Spike: Task 12
Title: Game Graphs from Data
Author: Sam Huffer, 101633177

Goals / deliverables:

- The specification for a text-file format for representing (at this stage) the details of a world, its locations, and the connections between them.
- Code that can:
    - Load the world from a text file from the Select Adventure stage
    - Store the world data in your program.
    - Allow players to move between locations in the world, with more directions than just "North", "East", etc.

Technologies, Tools, and Resources used:

- Visual Studio 2019
- Microsoft Word

Tasks undertaken

- I copied the spike report template into the task folder, stripping out the original content and replacing it with goals and resources pertaining to the task at hand.
- I created a blank Visual Studio project "Zorkish Adventure" and copied in all the source files used in Task 10: Game Data Structures, so that for future tasks I wouldn't have any folders or files named for previous tasks to worry about renaming.
- I wrote out the specification for recording the details of a world, its locations and its items in a text file that could be loaded into the game, as well as how to format a list of available worlds and their text files.
- I designed a basic world with a room for each cardinal direction, one in the middle of all of them, a cellar under the north room, an attic above the east room, and a void that one enters from the west room and exits from into the south room. I then typed up the details of the world in a text file, adding the items used in the previous void test world to the specification. Next, I added a text file for listing all available worlds, listing the test world in it.
- I reconfigured SelectAdventure.Setup() to read in the text file "Worlds.txt" to ascertain the worlds available to the player and print them out instead of a hard-coded list of worlds.
- I configured World's constructor to read in the lines in a file (ignoring blanks and comments) and split the text by colons. I added a set of if statements that check that lines that are read in have valid prefixes and the right number of details (that are filled in) for each prefix, and that those details refer to objects that have already been instantiated earlier in the file, producing error messages pinpointing where an error has been made by filename, line, and what the error is.
- I added code for creating objects according to properly formatted lines read from the text file. For this, I added a Path class that holds a description and a pointer to its destination, all to be stored in Location.paths. However, Visual Studio displayed a weird error that I couldn't convert a type's lvalue to that parameter type, so I had to change Path.destination to be the id of the location the path leads to rather than a pointer to the path itself.
- Once I finished putting together the code to create objects of the specified type, I ran the game with the test world that I had created, and identified errors in formatting in the text file, as well as bugs in the World class's constructor. I fixed those, and reran the code, and it worked as planned. When I quit the world to return to the main menu, and went back into the world, however, the options of available worlds had duplicated, so I modified the code to replace the list of world names and world file names every time SelectAdventure.Setup() is called, which fixed the issue. Next, I tweaked some of the output formatting.

- I added the ability for players to see paths when they look at their current location, and moved all code for the results of looking at one's location or inventory into their own private methods to avoid having to recode that multiple times.
- I added the ability for players to move around according to the directions that a location's paths lead in. Once I had that working, I added to the text file specification a formatting option for aliases of directions, I added to "Test World.txt" direction aliases in line with that formatting, I added to World's constructor checks to process and list aliases, I added to World the methods HasDirectionWithAlias() and GetDirectionWithAlias(), I added to Gameplay.Move() the capacity to swap out a direction alias for the direction it represents, and I added to Gameplay.Update() to handle moving when only specifying the direction or a direction alias.

Text-File Format

List of Worlds (Worlds.txt)

The list of worlds formatting is very simple. Each line is a world, listing its name and filename, separating the two with a ":". For example:

"World Name:world_filename.txt"

would have "World Name" displayed in the SelectAdventure stage, while "world_filename.txt" would be passed to the constructor of the World class to draw its specification from there.

Worlds (e.g. "Test World.txt")

The formatting for a world's details are a bit trickier. Again, different details are separated by ":"'s, but here different pieces of information are given prefixes to identify what they entail:

- #Comment ==> Ignore
- *(blank line; ignore)*
- W:World Name
- L:location_id:location_name:location description
- C:container_item_id:container_item_name:container item description:container_id
  (Note: container_id is what it resides within, whether a location or another container item.)
- I:item_id:item_name:item description:container_id
- P:location_from_id:direction:location_id_to:path description
  (Note: this lists the connections or pathways that a location (location_from_id) has to other locations (location_to_id), which are stored in the <string, string> map Location.neighbours, hence the prefix "N".)
- S:starting_location_id
- A:direction name:direction alias

Screenshots

```
locations = std::map<std::string, Location*>();
directionAliases = std::map<std::string, std::string>();
loadedSuccessfully = true;
name = "";
currentLocation = nullptr;

//Read available worlds from file
std::ifstream ifs(filename);

if (ifs.is_open())
{
    std::string line;
    std::vector<std::string> splitLine;
    std::map<std::string, Container*> containers = std::map<std::string, Container*>();
    int lineCount = 0;

    while (std::getline(ifs, line))
    {
        lineCount++;

        //Check for blank lines and comments
        if (line.length() == 0 || line[0] == '#')
        {
            continue;
        }

        splitLine = StringManager::Instance()->StringToVector(line, ':');

        //Check prefixes
        if (splitLine[0] == "W") { ... }
        else if (splitLine[0] == "S") { ... }
        else if (splitLine[0] == "L") { ... }
        else if (splitLine[0] == "P") { ... }
        else if (splitLine[0] == "A") { ... }
        else if (splitLine[0] == "C") { ... }
        else if (splitLine[0] == "I") { ... }
        else { ... }
    }

    if (name == "")
    {
        std::cout << "Error: No world name was provided.\n";
        std::cout << "\tFormat: \"W:World Name\".\n";
    }

    if (locations.size() == 0)
    {
        std::cout << "Error: No locations were provided.\n";
        std::cout << "\tFormat: \"L:location_id_:Location Name:location description\".\n";
    }

    if (currentLocation == nullptr)
    {
        std::cout << "Error: No starting location was provided.\n";
        std::cout << "\tFormat: \"S:starting_location_id\".\n";
    }
}
else
{
    std::cout << "Error: Unable to open file \"" + filename + "\".\n";
    loadedSuccessfully = false;
}

if (!loadedSuccessfully)
{
    std::cout << "Error: Could not load world from \"" + filename + "\" successfully.\n\n";
}

ifs.close();
```

Figure 1: Outline of World.World()

```
else if (splitLine[0] == "L")
{
    //Check Formatting
    if (splitLine.size() != 4)
    {
        std::cout << "Error, \"" << filename << "\", line " << lineCount << ": Wrong number of values for loading location. Value
        std::cout << "\tFormat: \"L:location_id_:Location Name:location description\".\n";
        loadedSuccessfully = false;
    }
    else if (splitLine[1].length() == 0)
    {
        std::cout << "Error, \"" << filename << "\", line " << lineCount << ": You must specify the location's id.\n";
        std::cout << "\tFormat: \"L:location_id_:Location Name:location description\".\n";
        loadedSuccessfully = false;
    }
    else if (splitLine[2].length() == 0)
    {
        std::cout << "Error, \"" << filename << "\", line " << lineCount << ": You must specify the location's name.\n";
        std::cout << "\tFormat: \"L:location_id_:Location Name:location description\".\n";
        loadedSuccessfully = false;
    }
    else if (splitLine[3].length() == 0)
    {
        std::cout << "Error, \"" << filename << "\", line " << lineCount << ": You must specify the location's description.\n";
        std::cout << "\tFormat: \"L:location_id_:Location Name:location description\".\n";
        loadedSuccessfully = false;
    }
    else
    {
        locations[splitLine[1]] = new Location(splitLine[1], splitLine[2], splitLine[3]);
        containers[splitLine[1]] = (Container*)locations[splitLine[1]];
    }
}
```

Figure 2: Example of one of the sets of nested if statements for validating a line of the text file and constructing the appropriate object if it's valid. This one checks if the location was specified properly.

```
else if (splitLine[0] == "I")
{
    //Check Formatting
    if (splitLine.size() != 5)
    {
        std::cout << "Error, \"" << filename << "\", line " << lineCount << ": Wrong number of values for loading item. Values required (including pr
        std::cout << "\tFormat: \"I:item_id:Item Name:item description:container_id\". (container_id: the location or container item the item is bein
        loadedSuccessfully = false;
    }
    else if (splitLine[1].length() == 0)
    {
        std::cout << "Error, \"" << filename << "\", line " << lineCount << ": You must specify the id of the container (location or container item) t
        std::cout << "\tFormat: \"I:item_id:Item Name:item description:container_id\". (container_id: the location or container item the item is bein
        loadedSuccessfully = false;
    }
    else if (splitLine[2].length() == 0)
    {
        std::cout << "Error, \"" << filename << "\", line " << lineCount << ": You must specify the item's id.\n";
        std::cout << "\tFormat: \"I:item_id:Item Name:item description:container_id\". (container_id: the location or container item the item is bein
        loadedSuccessfully = false;
    }
    else if (splitLine[3].length() == 0)
    {
        std::cout << "Error, \"" << filename << "\", line " << lineCount << ": You must specify the item's name.\n";
        std::cout << "\tFormat: \"I:item_id:Item Name:item description:container_id\". (container_id: the location or container item the item is bein
        loadedSuccessfully = false;
    }
    else if (splitLine[4].length() == 0)
    {
        std::cout << "Error, \"" << filename << "\", line " << lineCount << ": You must specify the item's description.\n";
        std::cout << "\tFormat: \"I:item_id:Item Name:item description:container_id\". (container_id: the location or container item the item is bein
        loadedSuccessfully = false;
    }
    //Check the item's container exists
    else if (!containers.count(splitLine[1]))
    {
        std::cout << "Error, \"" << filename << "\", line " << lineCount << ": The item's container_id must be the id of a container (i.e. location o
        std::cout << "\tFormat: \"C:container_item_id:Container Item Name:container item description:container_id\". (container_id: the location or c
        loadedSuccessfully = false;
    }
    //Create the container item
    else
    {
        Item* i = new Item(splitLine[2], splitLine[3], splitLine[4]);
        containers[splitLine[1]]->AddItem(i);
    }
}
```

Figure 3: And this one does the same for non-container items

```
#Test specification for Zorkish Adventures

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

W:Test World

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

#Starting location: the void
L:void:The Void:an empty, formless void.
S:void

#Where the player moves to next: the south room
L:south_room:The South Room:an empty room.

#The west room, which leads back outside into the void
L:west_room:The West Room:an empty room.

#The middle room, from which the player can go N/S/E/W
L:middle_room:The Middle Room: an empty room.

#The east room, which leads up to the attic
L:east_room:The East Room:an empty room.

#The attic full of items
L:east_attic:The Attic:a dim attic.
I:east_attic:book:Book:A dusty old book.
I:east_attic:pencil:Pencil:A short, used pencil.
I:east_attic:glasses:Glasses:A pair of glasses.
I:east_attic:quill:Quill:A black and grey quill.
C:east_attic:red_bag:Bag:A red bag.
I:red_bag:gold_coin:Gold Coin:A gold coin. This is valuable.
I:red_bag:silver_coin:Silver Coin:A silver coin. This is worth a bit.
I:red_bag:copper_coin:Copper Coin:A copper coin. This isn't worth much.
C:red_bag:small_box:Box:A small wooden box.

#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 4: Sample of the Test World's text file

```cpp
std::string SelectAdventure::Setup()
{
    setup = true;
    worldNames = std::vector<std::string>();
    worldFilenames = std::vector<std::string>();

    //Read available worlds from file
    std::ifstream ifs(worldsFilename);

    if (ifs.is_open())
    {
        std::string line;
        std::vector<std::string> splitLine;
        int lineCount = 0;

        while (std::getline(ifs, line))
        {
            lineCount++;

            if (line.length() == 0 || line[0] == '#')
            {
                continue;
            }

            splitLine = StringManager::Instance()->StringToVector(line, ':');

            if (splitLine.size() != 2)
            {
                std::cout << "\tError reading '" << worldsFilename << "' at line " << lineCount << std::endl;
            }
            else
            {
                worldNames.push_back(splitLine[0]);
                worldFilenames.push_back(splitLine[1]);
            }
        }
    }
    else
    {
        std::cout << "\tUnable to open file\n";
    }

    ifs.close();
```

Figure 5: SelectAdventure.Setup(), Part 1: reading in the text file

```cpp
    //Print setup output
    std::string result;

    result += "Zorkish :: Select Adventure";
    result += "\n-------------------------------------------------------";
    result += "\n";
    result += "\nChoose your adventure:";
    result += "\n";

    for (int i = 1; i <= worldNames.size(); i++)
    {
        result += "\n\t" + std::to_string(i) + (std::string)". " + worldNames[i - 1];      //It was complaining I can't append stuff like this without the string cast
    }

    result += "\n";

    if (worldNames.size() == 0)
    {
        result += "\nThere are no adventures to choose from.";
    }
    else
    {
        result += "\nSelect 1-" + std::to_string(worldNames.size());
    }

    result += "\n:> ";

    return result;
}
```

Figure 6: SelectAdventure.Setup(), Part 2: printing the worlds to the terminal.

What we found out

- Visual Studio didn't like "map[index] = MethodName(vector[int], new Class(map[vector[int]], vector[int])", specifically the object pointer "map[vector[int]]" being passed to Class()'s constructor. Assigning the pointer to a variable first didn't help. Just ended up passing the id string instead, and then looking up the object pointer later.
- When re-loading a stage, you need to wipe values that you read into a vector when you first loaded it, so that you don't add duplicate values when you reload it.
- C++ doesn't support nested functions.