Final Project

Let's now bring the things that we've been studying together and have a little fun along the way.

For this assignment, let's build a simple game. Here are the things we have:

• Game board

A 5 by 5 grid containing a total of 25 rooms.

• Rooms

Rooms may have up to 4 walls. Each wall may be *Open* which will allow unrestricted passage through *Solid* which will prevent passage *Magical* which will allow passage only with a secret word Some rooms are dark and some are light

• People

We will have 3 kinds of people

You the adventurous explorer, wandering from room to room.

Leprechauns the good wee people with a mischievous sense of humour. They can give you gold or magic words but you never know if they are telling the truth. They can teleport to any room.

Gnomes who guard the treasures of the earth and just might take your stuff if you're not careful. They can go one space in any direction - they know the magic word to walk through any kind of wall.

• Things

Food gives you energy to keep going.Gold pays the way.Fake gold given to you by mischievous leprechauns - also good for giving to gnomes.Teleporter moves you at random to some other room.Light allows you to see in a dark room - tells you what's in a dark room.

• Moves

Up or *down* allows you to move up or down by one space. *Left* or *right* allows you to move left or right by one space. *Teleport* allows you to teleport to some other room if you have a teleporter.

Each move consumes 1 food unit and teleporting consumes 5.

For this exercise, you need to design the various classes, their public interfaces, helper methods, and private data members. Some of these classes may be (abstract) base classes and others derived classes. Use your imagination and creativity when defining your classes, commands, and setting the rules of your game.

For extra credit, the configuration of the game board can be stored in a file and downloaded at the start of the game. The configuration can include who or what is in each room. Simple console input and output is sufficient for playing the game. The emphasis here is defining the classes, their capabilities, and their hierarchy, not graphical user interfaces.

Do not spend time creating any form of graphical interface or bringing in any other such packages.

- Rules and Commands
 You invent these. Use your imagination and creativity when defining commands and setting the rules of your game.
 Remember that none of the players can go outside the boundaries of the game board.
 - Simple console input and output is sufficient for playing the game. The emphasis here is on the design, not a fancy user interfaces.

For this exercise, you need to design the various classes, their public interfaces, helper methods, and private data members. Some of these classes may be (abstract) base classes and others derived classes.

Use your imagination and creativity when defining your classes, commands, and setting the rules of your game.

Your deliverables include the following,

Requirements Specification (15 points)

Requirements definition is the process of identifying and understanding what the needs of all interested parties are then documenting these needs as written definitions and descriptions. The focus is on *what* problem the system has to solve. The emphasis is on the world in which the system will operate not on the system itself.

The purpose of the *Requirements Specification* step is to capture and express a purely external view of the system. We refer to this view as the public interface of the system. We identify *what* needs to be done starting from the user's needs and requirements. Non-functional specifications also have to be added. We use these to explain constraints such as performance and timing constraints, dependability constraints, as well as cost, implementation and manufacturing constraints.

Much of this information can / must be captured through UML use cases – remember, graphics and text (including specifications and exceptions). These are part of your deliverables.

Design (20 points)

The purpose of the design phase is to find an appropriate internal architecture for the system that explains *HOW* the requirements are implemented according to an application-oriented viewpoint. The description based on a functional structure and the behavior of each function must be technology-independent. The designer uses the functional design as an entry point for this step.

Our goal is to define or develop the detailed solution to our design problem. In developing the design, we begin with the functional design. We begin by analyzing the problem. Through such analysis, we transform a vague understanding of the requirements into a precise description. The result of such a process is a detailed textual or graphical description of the system. When finished, we have a complete functional definition of the required tasks with no internal contradictions.

As you begin the detailed design phase, structure the system into classes, groups of related classes class or libraries, or types of jobs which include the notions of *helper*, *supporter*, or *doer*. For each class, specify precisely its operations and relation to other classes. Our goal at this stage is to have the classes crisply defined and the interclass relationships of manageable complexity. We define the abstract data types (ADTs) then define the underlying data structures.

Use a functional diagram, CRC cards, and UML class diagrams to help formulate the static architecture of your design. These are part of your deliverables.

Test Plan (10 points)

Test Plan identifies *what* tests need to be carried out based upon the original requirements specification. It describes in general terms the following information:

- *What* is to be tested?
- The testing order within each type of test.
- Assumptions made.
- Algorithms that may be used.

Test Cases and Results (10 points)

The test cases evolved from the test plan, provide the detailed steps for each test. They prove *how* you tested each of the requirements specified in your test plan. Annotated results of executing your test cases.

Source Code (90 Points)

Listings of your program.

Your program *must* be decomposed into main, test, implementation, and header files.

The solution must be implemented as an object centered design, not a procedural program using classes.

Your design must include the following:

- 1. The set of rules and constraints under which the journey proceeds.
- 2. At least 5 or 6 classes these must be legitimate classes....not classes created for the sole purpose of creating classes.
- 3. Copy constructor
- 4. Assignment operator
- 5. Overloaded insertion operator functions as appropriate.
- 6. Composition
- 7. Inheritance and polymorphism

Please clearly mark in your source code where you are using each of these.