Homework 1: Simple Dungeon Maps

**Due: Sunday, September 17th at 11:59pm (BEFORE midnight)**

Many early computer games were ASCII-based. For example, Rogue was a game that used the symbols found in the ASCII character set to represent an area the player was exploring:



Fortunately, this lines up well with the console applications we’ve been making so far. You are going to use that idea to create a simple visual dungeon in the console. Some code will be provided for you to use, since the program will be using some parts of C# that we have not covered in class yet.

# Task Overview

This is a brief overview of the tasks you must complete for this assignment. Specifics are given in the corresponding sections later in this document.

***Read the entire document*** ***before starting!!!***

## Activity 1

* Use the **starter project** provided on MyCourses
* Add **method calls** to the Main method to complete the dungeon’s layout
* Your completed dungeon should look *similar* to the **sample dungeon** in the “Activity 1” section later in this document

## Activity 2

* Now that you have a small dungeon, expand it by adding **at least two more rooms**
* Make sure **each room is connected** by one or more hallways
* Add the **various items** as described in the “Activity 2” section later in this document

Even though there are two separate activities, you will simply be using and submitting *one* C# project for this assignment. This may different slightly from assignments given to students in other sections of 105.

# Activity 1: Finishing the Dungeon

Begin this activity by unzipping and opening the starter project from MyCourses. You will *not* be creating a new C# project for this particular homework (future assignments will differ).

Running the program at this point will show a partially completed dungeon in the console, as seen to the right:

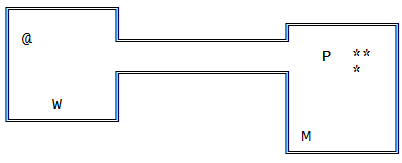
### Code Files

There are two C# files in the project. More detailed information can be found later in this document.

* **Dungeon.cs** – Contains methods you will use to draw the parts of your dungeon:
  + DrawWallVertical() – Draws a vertical “wall” (several characters in a column)
  + DrawWallHorizontal() – Draws a horizontal “wall” (a row of characters)
  + DrawWallCorner() – Draws the specified corner piece
  + DrawObject() – Draws the specified symbol
* **Program.cs** – Where all of your code will go for this homework assignment

The existing code in the main method draws part of a dungeon already. Your job for this activity is to continue calling methods to finish the dungeon so it has the same basic layout and symbols as the sample dungeon below. The sizes of your rooms and hallway *do not* have to match exactly!

### Sample Dungeon



The sample dungeon above was created in notepad to give you an idea of what a text-based dungeon *might* look like. This table shows the meaning of the symbols in the sample dungeon above:

|  |  |
| --- | --- |
| **Symbol** | **Meaning** |
| ║ | Vertical wall |
| ═ | Horizontal wall |
| ╔ ╗╚ ╝ | Corners |
| @ | Your character (the explorer/fighter) |
| \* | Gem |
| M | A monster |
| P | A potion |
| W | A weapon |

# Activity 1: Further Details

Before you jump into the code, here’s a more detailed explanation of what’s going on.

### Map Information

The map for the dungeon is a two dimensional grid. To indicate points on the map, we number them by specifying the X and Y coordinates. (0,0) corresponds to column 0 and row 0. This location is in the *upper left hand corner* of the map. Row numbers increase going *down* the map, while column numbers increase going right.

**Coordinate system example:**

Columns

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 |
| 0 |  |  |  |  |  |
| 1 |  |  |  |  |  |
| 2 |  |  | (2,2) |  |  |
| 3 |  |  | (2,3) |  |  |
| 4 |  |  | (2,4) | (3,4) |  |
| 5 |  |  |  |  |  |

Rows

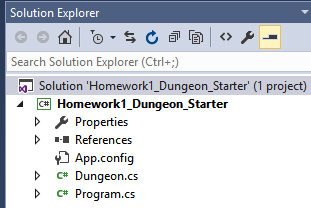
An easy way to design a dungeon is to start with a numbered piece of graph paper and draw the dungeon on it. The numbers will tell you starting points for walls or locations for objects.

### Method Information

Methods defined in the Dungeon.cs code file do the actual work of drawing. You won’t be editing any of these methods. Your job is to simply use them correctly by calling them from the Main method.

The dungeon is written to the screen as you call each method. You can even overwrite previously written parts of the map as your code is executed.

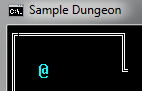
You should read over the comments for the methods in the Dungeon.cs file to see how they work. Pay particular attention to the parameters. After you have done that, take a look at the main method in Program.cs. It shows a partial implementation of the dungeon seen in Sample Dungeon section.



If none of the code files automatically appear when opening the project, you can always *double click* the files in Visual Studio’s “Solution Explorer”, as seen here:

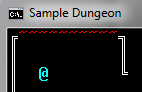
### Optional Parameters in Methods

Each method has some optional parameters, which you may choose to use or leave out. These optional parameters allow you to specify custom colors and characters (symbols) to use when drawing. Here are some quick examples:

dungeon.DrawWallHorizontal(1, 0, 10);

The required parameters (start X, start Y, and wall length) are present in the above method call. The wall will be drawn in the default shape with the default color, as seen to the right.

dungeon.DrawWallHorizontal(1, 0, 10, ConsoleColor.Red, '~');

The two optional parameters (color and character) allow you to change the color of this piece of the wall and the symbol that makes up the wall. Notice how single characters use *single quotes* instead of double quotes. Double quotes are for strings only! Feel free to use these parameters to customize the look of your dungeon if you’d like.

### ConsoleColor and WallCorner

Looking through the existing code in Main, you’ll probably notice the “WallCorner” and “ConsoleColor” parts of the code. These each contain pre-defined options you can choose from when specifying data to give to methods. The technical term for these is “enum”, which is short for *enumeration*.

**ConsoleColor**, an enum that is built in to C#, allows you to specify what color text in the console should be from that point in your code onward. We’ll be discussing this in a little more detail in class.

**WallCorner** is an enum I’ve defined in the Dungeon.cs file. It simplifies the act of drawing corners, and makes the code in Main a little easier to understand at a glance.

To make your life easier, Visual Studio will display a list of potential choices when you type the name of an enum followed by the dot “.” character.

### Summary of this Activity

Your assignment for this activity is to add method calls to the main method to create a dungeon. When you have added all of the required code, the map in your console window should match the general layout of the Sample Dungeon on page 2. The exact symbols and colors you use may change, but the overall appearance (two rooms, a hallway and some objects) should remain very similar.

When you’ve got that done, you’re ready to move on to Activity 2.

## Activity 2: Expanding Your Dungeon

At this point, you should expand your dungeon a little further. The official requirements of your final dungeon are as follows:

* At least four separate rooms connected by hallways (including the two from Activity 1)
* One explorer
* At least three monsters
* At least two weapons
* At least three potions
* Exactly twelve gems

Use the set of symbols defined in the table in activity 1.

It’s easiest to start out by drawing the dungeon on a piece of graph paper. If you don’t have graph paper, you could also map it out in Paint, Excel or Photoshop if you’d like. You don’t need to submit your dungeon plans – they’re just for your own benefit.

## General Expectations

You are expected to use the naming conventions we’ve discussed in class, and to comment your code appropriately. If you have questions on these concepts, refer to the C# Coding Standards document on MyCourses.

Your homework assignments are expect to compile and run without error. You will lose a significant portion of your grade if your assignments don’t compile or they crash during run. This is true for every homework assignment going forward.

## Deliverables

Turn in your homework to the appropriate dropbox on MyCourses. Like practice exercises, you’ll need to zip up your entire project folder before uploading it.

## Grading

See the included grade sheet to see exactly how your grade will be determined.