Spike: Task 13.P
Title: Composite and Component Patterns

Author: Mitchell Wright, 100595153

Goals / deliverables:

To demonstrate an understanding of components, composition and interaction between different game objects.

Items created during task:

- Code, see: \13 - Spike – Composite and Component Patterns\zorkish

Technologies, Tools, and Resources used:

- Visual Studio 2022
- SourceTree
- GitHub
- Lecture 5.1 – Composite Patterns & Components

Tasks undertaken:

- Composition
- Component Pattern
- Demo
- Commit to Git

What we found out:

1. Composition:

Implementing the composition pattern invovled refactoring a lot of the major classes from previous tasks and creating a super-class for game objects. This is shown below.

```
#pragma once
#include <string>
#include <map>
#include "Component.h"

using namespace std;

class GameObject
{
protected:
    string _name;
    string _desc;

    map<string, Component*> _components;

public:
    virtual string GetName();
    virtual string GetDesc();

    virtual Component* GetComponent(const string&);
    virtual bool AddComponent(Component*);
    virtual bool RemoveComponent(const string&);

    virtual void Update() = 0;
    virtual void Render() = 0;

    virtual void Execute(const string&) = 0;
};
```

A lot of reused functionality has been shifted from Locations, the player, Inventories, etc, and dropped into this class, in order to allow for the implementation of a rough composite pattern. Please note the absense of inventories from this, as attempting to get those working was difficult due to circular dependencies with items, the inventory and game objects.

The implementation of these functions is much the same as in prior tasks, with most returning a bool as a safe guard.

The Take and Put commands were implemented to take advantage of this structure.

```cpp
void TakeCommand::Take(vector<string> command, Player* player)
{
    Inventory* playerInv = player->GetInventory();
    Inventory* locInv = player->GetLocation()->GetInventory();

    Item* item = locInv->Get(command.at(1));

    if (item != nullptr)
    {
        playerInv->Add(item);
        locInv->Remove(command.at(1));
        cout << "The " << item->GetName() << " has been added to your inventory." << endl;
    }
    else
        cout << "Item could not be found." << endl;
}

void TakeCommand::TakeFrom(vector<string> command, Player* player)
{
    Inventory* playerInv = player->GetInventory();

    Item* item = playerInv->Get(command.at(3));

    if (item != nullptr)
    {
        Inventory* itemInv = item->GetInventory();

        Item* item = itemInv->Get(command.at(1));

        if (item != nullptr)
        {
            playerInv->Add(item);
            itemInv->Remove(command.at(1));
            cout << "The " << item->GetName() << " has been taken from " << command.at(3) << "." << endl;
        }
        else
            cout << "Item could not be found." << endl;
    }
    else
    {
        cout << "Item not found." << endl;
        return;
    }
}
```

Being able to take an object, get it's inventory and put an item in, meant that the composite pattern was somewhat working well. Again though, due to the lack of inventory implementation in the pattern, it wasn't possible to just use game objects rather than specific classes.

The Put command functions much the same, but in reverse.

2. Component Pattern:

Components were simple to create, but time-consuming to retrofit into the code. I designed the open and lock commands in order to work with a lock component, so that the pattern could be demonstrated.

```
using namespace std;

class Component
{
protected:
    string _name;

public:
    virtual string GetName() const;

    virtual void Update() = 0;
    virtual void Render() = 0;

    virtual void Execute(const string&) = 0;
};
```

Each component has an Update, Render and Execute. Though, in this case, I've just opted to move the lock's functionality into it's own function.

```
class Lock : public Component
{
private:
    bool _locked;

public:
    Lock();
    ~Lock();

    bool GetLocked();
    void ToggleLock();

    void Update();
    void Render();

    void Execute(const string&);
};
```

The open command, as shown below, checks the item for the component, then toggles the component's lock function. A dynamic cast is used to allow the program to treat the generic component as a lock.

```
void OpenCommand::Execute(vector<string> command, Player* player)
{
    if (command.size() == 2)
    {
        Item* item = player->GetInventory()->Get(command.at(1));

        if (item != nullptr)
        {
            Lock* lock = dynamic_cast<Lock*>(item->GetComponent("lock"));
            if (lock != nullptr)
            {
                lock->ToggleLock();
                lock->Render();
            }
            else
                cout << "There is no lock to open." << endl;
        }
        else
            cout << "Item could not be found." << endl;
    }
}
```

The look in command uses a lot of the same structure, though does combine that with the Inventory command.

```cpp
void LookCommand::LookIn(string itemName, Player* player)
{
    Item* item = player->GetInventory()->Get(itemName);

    if (item != nullptr)
    {
        Lock* lock = dynamic_cast<Lock*>(item->GetComponent("lock"));
        if (lock == nullptr || !lock->GetLocked())
        {
            item->GetInventory()->Render();
        }
        else
            lock->Render();
    }
    else
        cout << "Item could not be found." << endl;
}
```

3. Demo:

```
Welcome to Zorkish Adventure
You're in Anor Londo. What will you do?
look in bag
It is locked.
You're in Anor Londo. What will you do?
open bag
It is unlocked.
You're in Anor Londo. What will you do?
look in bag
It's empty.
You're in Anor Londo. What will you do?
take sword
The sword has been added to your inventory.
You're in Anor Londo. What will you do?
put sword in bag
The sword has been added to the bag.
You're in Anor Londo. What will you do?
look in bag
It's a sword. pointy boi.
You're in Anor Londo. What will you do?
open bag
It is locked.
You're in Anor Londo. What will you do?
look in bag
It is locked.
```

4. Commit to Git: