanded world / adventure file which should:
  - Be easy to produce with different values; demonstrate by showing at least two working configurations.
  - Go beyond graph implementation to include one of the following:
    - Entities (items, containers, etc.)
    - Commands
    - A unique idea confirmed with your tutor

Technologies, Tools, and Resources used:

- Visual Studio 2019
- Microsoft Word
- Draw.io

Tasks undertaken:

- I copied the "Zorkish Adventure" project and the task 15 spike report into the task folder, stripping out the spike report's original content and replacing it with goals and resources pertaining to the task at hand.
- I had a look at the extension's requirements, and found that I had already implemented the first suggestion in earlier spikes (fig. 1, fig. 2), specifically Task 12 – Game Graphs from Data, as I saw no

```
#Test specification for Zorkish Adventures

##Formatting and Notes--------------------------------------------------------------------------

#World Name          "W:World Name"

#Starting Location   "S:starting_location_id"

#Locations           "L:location_id:Location Name:location description".
#                    Default components: container, description.

#Paths               "P:path_id:Path Name:location_from_id:direction:location_to_id:path description".
#                    Default components: description.
#                    If you leave path_id or Path Name blank, it'll construct them from location_from_id and location_to_id.

#Direction Aliases   "A:direction name:direction alias"

#Items               "I:container_id:item_id:Item Name:item description:component_id_1,component_id_2, . . .".
#                    Default components: description.

#Components          "C:component_type:game_object_id[:component-specific parameters]"
#                        "C:container:game_object_id:is_open?:always_open?"
#                        "C:description:game_object_id:description of game object"
#                        "C:lock:game_object_id:is_locked?:unlockable_with_item_id_1,unlockable_with_item_id_2, . . .".
#                            If not unlockable, specify "none".
#                        "C:movable:game_object_id"

#WARNING: don't specify lock or description components in an item's basic specification; ComponentFactory won't return them
#as it can't handle extra parameters.

#Locations, Paths, Items and the Player are game objects

##World Name-----------------------------------------------------------------------------------

W:Test World

##Locations------------------------------------------------------------------------------------

#Starting location: the void
L:void:The Void:an empty, formless void.
S:void
I:void:strongbox_key:Strongbox Key:A key for a strongbox.:movable
I:void:strongbox:Strongbox:A strange-looking strongbox.:none
C:container:strongbox:No:No
C:lock:strongbox:Yes:strongbox_key
C:movable:strongbox
```

Figure 1: My existing Test World specification, which outlines how to specify various game entities to be loaded into Zorkish Adventures, then features examples of specifying the world name, starting location, items, and components.

```
#The attic full of items
L:east_attic:The Attic:a dim attic.
I:east_attic:book:Book:A dusty old book.:movable
I:east_attic:pencil:Pencil:A short, used pencil.:movable
I:east_attic:glasses:Glasses:A pair of glasses.:movable
I:east_attic:quill:Quill:A black and grey quill.:movable
I:east_attic:red_bag:Bag:A red bag.:movable,container
I:east_attic:wardrobe:Wardrobe:An old wardrobe.:container
I:east_attic:boulder:Boulder:A boulder someone must have hauled up the ladder. It's not going anywhere.:none
I:red_bag:gold_coin:Gold Coin:A gold coin. This is valuable.:movable
I:red_bag:silver_coin:Silver Coin:A silver coin. This is worth a bit.:movable
I:red_bag:copper_coin:Copper Coin:A copper coin. This isn't worth much.:movable
I:red_bag:small_box:Box:A small wooden box.:movable,container

#The north room, which leads to the cellar
L:north_room:The North Room:an empty room.

#The dark cellar
L:north_cellar:The Cellar:a dark cellar.

##Paths----------------------------------------------------------------------------------------

#Paths from void
P:::void:forward:south_room:A glowing white . . . something . . . in front of you . . . slowly moving FORWARD.

#Paths from south_room
P:::south_room:north west:west_room:A door to the NORTH WEST.
P:::south_room:north:middle_room:A door to the NORTH.
P:::south_room:north east:east_room:A door to the NORTH EAST.
```

Figure 2: Another excerpt from my existing Test World specification, which specifies locations, items and their components, and some paths between locations. Some of the items are placed in locations, others in other items.

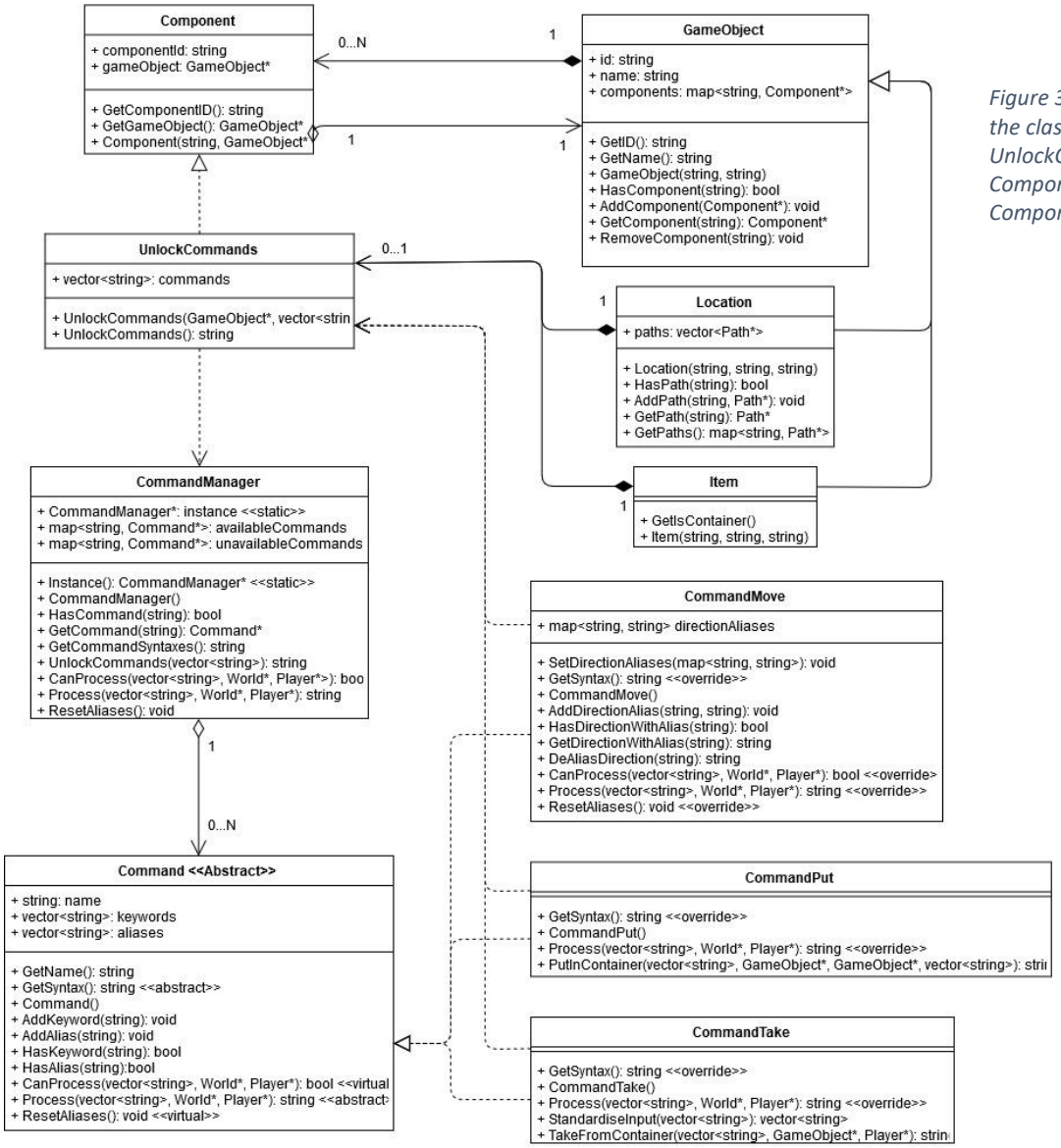

Figure 3: a UML diagram of the classes relevant to an UnlockCommands Component, as well as the Component itself.

sense in specifying locations without specifying the items in those locations when the commands for moving them about had already been created for Task 10 – Game Data Structures, and Task 15 – Composite and Component Patterns, which required the implementation of the component pattern. As all game objects were being specified in the text file already, I saw no reason not to add components to the specification as well.

- I emailed Tien, my tutor, about what I could do for this task, and settled on specifying commands to be unlocked when a player gets to a particular location or adds a particular item to their inventory. As such, I put together a UML outlining how that might be done (fig. 3).
- I updated the text file specification to accommodate the UnlockCommands Component.

```
#Starting Configuration     "S:starting_location_id:available_command_id_1,available_command_id_2, . . ."
#                           Command types: all, alias, debug, look, open, move, take, put, drop, help, hiscore, quit
#                           Default commands: look, help, quit
#                           At least move or take must be made available from the start, so that the player can move to locations
#                               that unlock further commands, or pick up items that unlock further commands. In the latter case,
#                               at least the move command must be unlocked in that manner.

#Locations                  "L:location_id:Location Name:location description".
#                           Default components: container, description.

#Paths                      "P:path_id:Path Name:location_from_id:direction:location_to_id:path description".
#                           Default components: description.
#                           If you leave path_id or Path Name blank, it'll construct them from location_from_id and location_to_id.

#Direction Aliases          "A:direction name:direction alias"

#Items                      "I:container_id:item_id:Item Name:item description:component_id_1,component_id_2, . . .".
#                           Default components: description.

#Components                 "C:component_type:game_object_id[:component-specific parameters]"
#                               "C:container:game_object_id:is_open?:always_open?"
#                               "C:description:game_object_id:description of game object"
#                               "C:lock:game_object_id:is_locked?:unlockable_with_item_id_1,unlockable_with_item_id_2, . . .".
#                                   If not unlockable, specify "none".
#                               "C:movable:game_object_id"
#                               "C:unlock_commands:game_object_id:command_id_1,command_id_2, . . ."
#                                   Should only be attached to Locations and/or Items.
```

Figure 4: the updated text file specification. The starting configuration now lets users list Commands (or all Commands) to be available from the start, and an UnlockCommands Component can be attached to GameObjects as a custom Component.

- I created the UnlockCommands Component's .h and .cpp files, although I swapped out the non-constructor UnlockCommands() method for a GetCommands() public property. I then added to CommandManager the UnlockCommands() method, which iterates over each command id, checks unavailableCommands for a matching command (or just accepts it if the id listed is "all"), adds it to availableCommands if that Command type isn't already in availableCommands, adds to an output string a "Command Unlocked" message listing the new command, and removes the Command from unavailableCommands. If the Command is already in availableCommands or does not exist in unavailableCommands, an appropriate error message is appended instead.

```
std::string CommandManager::UnlockCommands(std::vector<std::string> commands)
{
    std::string result;
    std::vector<std::string> unlockedCommands = std::vector<std::string>();

    for (std::pair<std::string, Command*> pair : unavailableCommands)
    {
        for (std::string command : commands)
        {
            if (command == pair.first || command == "all")
            {
                if (HasCommand(pair.first))
                {
                    result += "Error: Command " + pair.second->GetName() + " is already unlocked.\n";
                }
                else if (!unavailableCommands.count(pair.first))
                {
                    result += "Error: Command " + pair.second->GetName() + " is not available to be unlocked.\n";
                }
                else
                {
                    unlockedCommands.push_back(pair.first);
                    availableCommands[pair.first] = pair.second;
                    result += "New command unlocked: command " + pair.second->GetName() + ".\n";
                }
            }
        }
    }

    for (std::string command : unlockedCommands)
    {
        unavailableCommands.erase(command);
    }

    return result;
}
```

Figure 5: CommandManager.UnlockCommands()

- I added to CommandMove.Process() a check when the player moves to a new Location for whether or not that Location has an UnlockCommands Component, passing its Command ids to CommandManager.UnlockCommands() and printing the result if so, before deleting the Component.
- I modified "Test World.txt" such that the starting location only had the default Commands (look, help, quit) and move available, and that the next location the player would reach would unlock the remaining commands.
- I modified World.World()'s handling of the starting configuration to validate that starting commands were specified, to unlock commands specified, and verify that the commands were all unlocked, printing an error message if they were not unlocked.
- To allow printing of correct formatting for any line or object at any point in the file loading loop, I copied an instance of each set of formatting into a string variable outside of World.World()'s while loop, and replaced each instance of that formatting with that string variable.
- I added to the components checks a set of checks for the UnlockCommands Component, checking the appropriate information was entered in the read line of the text file, before constructing a new UnlockCommands component with the list of Command ids, and adding it to the specified GameObject.

```
else if (splitLine[1] == "unlock_commands")
{
    if (splitLine.size() != 4)
    {
        std::cout << "Error, \"" << filename << "\", line " << lineCount << ": Wrong number of values for loading unlock commands component. Values required (including pre
        std::cout << generalComponentFormat;
        std::cout << unlockCommandsComponentFormat;
        loadedSuccessfully = false;
    }
    else if (splitLine[3].length() == 0)
    {
        std::cout << "Error, \"" << filename << "\", line " << lineCount << ": You must specify the commands to be unlocked, or \"all\" if the remaining commands are to al
        std::cout << generalComponentFormat;
        std::cout << unlockCommandsComponentFormat;
        loadedSuccessfully = false;
    }
    else
    {
        gameObjects[splitLine[2]]->AddComponent((Component*) new UnlockCommands(gameObjects[splitLine[2]], StringManager::Instance()->StringToVector(splitLine[3], ',')));
    }
}
else
{
    std::cout << "Error, \"" << filename << "\", line " << lineCount << ": Invalid component type \"" + splitLine[1] + "\". No component exists of that type.\n";
    std::cout << generalComponentFormat;
    loadedSuccessfully = false;
}
```

Figure 7: World.World()'s checks for the UnlockCommands Component.

- I updated CommandPut.PutInContainer() and CommandTake.TakeFromContainer()if the item moved had an UnlockCommands Component if it was put in the player's inventory. However, I found PutInContainer()'s containerTo parameter is a GameObject*, not a Player*, giving no way of knowing if the item is being put in the player's inventory. To fix this, I added to GameObject a public property GetType() to allow for this to be checked, and made GameObject's constructor require child classes to specify their type.
- I modified Command.Command() to require the Command's name to be passed as a parameter, and then made sure each Command passed its name capitalised. I also tweaked the formatting of the unlocked commands messages to clean them up a little bit.
- I added to CommandManager.GetAllSyntaxes() a bool parameter to allow for differentiation between retrieving only available commands' syntaxes, and retrieving all commands' syntaxes, and then made CommandHelp retrieve only available commands but made StageHelp retrieve all.
- To demonstrate the new UnlockCommands Component more fully, and because I need to submit 2 world files, I created a second world, "Dungeon World.txt" (fig. 8), and added it to "Worlds.txt".
- While loading Dungeon World in World.World(), I found that I had neglected to put customly-created Container Components in the containers map, so I added a line to add the newly created container to the map. I also found that the check for valid container GameObjects was printing the wrong part of the input string as the GameObject's id, so I fixed that to output the ids properly.

- I also found that I had neglected to add a check that the specified starting location was a valid location that had already been created, so I added a check to check if the id provided was that of a location in World.World().locations, throwing an appropriate error message if it was not.

```
#Starting location: empty Dungeon 1
L:dungeon_1:Dungeon 1:a cold, dark dungeon with a 1 painted on the wall.
S:dungeon_1:move,alias,debug,hiscore

#Dungeon 2: unlock take
L:dungeon_2:Dungeon 2: a cold, dark dungeon with a 2 painted on the wall.
C:unlock_commands:dungeon_2:take
I:dungeon_2:rock:Rock:A large, heavy rock.:none
I:dungeon_2:book:Book:A dusty old book.:movable

#Dungeon 3: unlock drop, open container with unlock put in
L:dungeon_3:Dungeon 3:a cold, dark dungeon with a 3 painted on the wall.
C:unlock_commands:dungeon_3:drop
I:dungeon_3:red_bag:Bag:A red bag.:movable
C:container:red_bag:Yes:No
C:unlock_commands:red_bag:put
I:red_bag:gold_coin:Gold Coin:A gold coin. This is valuable.:movable
I:red_bag:silver_coin:Silver Coin:A silver coin. This is worth a bit.:movable
I:red_bag:copper_coin:Copper Coin:A copper coin. This isn't worth much.:movable

#Dungeon 4: unlock open with container
L:dungeon_4:Dungeon 4: a cold, dark dungeon with a 4 painted on the wall.
I:dungeon_4:box:Box:A small, wooden box.:movable,container
C:unlock_commands:box:open

#Dungeon 5: unlock chest with key
L:dungeon_5:Dungeon 5: a cold, dark dungeon with a 5 painted on the wall.
I:dungeon_5:iron_key:Iron Key:A small, dull, iron key.:movable
I:dungeon_5:wooden_chest:Wooden Chest:A large, heavy-looking wooden chest.:container
C:lock:wooden_chest:Yes:iron_key

##Paths and any of their custom components----------------------------------------

#Paths from dungeon_1
P:::dungeon_1:north:dungeon_2:A long corridor NORTH, dimly lit with torches.

#Paths from dungeon_2
P:::dungeon_2:north:dungeon_3:A long corridor NORTH, dimly lit with torches.
P:::dungeon_2:south:dungeon_1:A long corridor SOUTH, dimly lit with torches.

#Paths from dungeon_3
P:::dungeon_3:north:dungeon_4:A long corridor NORTH, dimly lit with torches.
P:::dungeon_3:south:dungeon_2:A long corridor SOUTH, dimly lit with torches.

#Paths from dungeon_4
P:::dungeon_4:north:dungeon_5:A long corridor NORTH, dimly lit with torches.
P:::dungeon_4:south:dungeon_3:A long corridor SOUTH, dimly lit with torches.

#Paths from dungeon_5
P:::dungeon_5:south:dungeon_4:A long corridor SOUTH, dimly lit with torches.
```

Figure 7: The specification for the new Dungeon World. It's simpler than the Void World, but it does a better job of showcasing the new UnlockCommands Component, and its uses with Items and Locations for unlocking commands

What we found out:

- C++ doesn't like non-constructor methods with the same name as the class's constructor.
- Having a base class require a particular value as a parameter in its constructor is an easy way of ensuring all of its derived classes do provide such a value. Setting up the base constructor and then building is a quick an easy way of finding all the cases where either the base constructor needs to be explicitly added, or the parameter needs to be added.
- If a manager class has multiple lists of the same class of object, say, for objects that are available to the player and others that are not, it's good to have a master list of all of the objects in case that needs to be accessed rather than the lists of available or unavailable objects.
- I probably should have tested that the error checks for various game world components were working properly both in their accurately checking that there was an error and their outputting of error messages.
- Restricting when players gain access to particular actions is a good way of building up players' familiarity with those actions. Locations seem to be the more player-friendly GameObject to use for this, as the player can be forced to go through them, whereas the player may has to use the take or put in commands to pick up an Item before they can gain its affordances.