Text Adventure

Professor Caleb Fowler
June 5, 2023

Problem.

Create a simple text based adventure game. You explore the environment by issuing key commands. The computer produces textual descriptions of what you find. A discussion with example output can be found here: www.cabinetmagazine.org/issues/64/lucas.php

The game will be comprised of Room() constructs. These constructs will have a room_name, room_description, room_id, occupied code, and 4 exit codes (’N’, ’S’, ’E’, ’W’). You are free to add other variables as necessary. The room name always appears. The room description only appears the first time you enter that room or when you enter a ‘L’ook command. The room id doesn’t do anything unless you are using an array - in that case it is the element number of that room in the array. The occupied code just indicates if there is anything in that room. The exit codes are the commands you use to move to the next room. Allowable exits should always display along with the room title.

If the room is occupied, randomly select one of 4 monsters to spawn in the room. What, exactly, they do with your character is up to you. Not every room needs a monster! In fact, you are likely to not see some of the monsters on any given run. You **DO NOT** need to create a visual map.

Requirements.

- Move throughout the adventure with the N,S, E, W keys (moving you in the four cardinal directions. This moves you from room to room (if there is a room in that direction). You are NOT creating a map to move around with.
- Create a minimum of 5 rooms in your dungeon. The exit does not count as a room.
- The game ends when the player leaves through the exit - tell them the game is over.
- When a room is occupied, spawn a monster.
- Use functions whenever possible to modularize your code. Use function prototypes and code the functions under main().
- Remember, at a minimum you will have 5 commands to implement: N, S, E, W, L. Re-prompt if an incorrect command was entered. Keep re-prompting until the client get’s it right.
- Include a void ProgramGreeting() function. This will run automatically once when the program first starts. This function should display (on individual lines):

  – A welcome message.
  – Your name (as author).

– System date. Format this as MonthName day, year. Example:
June 30, 1988.

- Store the results of all computations.
- Include your specification comments above the main portion of your code where you implement the specification. No credit if this is missing!
- Use white-space and comments to make your code more readable.
- Put a Source File Header at the top of your source file.
- No global variables or global variable look a-likes.
- Use function prototypes for all functions. Functions go below main().
- Do not use c (.h) style libraries. Use C++ libraries instead.
- Your program must compile in C++ on Ubuntu.
- Your program must generate logically correct output.
- Program activities are split into logical 'chunks' or paragraphs. I'm expecting paragraphs for input, processing (if any), and output operations.
- If there is non integer output, force the computer to always display 3 places to the right of the decimal.

Specifications.

// Specification C1 - Main Data Structure
Show me the main data structure you are using to store the room information. You can hard code this, create an array, create a parallel array or use dynamic memory. You may NOT use the standard template library (ie vectors, etc. etc.).

// Specification C3 - Input Validation
Perform input validation with the command options. Only allow the entry of valid commands.

// Specification C3 - Detailed Look
Show the room_title and room_description every time the player enters the 'L'ook key.

// Specification B
You are to select 3 features to add to your design to meet this specification bundle. Comment each feature you add the same way you've been doing it all class (// Specification B1 ....). I still want to see 3 specification comments. Don't forget a 1 sentence description of the feature you just added in your specification comment. You are free to create other specifications if you wish - just make sure they are equivalent in complexity to these (no "Ready player 1" cout's).

- Add more command options.
- Add combat.
- Add trap encounters. Once revealed (visible) these are permanent. They will appear whenever you subsequently reenter that room.
- Add treasure. This practically begs you to keep some sort of inventory as well. I suppose you'll also need an <i>nventory command to see it.

- Add the concept of light and dark rooms - the status of which will have some sort of game effect.
- Add more monsters (I'll give you credit for only 1 more monster, regardless of the number you actually add).
- Add more rooms (only 1 more counts here, more than 1 new room doesn't count).
- Use a random room generator to create an unlimited number of rooms.
- Use an array of structs or classes.
- Put your monsters in a Monster() class.
- Create a pseudodynamic array. You don't need to grow or shrink this - easy!
- Add directions of 'u'p and 'd'own as well.

// Specification A - Reflection
This reflection exercise is different from the prior ones. You will not use ChatGPT and you will not be reflecting on this code.

Think about how much you knew about C++ at the start of this class. Now, look at all you have done for this assignment. Review and reflect on your experience in this class. Write this up in 250 words or more. Include your write up as a block comment at the bottom of your assignment. **Also indicate the number of words in your write up, as well**. You may wish to:

Compare and contrast how much you knew at the start of the class vs. how much you know now. You can comment on what you liked learning about and what you didn't like. Feel free to also discuss the usefulness of this reflection activity and the role ChatGPT played in it.

Commentary.

This assignment generates more features from my students than any other assignment I give (in any class). The current record holder is 18 additional features.