# Deliverable 5

For this assignment you will focus on Class Libraries, static classes, LINQ, and subforms. You will expand off of your core objects you have in Deliverable 4. You will need to place all of the custom core classes in a class library. You will need to write new Door and DoorKey classes. You will modify the Potion, Map, and Hero classes. You will need to create proper associations amongst all of your classes as appropriate. You will need to create a UML class diagram. You will need to create a WPF form that tests your objects and classes.

Some things that might be useful to understand and use for this implementation will be static classes, referencing of separate projects within a solution, and opening and closing forms. Object references in this project should be of the type specified.

You will need the following:

## Class Library

You will need to add a class library project to the solution, this will contain all of your custom classes: Actor, Door, DoorKey, Hero, IRepeatable, Item, Map, MapCell, Monster, Potion, and Weapon. Your WPF application will be in its own project. Reference this Class Library project from your WPF application project.

### DoorKey

Create a DoorKey class. DoorKey inherits from Item.

Add a string field for a code. The code will be used to match a Door.

Create an overloaded constructor that accepts the name, value, and a code and calls the base overloaded constructor. The code will need to be set in the overloaded constructor.

Expose the Code filed using a read-only property.

### Door

Create a Door class. Door inherits from Item.

Add a string field for a code. The code will be used to match a DoorKey.

Create an overloaded constructor that accepts the name, value, and a code and calls the base overloaded constructor. The code will need to be set in the overloaded constructor.

Add a method to check if a given DoorKey matches this Door. The method accepts a DoorKey as a parameter and return true if the code of the DoorKey matches the code for this door. Return false if they do not match.

### Potion

Create a colors enum that will contain a unique color for each type of potion you have defined in your Map Class. For example, if you have 4 different types of potions (small healing, medium healing, large healing, extra-large healing, etc.), you should have 4 different colors defined in the enum. These can just be by name.

Change the datatype of the Color property and field to be your enum.

Fix any changes in your classes that need to be adjusted because of this change such as, removing the using statement for the system media color and updating data types.

## Map

The collections of Potion, Monster, and Weapons must be implemented using dynamic generic collections. Your MapCell collection should still be a non-dynamic type of collection, like maybe an array.

Modify the method to fill your collection of MapCells so that it does the following:

> Randomly determine if the MapCell contains a randomly selected Potion or nothing. Make sure to create a DeepCopy when filling the MapCell with a Potion.

Add a method that returns the MapCell that is at the current position of the Hero based on the X and Y location on the Hero object. If the hero is not on the board return a new MapCell.

## Hero

Add a boolean field IsRunningAway. This will track whether the hero is running away from a monster. You will need to expose this field through a public property.

Add a DoorKey field to keep track of the DoorKey if found. You will need to expose this DoorKey as read-only. There will be no public setter for this field.

Add a read-only property to get the Hero's AttackValue. If the Hero does not have a weapon equipped, his/her AttackValue is 1. If a weapon is equipped, the AttackValue is the AffectValue of the weapon that is equipped.

## MapCell

Nothing will be done to MapCell in this deliverable, it just needs to be included in the class library.

## Actor

Nothing will be done to Actor in this deliverable, it just needs to be included in the class library.

## IRepeatable

Nothing will be done to IRepeatable in this deliverable, it just needs to be included in the class library.

## Item

Nothing will be done to Item in this deliverable, it just needs to be included in the class library.

## Monster

Nothing will be done to Monster in this deliverable, it just needs to be included in the class library.

## Weapon

Nothing will be done to Weapon in this deliverable, it just needs to be included in the class library.

### UML Diagram

Include the UML diagram for your class library. Make sure to update the diagram to properly show any and all associations.

## WPF Test Application

Add the Class Library Reference to the WPF application.

### Game

This class will be static and will keep track of our game. It will be in the WPF Application project. Do not forget to add a namespace reference.

Add an enum for GameState, it has three values; Running, Lost, Won.

Add a static GameState field. This will track the state of the game.

Expose the GameState field in a public property. If the Adventurer on our Map is ever dead, the GameState is Lost. Add this logic to the getter of the GameState.

Add a static Map field. This will keep the Map object of our game.

Expose the Map field using a read-only property.

Create a public static method called ResetGame. This method accepts the height and width of the gameboard as parameters. This method will set the GameState to Running, instantiate a new Map object based on the given height and width, and set the Adventurer on the Map to a new Hero. The Hero will be placed at a random position on the Map where the MapCell does not contain anything.

### frmFoundIt

Create a subform that will pop up anytime that the hero is in a MapCell that contains a Potion and contain the following:

> Text informing the user that the hero entered a MapCell that contains an Item needs to be shown.
>
> An “OK” button that closes the form when clicked.
>
> No other window in the application can be used until this window is closed.

### frmMain

The main window will need to contain the following:

> There should not be any variables on this form relative to any of our custom classes, everything will be handled through and using the static Game object.
>
> A grid control to show the current contents of the map.
>
> A method to populate the grid with controls to show if any cell contains a Potion. Also display the location of the Hero based on the Hero’s x and y. This is referred to as drawing the map.
>
> A method that displays the Hero’s name and current X and Y position.

Buttons to move the hero up, down, left, or right. After moving the hero on the Map, you will need to call the methods to draw the map and update the hero stats. After the Hero has moved, if the Hero location contains a cell that contains a Potion, show the Found It Form.

You will not lose points if the Hero wanders off the board.

A button to start/reset the map. When pressed, call the ResetMap method on the static Game object and the methods to draw the map and calculate the statistics.

A method that displays to the form using LINQ and Lambda expressions, the following statistics:

The HP of the Monster with the most HP in your monster list.

The affect value of the Weapon which does the least damage from your weapons list.

The Average healing affect value for all of the Potions in your potions list.

## Grading

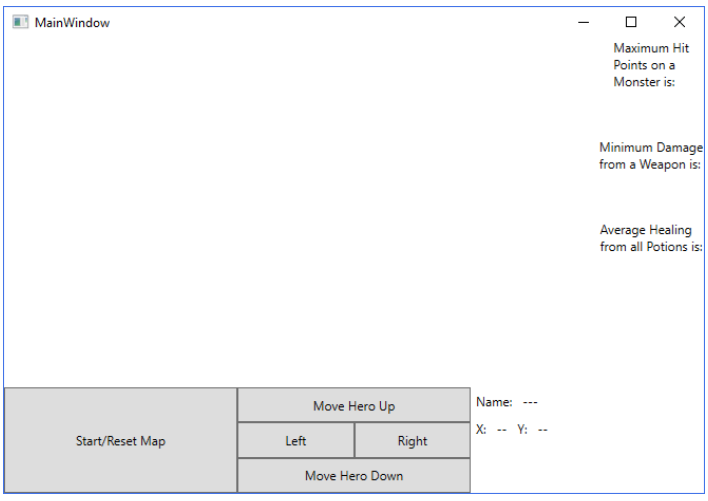
Points will be awarded as to how well the classes match the requirements given in this document. Points can be lost for unprofessional looking code or applications. There are no specific expectations on presentation of your form as long as it is able to show the object behaviors needed and is professional in appearance.

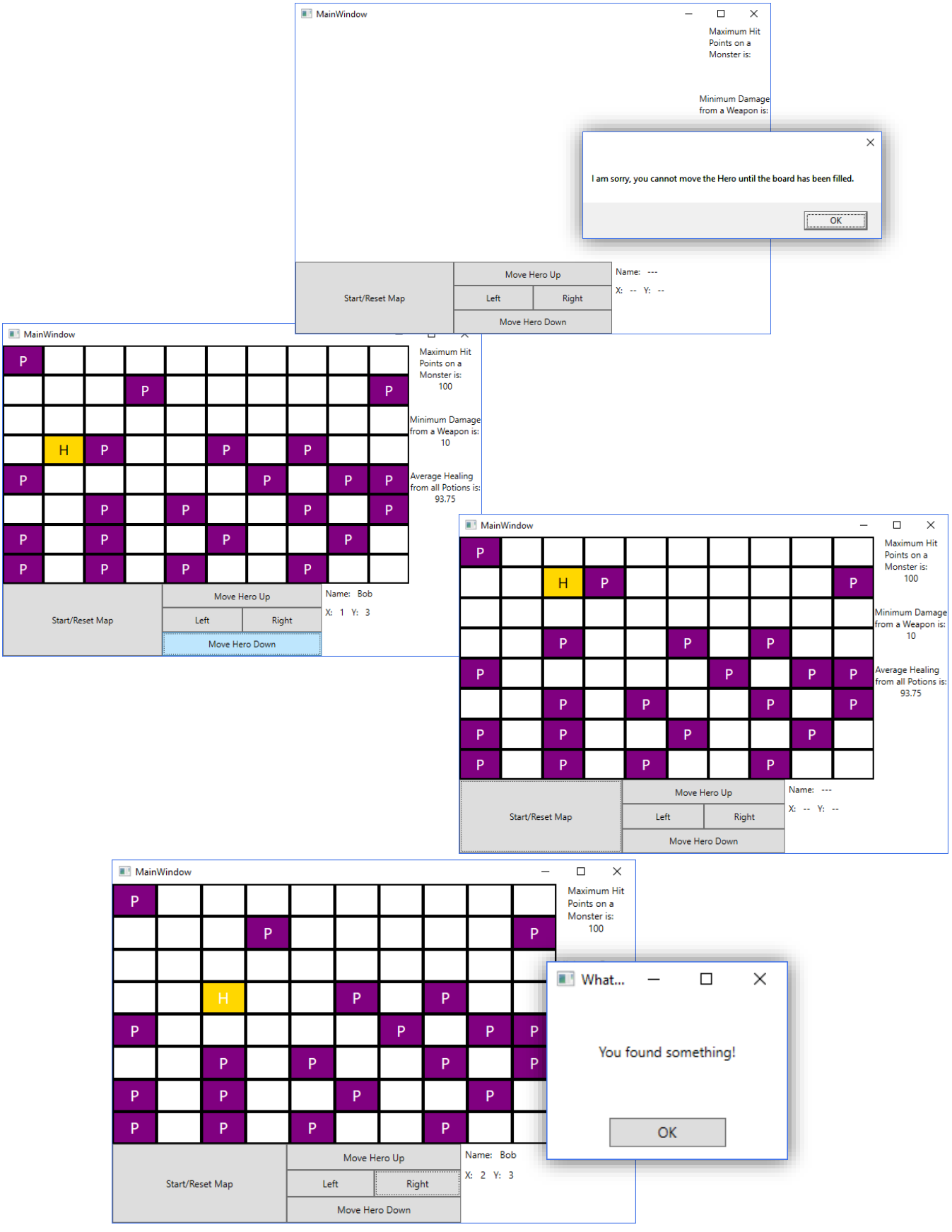
## WPF Test Application

You will need to create a WPF application to test and prove your objects work the way expected. Use Grids, TextBlocks, Buttons, StackPanels, Windows, and any other controls to create a test environment for your code. All existing object behavior is should still be available in your classes, but the application should specifically prove the following:

- A map object is created and filled randomly with Potions.
- Refreshing the map, will repopulate the map randomly.
- Display the contents of each cell in the map.
- The hero will move around the map respectively to the appropriate button press.
- If a non-empty is discovered, the subform is displayed as a modal.

Here is an example of what your form might look like:




MainWindow
Maximum Hit Points on a Monster is:
Minimum Damage from a Weapon is:
I am sorry, you cannot move the Hero until the board has been filled.
OK
Start/Reset Map
Move Hero Up
Left
Right
Move Hero Down
Name: ---
X: -- Y: --
MainWindow
Maximum Hit Points on a Monster is: 100
Minimum Damage from a Weapon is: 10
Average Healing from all Potions is: 93.75
Start/Reset Map
Move Hero Up
Left
Right
Move Hero Down
Name: Bob
X: 1 Y: 3
MainWindow
Maximum Hit Points on a Monster is: 100
Minimum Damage from a Weapon is: 10
Average Healing from all Potions is: 93.75
Start/Reset Map
Move Hero Up
Left
Right
Move Hero Down
Name: ---
X: -- Y: --
MainWindow
Maximum Hit Points on a Monster is: 100
What...
You found something!
OK
Start/Reset Map
Move Hero Up
Left
Right
Move Hero Down
Name: Bob
X: 2 Y: 3