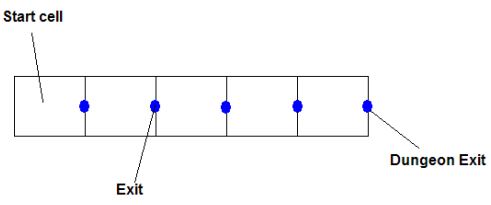
# The Zork Game

In this project, you will be creating a simple adventure game. The game will be text-based or menu-driven similar to the old Zork (<https://en.wikipedia.org/wiki/Zork>) games. The game is played in a dungeon that has between 5 and 10 cells/rooms. The objective of the game is to exit the dungeon while still alive. Upon startup of the game, the dungeon is initialized with at least 5 cells and at most 10. The number of cells is determined randomly. Each cell will have an east exit, a west exit, or both. Each exit leads to an adjacent cell or out of the dungeon. The rightmost cell in the dungeon will have the exit out of the dungeon. See the diagram below.

If the dungeon is to have 5 cells, then the following conceptual structure is to be created:



Notice that the start cell has one exit and the others have two exits.

The game begins with the player in the start cell. The player starts the game with 100 health points. To move from room to room, the user types “**go <direction>**”, where direction can be either “**east**” or “**west**”. If it is possible to go in the specified direction, the player then goes to the specified cell. If it is not possible to go in that direction, the program should display “**Sorry, but I can’t go in that direction.**” If the player reaches the exit, the program should print “**You have beaten the dungeon!**”

There is a **50%** chance that any cell other than the start cell has a monster in it. A monster starts with 20 health points. If there is a monster in the cell, the program should display “**There is a monster here!**” The player has to kill the monster before they can exit the cell to an adjacent cell. There is a **10%** chance of missing the monster. If the player misses the monster the system should display “**a miss**” otherwise the system displays “**a hit!**” If the player hits the monster, 5 health points are deducted from the monster. After which, it is the monster’s turn to hit the player. There is a **20%** chance that the monster misses the player. If the monster misses the player, the program displays “**The monster missed the player.**”, otherwise the program should display “**The player is hit.**” If the monster hits the player, 4 health points are deducted from the player. The fight continues until either the monster’s health points or the player’s health points drop to zero. If the player’s health points drop to zero then the game is over and the program displays “**The player is dead. The game is over!**”

Once a monster is dead, it remains dead for the rest of the game. If the player returns to the cell for some reason, there is no need to fight the monster again.

When the player enters a cell, the program should display the amount of health points the player has left. If the user enters a command that is not recognized then the program should display “**I do not know what you mean.**” Feel free to add any additional messages that you want.

The player may find a single weapon in one of the cells. The weapon can either be a sword or a stick with equal probability. If there is a weapon and a monster in the same cell, the player can pick up and use the weapon. If the player has a sword then they do an additional 3 points of damage to the monster. A stick causes an additional 1 point of damage. The player may take the weapon to other rooms and use it there once they have it.

You are required to design and then program this game in C#. The program may be menu-driven if you wish. You may add other features such as additional types of weapons or monsters to the game. Use inheritance appropriately.

# Restrictions

* Use good object-oriented principles.
* Your design **must** use **arrays** and/or **Lists** where appropriate and useful.
* All **input** and **output** is restricted to the **driver** program.
* Use **inheritance** where appropriate. For example, the **Sword** class and the **Stick** class must both derive from a **Weapon** class (possibly abstract). The **Player** and the **Monster** classes must derive from a **Participant** class (possibly abstract).
* Use exception handling to deal with any unexpected situations that are encountered.

# Hint: Percentages

There are several places in this assignment where you are asked to do or not do something a specified percentage of the time. The easiest way to handle this is to use an object of the Random class to generate a random integer between, say, 0 and 10,000. If the generated random number is in the first “N percent of the range” where N is the percentage of the time the action should be taken, do the action. Otherwise, do not do it.

For example, if **actionX** is to be done **20%** of the time, **20%** of 10,000 is 2,000. The following code shows how you might use this face.

**Random r = new Random();**

**…**

**if(r.Next(10000) < 2000)**

**{**

**// do actionX**

**}**

**else**

**{**

**// do not do actionX**

**}**

# Extra Credit

* **Up to 5 bonus points:** Derive at least 3 types of **Monsters** (e.g., **Cyclops**, **King Kong,** and **Frankenstein**) from the **Monster** class, each with its own number of health points and its own amount of damage it can inflict. When assigning a monster to a cell, first decide whether some type of monster will be in the cell based on probability. If so, select one of the types randomly (with equal probability) to assign to that cell. A cell can have 0 monsters or one total monster from among all the types. There is still only a **50%** chance of some monster being in a given cell. Also derive at least 3 other types of **Weapons** (e.g., **Knife**, **Laser**, **Bomb**, or **Gun**) from the **Weapon** class in a similar way.
* **Up to 5 bonus points:** Make the dungeon two-dimensional with at most 5 rows of cells (3 rows minimum) and at most 10 columns of cells (4 columns minimum). You will need to add the ability to move **north** and **south** as well as **east** and **west**. The dungeon has only **one entrance** and **one exit**. The **entrance** is in the **top, left** cell. The **exit** is on the **east**, but may be on **any row**. Set the **exit** row **randomly** with equal probability that is on any row.
* If you plan to attempt the extra credit part, your design **must** include that.
* Additional features such as sound or video that make your solution more interesting are welcome. You can add any additional features you want to make your game more interesting, you just must meet, at the bare minimum, the specifications given.

# Project Documentation (via Comments)

**Project documentation is NOT optional in this course.** It is absolutely required in **every** programming assignment. See the **Documentation Standards** on D2L for details and for examples of proper documentation.

An otherwise perfect project may still receive a **failing grade** if the required documentation is missing, incomplete, or poorly done. Examples of proper documentation appear in the course **Documentation** **Standards** document

# Submission

Create a **design document** and include it in your submission.

You may work in groups of **two** or **three** on this assignment. Each person should contribute approximately the same amount of the work; be clear in the comments of your code who worked on what. Names of all team members should be included in the names of the files you submit, e.g., ***1260-NasiumJim-HandLinda-Project5*** may be the name of the submission by Jim Nasium and Linda Hand. Only **one** team member needs to submit the code to the dropbox.

# Example Output

The following pages shows an example of the interaction between the program and the user for one game. **This is only an example**. You are encouraged to use your imagination to make the program even more user-friendly and fun to play. Remember that the driver is responsible for all user interaction.

In the example output below:

The game status shown at each stage uses a **P** to indicate where the **player** currently is, an **M** to show the presence of a **Monster**, **St** to show a **Stick**, and **Sw** to show a **Sword**. The vertical lines mark the boundaries of a cell. The spaces displayed between the cells are for readability only. If you create additional **Weapon** and/or **Monster** types, use an appropriate strategy to distinguish between them clearly in the game dialog. Program input is not case-sensitive.

