Software Engineering I

Program Development Project Technical Report

Name: George Panitsas

ID: N0687345

Contents

[Introduction 2](#_Toc503876165)

[Specification 3](#_Toc503876166)

[Design 4](#_Toc503876167)

[Testing 5](#_Toc503876168)

[Implementation 6](#_Toc503876169)

[Analysis/Reflection/Conclusions 7](#_Toc503876170)

[References 8](#_Toc503876171)

# Introduction

The game is heavily influenced by the Zork interactive fiction game series. In the Zork games, the player is not limited to verb-noun commands, such as "take lamp", "open mailbox", and so forth. Instead, the parser supports more sophisticated sentences such as "put the lamp and sword in the case", "look under the rug", and "drop all except lantern".[[8]](https://en.wikipedia.org/wiki/Zork#cite_note-byte1980-8) The game understands many common verbs, including "take", "drop", "examine", "attack", "climb", "open", "close", "count", and many more. The games also support commands to the game directly (rather than taking actions within the fictional setting of the game) such as "save" and "restore", "script" and "unscript" (which begin and end a text transcript of the game text), "restart", and "quit".1

The program in question takes several elements of the original Zork games in a simplified form. The user assumes the role of a typical gym goer. At the start of the game the player starts at the main area of the gym and has the freedom to use the available facilities scoring points performing actions (i.e workouts, taking classes, changing clothes etc). Upon reaching a certain amount of points, a congratulatory message will inform the player that he has completed his daily workout. Secondary objectives are discussed in the Conclusion section of this report.

# Specification

The game uses the command prompt, hence there is no consumption of resources to mandate a high-end machine to run. An operating system with a C# compiler is sufficient for the program to run. The game was built and tested on Windows 10 using Microsoft Visual Studio.

The player can navigate through the predetermined areas of the “gym” and can performed specific actions in each room. A basic structure of the maps is shown in figure 1. The main room consists of a series of available equipment for the player to use along with the options to enter a different room. The locker rooms’ available actions include changing attire, safe-locking personal items etc. while the group classes allows the player to join one of the available classes. The maximum amount of classes the player can partake is two every time the program is run and the player is notified when the maximum amount of classes taken is reached. The helpdesk functions as a game manual that has a list of available commands, where each one provides a brief description that facilitates the use of the program.

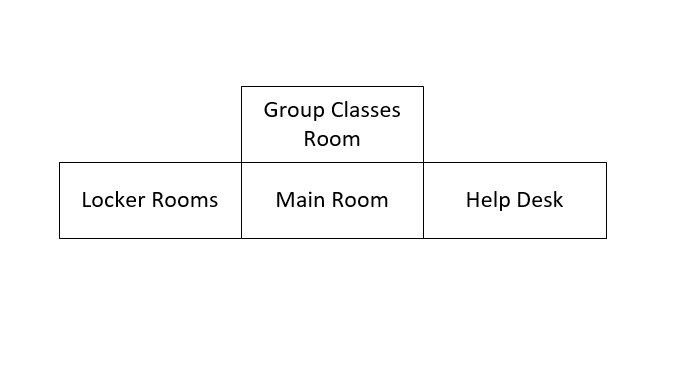


Figure 1

# Design

The initial version of the game will feature the use of fundamental elements of C#, with the most prevalent being functions, enumerators and data structures (specifically dictionaries). The use of an open-ended loop will be used to facilitate a controllable exit of the program while conditionally keeping the program running “perpetually”. Enumerators will be used to control navigation of the player (north, south, east, west) through predetermined states. In addition, the program will allow the user to move to any room starting from the central room, while in the rest of the rooms he will be given the option to return to the main room only. The use of multiple Dictionaries will facilitate the use of different keywords the player is allowed to use. In detail, the use of multiple dictionaries will enable the distinction between physical language elements (such as verbs, nouns etc.). A scoring system will be implemented and will reward the player a certain amount of points and when the threshold is reached, the player will be shown a congratulatory message. The player will still be allowed to use the “gym’s” facilities but point accumulation will stop.

Code optimization will start once the key features mentioned in the Specifications work to a satisfactory level. The primary step for optimizing the program is to read the majority of physical language texts, such as starting messages and keywords from .txt files.

# Testing

Primary tests show that the program conforms to the minimum requirements in the sense of the player being able to navigate in the rooms in the manner mentioned in the previous section and a faulty input (unidentifiable words or empty space) is treated accordingly. Figures 3-5 show that the logic of the program is functional to a satisfactory level. The player can go to any room from the starting point and remain in the same room in case of unrecognizable or wrong input. Room restriction was also achieved, with the player being able to return only to the main room.

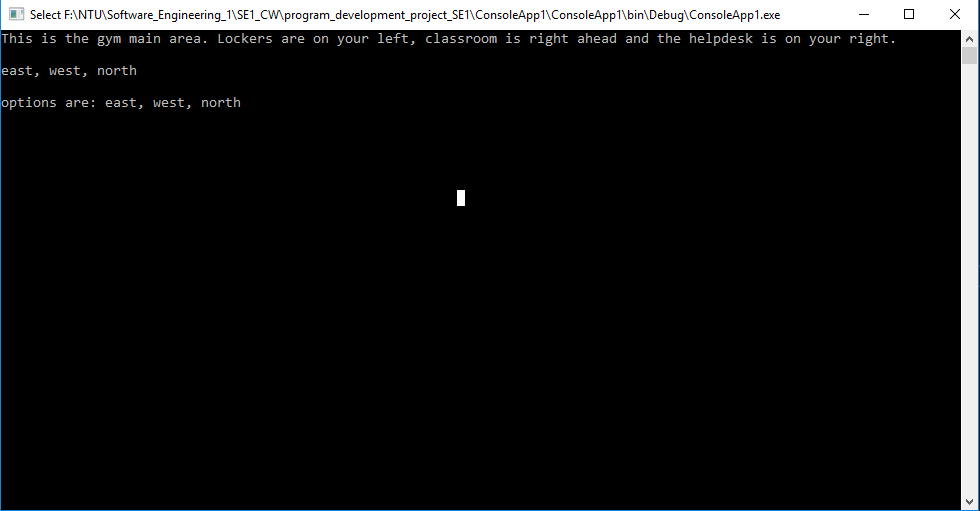


Figure 2

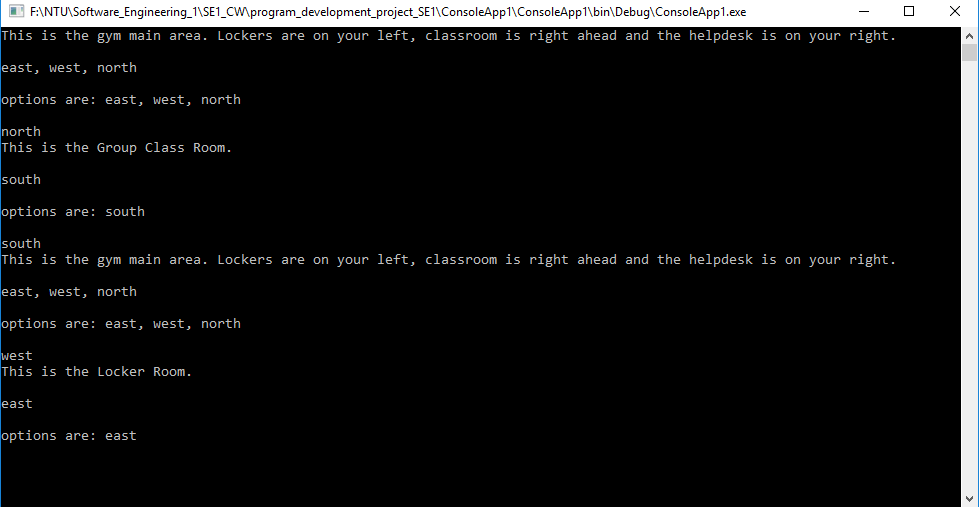


Figure 3

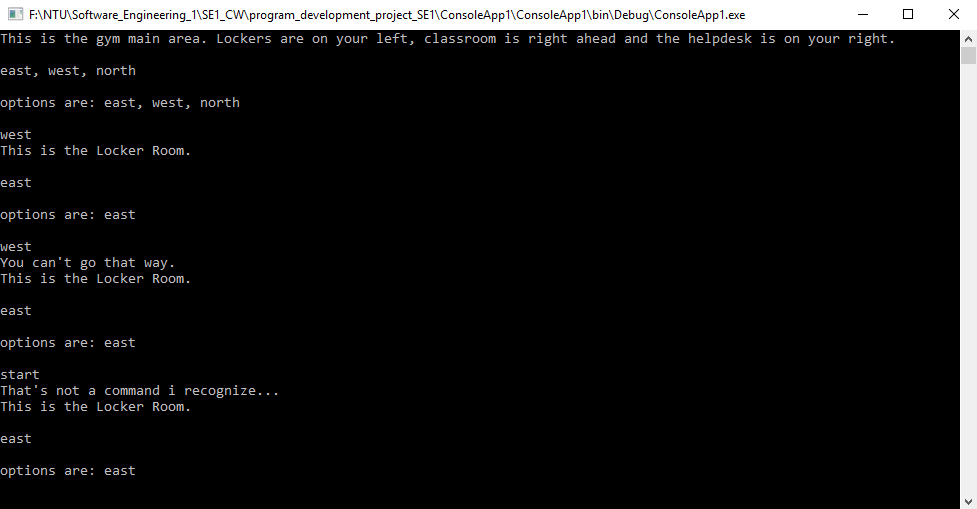


Figure 4

# Implementation

In early tests, wrong input, despite being dealt with, would cause the program to return the player to the previous room. This was fixed with the if-else clause shown in figure 5. A major implementation problem that lead to code revision and refactoring was parsing of multiple words from different dictionaries. The current implementation (parts of which still exist in the code submitted) would produce errors that caused the program to crash, or not being recognized at all.

With multiple words not being read from the program, implementation of scoring system along with certain major feature were stalled and were left in pseudocode until the parsing problem was solved. This was caused due to the intrinsic relationship of certain phrases that would reward scoring points to the player.

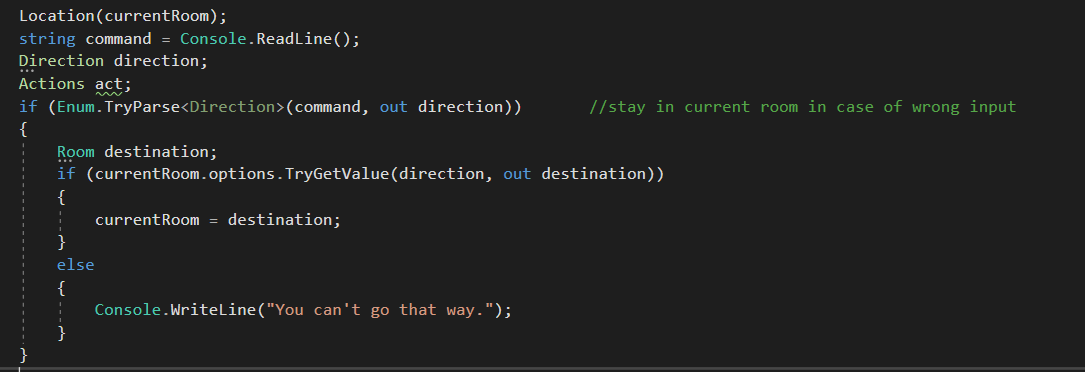


Figure 5

# Conclusion

The program satisfies the minimum requirements in the sense that the player can navigate in the rooms with appropriate handling of faulty input. During all phases of development, various features have been attempted for implementation, as separate modules, or in the form of pseudocode. However, from early tests it was apparent that a different approach should have been used for the parsing of multiple words. A possible solution to this is reading words from separate .txt files, where each text file includes lists of the appropriate keywords. This approach would additionally facilitate the modification of the games “dictionaries” to be changed without having to access the code. This approach would also be beneficial for messages, or texts that would be shown to the player.

Improvements to the program would include the use of a secondary “hidden” scoring system (etiquette points) for when the player performed actions in a specific sequence, and unlocking access to an additional room and/or additional actions to the existing rooms

Saving game state on a text file.

Date and time based features, such as different classes that would be available during different times of the day. In addition, point accumulation will reset after midnight, while keeping score for each date that would be available to display on the screen when the player inputs a specific command to the game itself.

# References

1. <https://en.wikipedia.org/wiki/Zork>
2. <https://social.msdn.microsoft.com/Forums/vstudio/en-US/fe4d77f3-c3d2-4851-b5f4-b500e7f1716c/c-zork?forum=csharpgeneral>