Final Project - Adventure Game Builder

by JL Popyack & Paul Zoski (as found at https://www.cs.drexel.edu/~mcs172/Sp02/Assignments/HW6/index.html)

Due Date: ***Tuesday, December 18 at midnight!***

1. **Background**

   In the early days of computing, before windows and graphic user interfaces, text based adventure games (such as the original *Zork*) were popular. Multi-user Internet variations of these games are still popular today.

   In the text based adventure game, the player moves from room to room reading descriptions, finding objects which could be inspected or possibly carried, and interacting with hostile and benevolent creatures, all in a text based environment. The game would display a description of the room the person was in; and the user would attempt to interact with his environment via simple text commands. A sample transcript of such a game can be found here.

   You assignment is to design a *tool* that will allow the user to create and connect rooms for a text based adventure game. ***You do not have to design an actual game!*** Many of the features in the above example will not be implemented in your program (see extra credit).

   The Room Class

   For this assignment we have provided a Room class that will allow you to build rooms for your adventure game world constructor. You can find the class in the following files: Room.h and Room.cpp. You are not allowed to modify this class while working on your program. You should however study the implementation to familiarize yourself with it.

   Notice that every room has a name and a descriptions. These values can be changed to produce a variety of different rooms.

   Inside the `Room` class is a private vector of pointers to other rooms. These pointers are set to `NULL` when no exit exits. However, when a pointer is set to another Room, that means that exiting in that direction will lead you to that other room. The vector should be indexed using the `Direction` data type:

   - `exits_[NORTH]` - a pointer to the room through the north exit (`NULL` for no exit).
   - `exits_[EAST]` - a pointer to the room through the east exit (`NULL` for no exit).
   - `exits_[SOUTH]` - a pointer to the room through the south exit (`NULL` for no exit).
   - `exits_[WEST]` - a pointer to the room through the west exit (`NULL` for no exit).

When connecting to another room (see the `connect()` method), exit pointers for both the implicit parameter object and the explicit room object are changed so that the two rooms lead back to each other. Note that the following are all allowed:

- Rooms that connect to themselves - you exit to the north and appear in the same room.
- Exits that don't match - you exit one room to the north, but to go back you must go east instead of south.

By exploiting these features, it's possible for the game designer to create mazes and other puzzles regarding odd navigation.

Also notice that this class makes use of a private method. This method is only used as support methods for the destructor of the class, thus not made available in public. It searches all the exits of any room connected to this room, and removes those exits, preventing players from attempting to "enter" a room that no longer exists.

The Programming Project

Using the room class, create a tool that will allow the user to build and connect rooms. The user can traverse the rooms, create new rooms, modify room descriptions, and connect existing rooms. The user has access to the following commands:

Command	Function
n or north	move north
s or south	move south
e or east	move east
w or west	move west
l or look	show room description
x or exits	list exits from current room
c or connect	connect this room to another (already existing) room
r or rename	rename the current room
d or desc	change the description of the current room
a or add	add a new room to the game
q or quit	quit the roombuilder tool
?	show a help menu

A sample executable file can be found here.

All commands should be robust. That is, your program should perform a great deal of error checking while trying to process each command. For example:

- The user cannot move in a direction if no exit exists there.
- When connecting two rooms, the user cannot break existing connections (this might cause another room to be leaked). This means the user cannot make a connection or add a new room while in a room with four connected exits.

Students are encouraged to *run* each other's code and offer feedback on the interface.

---

2. Extra Credit for the Programming Assignment

Note: extra credit options with *more* in them (e.g. ***5 more points***) are dependent upon the completion of other portions of extra credit first.

(5 points) Implement a "jump" command. This allows the user to jump from one room to another, even if they're not connected. (This will save developer time by not needing to traverse all the rooms between here and there).

(15 points) Implement a "save" and a "load" command. These command will let the user save the collection of rooms *and their connections* so that the entire complex doesn't need to be recreated every time the application is started. Hint: you might find it helpful to assign each room a number during the saving process. Use the numbers to help you save connections. Don't forget to test your save games on rooms with the same name and different descriptions (or different names with the same descriptions).

(5 more points) Allow the user to toggle builder mode ON or OFF at the beginning of the program. When set to OFF, the game is in play mode, and many features (those that allow building) will be disabled. (This command is useless without being able to save and load the rooms).

(10 points) Implement an `Object` class. `Object`s can be placed in rooms, but also picked up and carried around by the user. All `Object`s have names and descriptions ("A Hammer", "Something you might pound a nail with."), and know what room they belong in (their home room). When an `Object` is in a room, it should appear in the room's description ("There is a hammer here"). When the `Object` is removed this message does not appear. You will need to modify the `Room` class to have a vector of pointers of `Object`s. The user will also have a vector of `Object`s which can be examined using the i or inventory command. The user must have a command to pick up objects as well ("Get Hammer").When a user picks up the object, an appropriate confirmation message should appear ("You get the Hammer"). The user will need a way to look at the mobile and read its description. (Note that the design tool must allow for the creation of objects as well).

(5 more points) Including the following feature for objects: objects can be *movable* or *locked down*. If an object is movable, the user can pick it up and add it to the inventory. A locked down object cannot be picked up (it is only there as decoration). When a movable object is taken a *success* message appears ("You take the Hammer."), otherwise a *fail* message appears ("Further inspection reveals the Hammer is chained to the workbench."). These messages can be different for each object. (Note that the design tool must allow for the modifications of these messages, and the object's status).

(5 more points) Including the following feature for objects: objects can either be *sticky* or *homing*. A sticky object stays in place when dropped (and is added to that room's vector of Objects). A homing object returns back to its home room (see above). For this portion of the extra credit, you must, of course, implement a drop command that allows users to remove objects from the inventory. When a user drops the object, a message indicates to the user what has happened to it ("You drop the Hammer on the floor.", "The Hammer vanishes in a puff of smoke."). These messages can be different for each object.(Note that the design tool must allow for the modifications of these messages, and the object's status).

(10 points) Create a **`Mobile`** class. **`Mobiles`** are objects that move around as the player plays the game (e.g. monsters, robots, beastly fidos, etc.). Each Mobile has a name, a description, and a room that it currently occupies. After each command the player enters, the mobile has a chance of moving from the room that it's currently in, into another room. (A simple algorithm is to assign percentages to attempting to move in each direction, and another percentage to not moving - obviously the mobile cannot move in a direction where there is no working exit). Thus, if the player/builder is working in a room and entering several commands, the mobile might enter and leave several times. Messages should be printed to the screen when a mobile enters or leave the room that the player is in ("A gelatinous mass oozes in from the north." or "The slime slithers away to the south.") Rooms will now need to have vectors of (pointers to) **`Mobiles`** to track their contents.(Advanced programmers may find an easy way to do this using inheritance). The design interface must provide a way to introduce mobiles into the world. The user will need a way to look at the mobile and read its description.

(10 more points) Give **`Mobiles`** the ability to follow other **`Mobiles`** around. Not all mobiles should have this property, so create an attribute to toggle following on or off. This property can be useful for creating "packs" of creatures. When creatures are following, the user should be alerted as the creatures move ("Grumpy Dwarf arrives from the south, following Snow White.", "Grumpy Dwarf chases after Snow White.") (As always, allow for the creation and use of these new abilities in the interface).

(5 more points) Allow **`Mobiles`** to pick up and drop objects as they move through the rooms. Not all mobiles should have this ability, so create an attribute to toggle it on or off. When a mobile enters a room with an object, give a percentage change that the mobile will pick the object up *instead* of moving. When the mobile has an object in its inventory, there is also a possibility that it may drop the object instead of moving. Mobiles can carry multiple objects. Make sure carried objects are displayed when the

user looks at the mobile. ("Marvin, the paranoid android, is made of dull gray metal. When he speaks, his voice just drones on and on..... Marvin is carrying: a small locket, a thin dagger made of ice.") The user should also be alerted when a mobile picks up or drops an object (assume they're in the same room) ("Marvin, the paranoid android, a small locket." "Marvin, the paranoid android, drops a this dagger made of ice."). Note that if you have implemented locked down objects (above), other messages will be needed. ("Marvin, the paranoid android, strains himself trying to pick up the piano.")

WHAT TO SUBMIT:

For this assignment, you must submit:

- Your C++ source code. (Should be several files).
- A Word document with your the written documentation for your program, including a user manual, system manual, and sample program runs. (***Suggestion***: For sample program runs, have your program echo all output to a file, and submit that file). Your sample program should thoroughly test your program. You may wish to write test programs for your new classes before you write your final program.

ACADEMIC HONESTY:

You must compose all written material yourself. All material taken from outside sources must be appropriately cited. If you need assistance with this aspect of the assignment, see a consultant during consulting hours.