**Unit Testing**

The haunted building game was developed using Visual C# with a graphical user interface and SQL Server for its database. No off the shelf game engine was used. The main purpose of our game unit testing was to test most of the main functionalities that were provided in the final report. Specifically, testing whether certain features where available or if they behaved as specified and expected.

The code was tested by running test suites on several functions in an attempt to determine how stable the program was. Our main goal was to prevent the game from crashing as much as we can mostly because of the invalid input entered by user, such as input with zero length, or out of bound or even worst, entering hacking codes, etc.

First we decided to conduct the test units by determining those type of inputs which may cause the program to crash or raise exceptions in an inappropriate way. Then, we clarified our expected result with having such an invalid input. So, we executed the program with such a input to see the actual result. If the result was not the one we expected, then we tried to discover what was causing the exception by single stepping in the debugger.

The code works very well while using correctly with valid input. However, there were several situations that could lead the program to failure state. So we documented those situations with exact result in our report which is one of the most important part of the report mostly because of their potential to destroy the program structure or cause the program to crash. It is notable that two methods of White Box and Black Box testing have been applied to our test suites.

Moreover, running acceptance test which its main concern is playing the game from user's point of view, is carried out with a checklist of the functional and interface requirements with the aid of Black Box testing method. The acceptance test is reported based on whether the desired requirements are satisfied or not. The result of the acceptance test shows that most of the requirements were not achieved. However, the main requirements that made the game playable were accomplished along with providing the user a nice and friendly User Interface.

**Inspection Testing**

Overall, each section of code that we inspected held up very well. Every function had correct and consistent naming conventions and was formatted appropriately. IF-ELSE statements and WHILE loops were always nested properly and correctly exited when a certain criteria was met. All of the functions work properly and repeatedly over multiple inputs and different game settings.

Using the *Generic Code Review Checklist* as our guide through code inspection, we were able to thoroughly and diligently examine a magnitude of possible issues with the code. The checklist contained some statements and questions that did not pertain to all the pieces of code we inspected. In the case of an inapplicable statement or question, we simply gave it a check as it technically ‘passed’ those specific criteria.

One reoccurring problem that was prevalent in all the code segments we inspected was that it was inadequately documented or commented. In some cases, there were little lines of comments listed before each function or method that briefly explained what the function did. This was only helpful to the creator of the method. The comments were often too vague and it would provide little insight to another party viewing the code for the first time. Documentation was also placed in areas that broke up the code too much. For example, in the monster code review the bulk of the documentation is in between IF-ELSE statements. This creates a large, unnecessary break between the condition statements and would be better suited if it were placed before the function itself.