Maintenance Report

Introduction
This document describes the improvements that were made to our code during the maintenance phase. It starts with a description of closed bug fixes and feature requests. Then moves on to improvements that were made to our source code in general. Finally, there is a discussion of design issues that we identified and the fixes or reasons we chose not to address them at this time.

Bug Fixes
We were able to close all 3 bugs that were submitted by the testing team. The first 2 of these bugs related to the characters name and saving and loading.

The first bug was that Linux file names cannot be larger than 255 character (or so). We fixed this issue with names that are too long by limiting names to 100 characters or less. This still allows the player a great deal of freedom when naming their character, but prevents the file name length issue.

The second related to using characters in a players name with symbols like / . or “ etc. We solved this by hashing the name into an integer and using this integer as a file name. This means that a player can put whatever legal ASCII character into their name and the file will properly save and load. The name is hashed for saving and also before searching for a file to load. This also will work to fix the first issue too as our hash method would require an extremely large name to be 255 digits long. However, to prevent issues with name length we felt it was still a good idea to have a char limit on names. We do like the idea of a player being able to put spaces and symbols in their name, so this is why we hashed it. It allows more flexibility and freedom of expression for the player, and hopefully a more enjoyable user experience.

Feature Requests
We added one of our own feature requests. This was that with windows looking at the window describes the widow and use tells you what is outside of it. We wanted to change this behavior, but we decided that it was not worth the gains. Instead we just made the text for the description of the window that we actually have in the game take the same text as the use text. This means that for now it is more intuitive. However, in the long run it would not be ideal. A new class for a window would probably be the best fix. Since our game only has one window we feel it is not a good use of our time. If the project were to progress in the future it would be something to add if windows were going to be added in more places.

Source Improvements
We could easily have spent a large amount of time on source improvements. There are very few to no comments in our source code. And there are a few methods that are a bit longer than is desirable.

We chose with the time we had to address an issue in our loading class and code. We were a bit short of time and our game had a lot of complex elements so loading them required a good deal of thinking. This meant that we kind of hacked together a solution. We introduced a number of new functions that we only declared and implemented in the load.cpp file. These functions were not part of the load class. They also by the time we were done implementing them all took a game* as their first argument. This allowed us to pull these functions out of the load implementation file and move them into the game class. They all had to do with updating items that were stored in the game class anyway so they belong there. There may still be some design issues with the game class as it now is quite a bit larger, but the operations are at least related to the game. Also, this change makes the load class more manageable.

Design Changes
When it came to design changes we were unable to tackle any of the things we felt were necessary in the time we had. All of them were quite involved and so we chose to leave them for now. In the future they would need to be fixed.

The most problematic of these is our game building code. Because we were out of time we did not get to implement a proper factory or builder class for our game objects. We did get a factory for the game (hence the gameBuilder), but the object creation inside it is just done with constructors and setters. Now our factory method for building a game as all the code to instantiate every game object and connect related objects together. This has become unmanageable. If we want to pass the style check and keep our pipeline passing we cannot actually add any more objects to the game. So it would be impossible with the current implementation to grow the game. However, this is a huge task as we would need to make factories or builders for each of the class types we have and some initialization code to assemble different parts of the game and add them to the game. So in the future it would be the first thing to work with.

Along the lines of building we also have more data saved to reload the game than we currently need. We originally designed the system to save and load to keep reference to all game objects and events with textual data. The idea was we could build the game from a JSON like text file. When this was too much to fit in we did not have time to change the setup. This means that currently our save files are saving more data than they need. However fixing this would require going into each one of our classes and changing the make blueprint method. Then tests would need to be rewritten to work with the new desired behavior.

Another design flaw that we were unable to get to is that in our game all objects are stored in maps by their name. We felt that this was OK because we had no intention of having objects with the same name. However, once we made the game we realized that some rooms would have multiple doors to them and those doors had to have different names. For example: common room door, door to common room, hall to common room. This is cumbersome to work with and really a pain to add new classes if you have to make sure that they don't have the same name as another game object. What we would like to do is to implement a system that adds an id field to game objects. This way they could be stored in maps by their id instead of name. Then objects could have the same name (though this would introduce some issues if 2 objects with the same name were in the same room). The problem with this is that our save and load code is dependent on using the name to store things. Also all searching for objects uses name, etc. So while adding an id field would not change any other class, switching other classes to use it would require changing many other classes. And because of the way this worked they would all have to be changed before any of it would work again.

If we had time we would probably also change our parser. We would like to have a little more dynamic command structure to allow players to type more English like commands. Ie: “give wrench to sam”, “get the shiny coin”. Or we would also like to be able to use only part of a noun instead. Ie: “use hall door”, instead of “use door to hall”. But this also is a large undertaking and not really in the scope of the project. However, we are pleased to know that because our parser has an interface and our command objects also have a good interface we could easily code up a new parser and new commands and plug them into our current structure.