Samantha Berry

CMPT 220 Section 201

Dr. Rivas

8 May, 2017

Text Adventure

Abstract

This paper discusses the development of my final project, a text adventure game. First the program and its origins are introduced in the introduction. Then the details of how the program works are laid out in the system description, which includes a UML diagram. This is followed by system requirements, and a brief user manual. At the end the project and the developmental process are reflected upon in the conclusion.

Introduction

For my final Software Design 1 project I chose to create a text adventure game. I thought that making a text adventure game would be a good way to show my skills at programming, while also covering a topic that I felt was interesting. My interests in computers started when I was a child due to my interests in video games, so I felt that creating a game for my final project would be fitting.

In this paper, I will cover in depth the process of how my text adventure works, using UML to better describe it. Then I will briefly mention requirements before proving a user manual for my program.

System Description

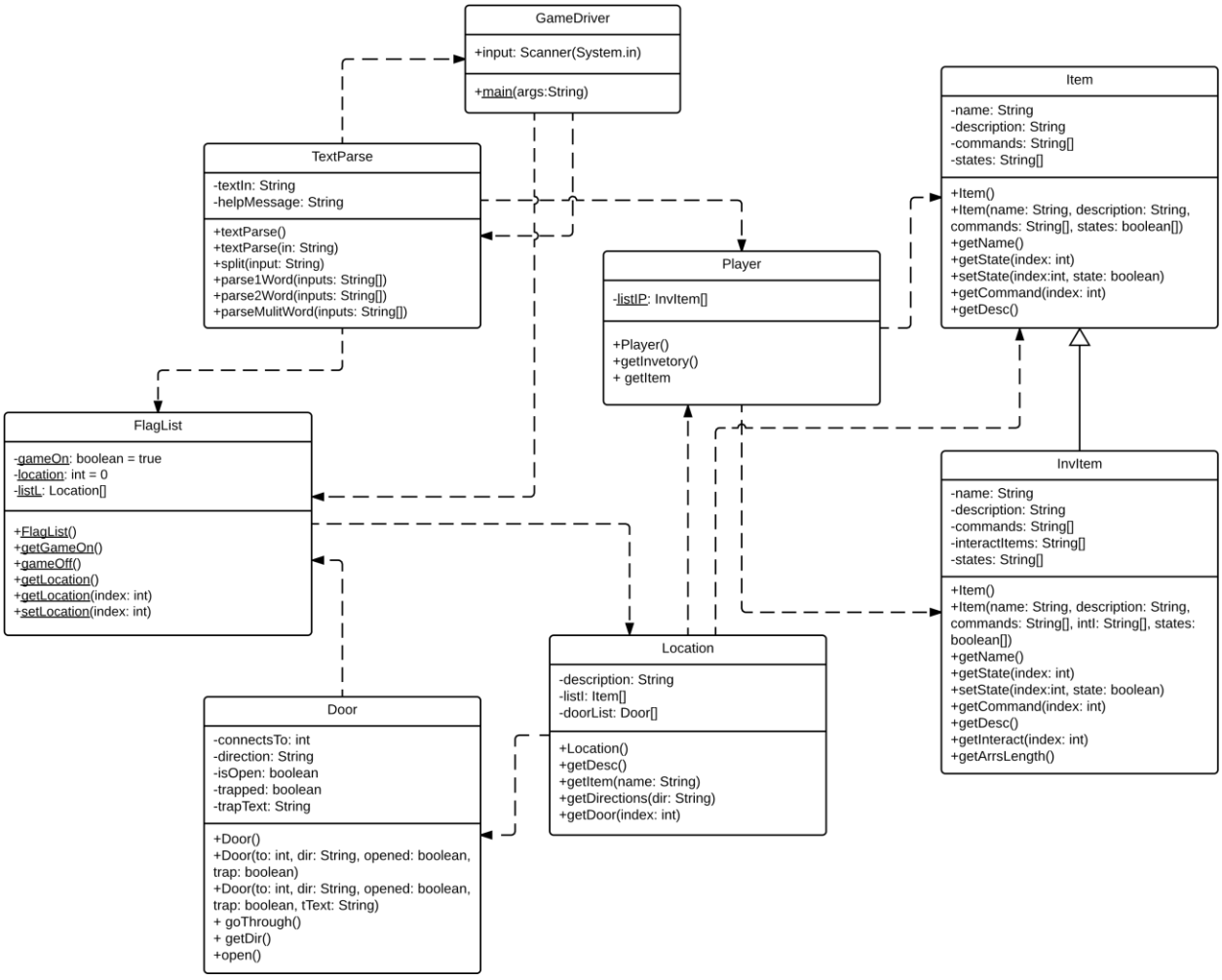
The System used for my program is composed of eight Java files. The GameDriver file contains the main method that is used to run the program. The main method contains a while loop which will not stop until some end state is reached. Within this loops the program reads input taken from the user, which is then sent to the TextParser class.

This text is first formatted and then, using the split() method, is separated into an array. Depending on how long the resulting array is it is sent to one of three text parsing methods. If the array is only one word long it is sent to parse1Word(). This class checks the word against a list of predefined words, and then a response is returned based on the word used. If the array contains two words parse2Word is used. Parse2Word can read inputs in the format ‘verb’ ‘item’. Parse2Word first checks if that item exists at the location and if it is still there, which is implemented using two if statements. Then the method checks if the commands useable for the item match the given verb, and if that command can be performed. If all the above are true, the state of the item is changed and the action can no longer be performed. If the item uses correlates with an inventory item then it is added to the player’s inventory. This sequence is complicated, but I have built it so that in all but one instance it performs the action as I want it to. If the text has more than three words it is sent to parseMultiWord(), and it is used in cases where players use items in their inventory. This method works in a similar manner to parse2Word, however, the if statements used to manipulate the inputs are more specific, due to the multiple ways that inventory items can be used.

The code used in the if statements of the parsing methods looks complex, and this is mainly due to how the objects in my program are organized. A Location object contains both

Door and Item objects, and a Player contains Inventory Item objects. Structuring the objects this way makes knowing where certain objects are in the game easier and more intuitive.

UML Diagram



Requirements

This program is not resource intensive and should run on any computer with Java capabilities without problems.

User Manual

In order to run the program, you must run the GameDriver.java file. When playing the game use the command line to enter text and press the enter key to complete your statement. When writing your commands make sure to eliminate unnecessary words such as ‘the’, ‘to’, and ‘in’. Two word commands should be written in the format ‘verb’ ‘object’. Three word commands should be written in the format ‘verb’ ‘inventory object’ ‘environment object’. Three word commands can only be performed using inventory items. To move from room to room use commands in the format ‘go’ ‘direction’ or ‘move’ ‘direction’. Enter ‘inventory’ to see your inventory items. Enter ‘quit’ to exit the game. Remember to look at items and the environment for clues on how to progress.

Conclusion

In the process of making this game I used all of the skills I have learned in programming, and there is no doubt that this is the largest and most complex program I have ever made. I learned many important lessons about time management, and how much time and effort has to go into making a program of this size. There were many times in class where I could quickly finish assignments without much thought, but I couldn’t do that with this project, and there where many times that I had to carefully plan out what I would code. Making this program was a gradual process, and there were many times when I went back and changed code I previously wrote in order to make it more efficient. Although I experienced many problems when developing this project, I still consider it a positive experience.