Magic Maze: Quest 0 Specification

Deadlines:

See the World Map for availability and dude dates.

Learning Objectives:

- Review Object-Oriented Programming and the Standard Template Library.
- Understand how Object Factories abstract the construction of an inheritance hierarchy of objects.
- Understand the structure and syntax of XML.
- Understand the theoretical relationship between XML and the Document Object Model (DOM).
- Understand how the DOM may be implemented in C++ (via a 3rd Party library).

Task Summary:

The purpose of this assignment is to give you experience working with tools that facilitate flexible game architecture. In this assignment you will:

- construct an extensible, flexible, SUD-style/text-based game space using an inheritance hierarchy of objects that exhibit polymorphic behavior and rely extensively on templated storage containers;
- construct a library of object assets using the ObjectFactory pattern; and,
- utilize the TinyXML library to load, parse, and configure a game state from XML.

Base Code Functionality:

The game space is a simple, open-world single-user-dungeon (SUD). We will implement the game space using three elements:

1. a Room object pointer,
2. RoomMap object pointer,
3. an integer (indicating the player's health).

The main game loop of the program is provided in source-file main.cpp. The basic flow of this loop is:

- construct the game's RoomMap
- assign the initial Room of the game using the RoomMap's findNext() function for a null pointer
- while game-is-not-done (based on the current room and the player's health)
    - assign the next Room by invoking the current Room's execute method, which takes as argument a pointer to the RoomMap object as well as the player's health
- display end-of-game (based on the current Room object and the player's health)

The game is always initialized to the location of the room identified as "start". The game terminates in one of four conditions:

1. the player reaches a WIN room,
2. the player reaches a LOSE room,
3. the player reaches a QUIT room, or
4. the player's health value becomes negative. The health value is decremented once at each legal transition between rooms.

Moreover, there are two kinds of rooms. "Normal" rooms perform normal room transitions according to a transition function, f, which given a current state, s, and transition command, a, produces a new state, s': s' = f(s,a). "Magic" rooms have a similar transition.

Program Elements:

To implement this functionality, we will need to define both the RoomMap class as well as the Room class (and its descendants).

class RoomMap

The RoomMap object stores the topological structure of the game.

private members

`std::map<std::string,RoomFactory*> roomLibrary;` This member maintains a map to the different classes of factories that can be used to construct rooms in the game space. In the base version of the game only two factories are non-abstract, the "Normal" factor and the "Magic" factory, mapping onto the NormalRoomFactory class and MagicRoomFactory class, respectively. This map may be hardcoded in the Initialize method.

`std::map<std::string,Room*> rooms;` This member maintains a map to each unique Room object in the game, which are stored with respect to their identifier member (see Room class below).

public methods

`bool Initialize(std::string configFile);` This method initializes the roomLibrary, sets the value of the configuration file (*.xml) and calls the LoadLevel method.

`Room* findNext(Room*);` Using the map's find method, the method identifies to the pointer value paired with a key that is generated by invoking the next() method of the Room object pointer passed as argument. If the argument pointer is NULL, then the searched key is "start". The found pointer is returned.

`void randomizeRooms();` This method rearranges the rooms map so that each string is associated with a random room pointer. Each room should be in the map only once. The "quit" room should never move.

private methods

`bool RoomMap::LoadLevel(const char* configFile);` This method utilizes the TinyXML implementation of the Document Object Model (DOM) to extract game elements. From the *xml configuration file, this method will identify the Puzzle XML-object, which contains a collection of Room XML-objects (each Room XML-object contains multiple attributes). When a Room XML-object is parsed, a new C++ Room object is constructed and

a pointer to the Room XML-object is passes as argument to its Initialize method. Once a Room object is initialized it is added to the rooms map according to its identifier member.

class Room

The Room class contains the basic game play functionality. Each Room of the game contains an interface to the game's player as well as an encoding of the game rules (i.e., mechanic or a set of transitions to other Rooms of the game).

protected members

`std::string transition;` This string stores the player's requested action, which is usually a single character but could be stored as a more complex system.

`END_CONDITION condition;` This enum type stores the Room's role in end-of-game according to the enum type specified in the definitions.h file.

`std::string identifier;` This string stores the unique name of the Room within the game space.

`std::string description;` This string should describe the Room in a way meaningful in the context of the game.

`std::map<std::string,std::string> neighbors;` This maps a transition keyword onto a particular room identifier. In theory this member implements the transition function (the current Room is the state, s, the key is the transition, a, and the paired value is the next-state, s'.

Note: accessor methods for class Room's members will also be needed.

public methods

`Room();` This default construct provides initial values to the protected members.

`bool Initialize(TiXmlElement*);` This method utilizes the TinyXML implementation of the Document Object Model (DOM) to extract Room elements. From the Room XML-object this method will initialize the object's members using attribute queries. This method will also look for possible Neighbor XML-objects contained by the Room XML-object. Each Neighbor XML-object is composed of a transition and target attribute, which would be assigned to the neighbors map member as a key-value pair.

`virtual Room* execute(RoomMap*, int &) = 0;` This is a purely virtual method used to create polymorphism.

`bool finish();` This method translates the END_CONDITION enum into a boolean value (i.e., WIN, LOSE, QUIT translate to true and **NONE translates to false**).

`std::string next();` This methods finds and returns the next Room identifier associated with the current Room and the current transition. The next Room identifier is found by invoking the neighbors member's find method.

class NormalRoom and class MagicRoom

To make game functionality interesting, two classes are derived from class Room: class NormalRoom and class MagicRoom. These two classes overload the execute method of class Room to create interesting game functionality.

`class NormalRoom` This class's execute method outputs the current Room object's identifier, the health of the player as well as a list of possible transitions (i.e. actions) possible. This method also block, waiting for a std::string input to be input from the keyboard. For example, after the initial splash screen, calling the execute method of the object associated with room1 specified in the example *.xml configuration file would look as follows:

EXAMPLE: ROOM EXECUTE OUTPUT

```
View:   Description of start room.
Health: 10

Available Moves: d q s

Select Move:
```

`class MagicRoom` This class's execute method displays the following message,

```
"You hear loud grinding sounds and you feel your strength restored!" << std::endl;
```

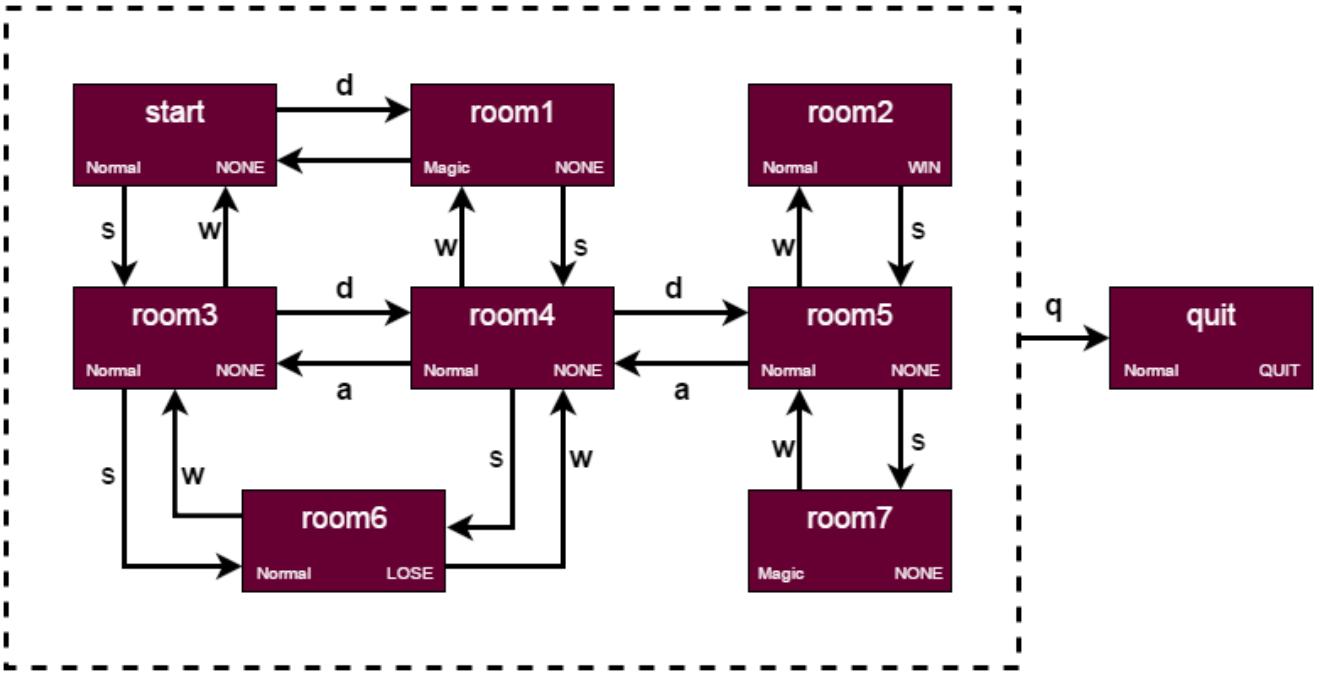
Increases health by 2, and invokes class RoomMap's randomizeRoom method before doing everything that a normal room does. In fact, it is recommended that a generic "execute" method is defined for the parent class "Room" that is called from the child classes.

object factory classes

As described in class, you are required to implement an object factory class hierarchy which overloads the create() method such that the RoomMap object's roomLibrary can be used to construct "Normal" and "Magic" type rooms.

Transition Function of Example puzzle.xml file

To assist you in completing the assignment, the instructor has provided a visual reference for the transition function that is encoded in the puzzle.xml file provided. Note, all rooms transition to the "quit" Room via the 'q' transition. Feel free to change the descriptions for the rooms in the xml file.



Submission

You should submit your Visual Studio project as a compressed archive. First you must name your project to adhere to the format: `userid.hmwk.0` where userid is your UALR user ID (i.e., the part of your UALR E-mail to the left of the @). You then should create a zipped archive, and submit this archive: `userid.hmwk.0.zip`.

Grading Rubric

Your job is to design and implement a C++ procedural paradigm program that performs the tasks according to this specification (and any supplemental specifications provided to clarify program tasks). As would be true in a real game development scenario, you will be evaluated primarily on your code's ability to implement all of the submission and functional requirements correctly. A secondary consideration, but also important, will be the structure, readability, and documentation of your code.

Program Element Values

Category	*XP*
Incorrect submission format*	*-100*
Does Not Compile	*-300*
Lack of Comments (max)**	*-200*
Feature (function) completely missing	*-250*
Some part of feature not working	*-100*
Ineffecient Code (each instance)	*-50*
Undescriptive Identifier (each declaration)	*-50*
Late (each day, up to 4)	*-150*

The minimum number of XP you can receive for a valid attempt is 1040XP. Therefore, be sure to submit an attempt on the due date, and for each day after in which you at least an additional 150XP in completed work.

Please check to make sure you submit your project properly. There is no reason to lose points and make the instructor's life more difficult.

****Comment your code. Comments are worth major points in this assignment.***

Badge Points

You can receive up to 300 Badge points on this assignment.

(200 Badge Points) Save/Load Game: Implement code to save the current state of the game to an xml file and load it back when the game starts. It is up to you to decide how games are saved and loaded (could be a key press, could be automatic, could be timed, could be a menu, etc). Also, each normal room should have an item in it that you pick up upon entering the first time. You decide what the items are and how they are distributed in the rooms, but they should be added to your inventory upon entering a room. The current inventory should be displayed along with the text for each room. When you enter a magic room, all items should be removed from your inventory and randomly placed in an empty normal room (i.e. one in which you already have taken an item from).

The following should be included in your save game:

1. Player health
2. Current room
3. Inventory
4. Items still in normal rooms.

(100 Badge Points) Updgraded Magic Rooms: Instead of simply rearranging the rooms in the map, the nighbors map in each room is cleared and new members are created randomly with the following restrictions:

1. Each "wall" in a room can only have one door! Using a map kind of enforces this, but not checking this may mess up requirement 3 below.

2. Each room must have a way in and a way out.
3. If you enter a room by pressing “s” you should exit that room with a “w” back to the room you pressed “s” to come from. Same for all other directions. In other words, doors are two way. For example if “a” takes you from room3 to room6, then “d” should take you from room6 to room3.
4. Every room’s neighbors map MUST have a quit room accessed with a “q”.

When submitting your assignment, indicate in the comments of the Blackboard submission link that you attempted the extra credit problem, and how you went about solving it.

Appendix

This homework project differs substantially from projects you may have seen in Programming II. The instructor has separated out the code into numerous directories: Config, Game, Source, and Temp. The Config directory contains the *.xml configuration file. The Source directory contains all source code, including a subdirectory for Third Party source code. The Temp directory is the location where temporary object files will be compiled and the Game directory will be the source of the final, compiled, linked, and loaded executable. If you want to recreate this project from scratch you will need to do the following:

1. Create a new, empty Visual Studio 2012 project.
2. Copy source code into the Visual Studio project directory
3. Copy and paste all source code (including third party source) into the Visual Studio solution explorer (this links your new project with the source code in the project directory).

To configure this project, do the following:

1. In the solution explorer, right-click on the Project and select Properties from the dropdown
2. Select Configuration Properties and select General
3. Edit the Output Directory field to be: $(ProjectDir)Game\
4. Edit the Intermediate Directory field to be: $(ProjectDir)Temp\

Your project should now compile and execute (and should be portable to other computers).

Last updated: February 5, 2015