CSC241 Project Description

Daniel R. Schlegel
Department of Computer Science
SUNY Oswego

January 17, 2024

This project involves constructing a virtual world simulation which the user will interact with via a string-based command line interface.[1]

Premise

In the simulation you will create, the player acts as a person who is trying to escape from a castle. The player begins in a dungeon cell and must find their way out of the castle. Along the way, the player will run into locked doors. So, they must use context clues from their explorations and inspect objects looking for keys along the way to open locked doors. The player may run in to guards and maids along their journey. If the guards are awake or the maids are cleaning, they have the possibility of getting annoted by your snooping around the room. If this happens, you will be sent back to the dungeon cell and your health will reduce. Lose all of your health and it's game over! But, make it out of the exit with some health remaining and you win!

Description of the World

The virtual world consists of a series of rooms which are joined together with doors forming a castle. A room can have at most 4 doors, one to each of the north, east, south, and west. Staircases are also considered to be rooms. The player may move from their current room to an adjacent room using a command for the cardinal direction (*i.e.*, `north`, `south`, `east`, and `west`). A room may contain items and characters. Characters other than the player always stay in the same room, and rooms and doors always stay in the same place. Some items, on the other hand, may be picked up and later dropped by the player. Of course, the player can move between rooms. Therefore, a character leaving a room always means that the character is entering another room – there are no hallways, and a character can only be in one room at a time. There are three types of characters: the player character (PC), and two types of non-player character (NPCs) – guards and maids. There is only one player character. There can be multiple NPCs.

Rooms, doors, items, and characters all have unique names. Each also have text descriptions (which may or may not be unique). Each character has access to a `look` command, which lets it see the name and description of the room it is in, as well as the names of the characters and items in the room. Characters also have access to the `inspect` command which shows the description of an item, door, or other character to learn more about it. Inspecting can also expose hidden items (*e.g.*, items in a drawer).

A door which is locked may have a key hidden somewhere in the castle. When the player encounters a key, they should add it to their inventory using the `pickup` command. If the player picks up things they don't need, they can drop them using the `drop` command. Items have weight and there is a maximum amount of weight the player an carry. Not all items are able to be picked up (for example, a chair someone is sitting on!) The inventory listing is available at any time with the `inventory` command. If the appropriate key is in the player's inventory, they can unlock a door using the `unlock` command, or lock the door with `lock`.

Each guard may be sleeping or not, and each maid may be cleaning or not. If the guard is awake, or the maid is cleaning, and the player inspects an item in the room, there is a 20% chance that the character notices the player is out of place and has them sent back to their cell.

The player starts with a full health bar: `[*****]`. Each time the player is sent back to their cell they lose one asterisk from their health bar. When it is empty, the player loses the game. If the player makes out to the exit with some health remaining, they win.

[1]This project is loosely based on previous CSC241 projects written by Alex Pantaleev and Lynna Cekova.

Input, Output, and Game Interface

The primary game interface is a command line, where the user can input String commands, and where text messages are shown to the user. Graphical components possibly will be added for portions of the interface.

Upon program startup, the user should be asked to provide an input file with information about the world (rooms, characters, and items). Your program should parse this file, and, based on the information in this file, create the rooms and the characters that are in them (including the player). A sample input file will be provided to you, together with the description of the file format. You can use this file in writing and testing your project (or you can make your own input files if you so wish), but keep in mind that other input files will be used upon evaluation of the projects. (This means you do need to parse the file based on the file format; hard-coding input values is not acceptable.)

Next, the user should be presented with a prompt to issue their String commands. When read by your program, these String commands should be parsed and used to make the player character perform actions (or perhaps order other characters to perform actions in an extension of the game we may explore). Output will be returned to the user (telling what is happening as a result of the actions), followed by another prompt for a user command. The user will be presented with prompts for commands until the user types `exit`, or until the game ends. If the game ends, the user receives a message. (Sample messages are "Congratulations, you escaped the castle and won the game!" or "You loser! You lost!").

A Note on the Wording of Messages and Commands

The wording of the messages themselves is of no importance, as long as all the information of what is happening is presented. In fact, you are encouraged to make adjustments and personalizations as this often makes the development process more fun. The wording of the commands is important because they have to be processed computationally, but again feel free to customize as long as you are consistent and the functionality remains. It's worth taking the time to decide in advance what your set of commands will be. Define the format of your own String commands and show them to the user, together with a description of what they do, any time the user types `help` at the prompt.