Deliverable 6

For this assignment you will focus on Class Libraries, Polymorphism, Operator overloads, and subforms. You will expand off of your core objects and game object in Deliverable 5. You will need to place all of the custom core classes in a class library in your solution. You will modify the 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 polymorphic behavior, writing operator overloads, referencing of projects within a solution, and GetType and typeof method calls. Object references in this project should be of the type specified.

You will need the following:

Class Library

You will need to use your class library project to the solution, this will contain all of your custom classes. Your WPF application will be in its own project. Reference your Class Library project from your WPF application project. This is the last deliverable we will modify classes in the Class Library. After this Deliverable, we will consider the Class Library and its classes "locked". There will be no instructions to modify them in the next deliverable. You can of course still edit them in the next deliverable, but you will want to make sure that these classes are polished for completion of this Deliverable.

DoorKey

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

Door

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

Potion

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

Map

Remove the Weapon and Potion field collections. These will be replaced by a single collection of type Item.

Modify your code so that the weapons and potions are added to the collection of Items instead of the previous Weapon and Potion collections.

The collection of Monsters field will not be modified.

Remove the public accessors to the Monster, Potion, and Weapon collections. The Item (Weapon and Potion) collection and Monster collection will no longer be publicly accessible.

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

> After the MapCell is created, the randomly determine if the MapCell is filled with either a randomly selected Item (Potion or Weapon) or a Monster. Get the Items from the Item

collection and the Monsters from the Monster collection. Make sure to create a DeepCopy when filling the MapCell with an Item or Monster.

Not every MapCell will be filled. Make sure you only fill some but not all of the MapCells. Remember that if a MapCell can only contain a Monster or Item; it cannot have both.

Randomly place a single Door in a MapCell where there is not a Monster or Item.

Randomly place a single DoorKey in a MapCell where there is not a Monster, Item, or Door.

The Door and DoorKey must have matching codes.

Create an read-only property to get the current location of the Hero(Adventurer). This property will return the MapCell relative to the x and y position in the MapCell collection relative to the location of the Hero as the Hero's current location.

Create a method that Moves the Hero based on the following criteria:

Accept a parameter for the direction that the Hero is to move. **Hint**: the Actor already has a move direction.

The method needs to tell the Hero to move but only if the Hero will not go off of the gameboard. Use the MapCell collection to tell if the Hero will move off of the gameboard.

If the direction to move, would cause the Hero to travel off of the edge of the gameboard, the Hero will not move. For example, if the Hero's location is x:0,y:0 s/he cannot move up or left; but can only move right or down.

Hint: 0 is the left-most or top-most cells' index. Think about what would be right-most and bottom-most.

Mark the cell as seen where a hero is currently at.

Return true if the Hero's new location requires the Hero to act. The Hero needs to act if the Hero's location contains an Item or Monster. This is checked after not before the Hero has moved.

Add the following read-only properties. Each of these will not be able to use Lambda Expressions and LINQ to get the values requested. You will need to iterate across each cell and check the contents of each cell in the collection.

Return the count of how many Monsters are in the Map.

Return the count of how many Items are in the Map.

Return the percentage of how much of the map has been discovered.

Hero

Add a method that applies an Item to the Hero. This method accepts an Item as a parameter and returns an Item. The logic of this method will do the following based on the type of the Item passed as a parameter:

> If a Potion, heal the hero based on the AffectValue. Return null;
>
> If a Weapon, equip the Weapon by setting the Equipped Weapon field to the new Weapon. If a previous Weapon was equipped, return the previously equipped Weapon. If no Weapon was previously equipped, return null. **Hint**: return the previously equipped weapon.
>
> If a DoorKey, equip the door key by setting the DoorKey field to the new DoorKey. If a previous DoorKey was equipped, return the previous DoorKey. If not, return null; **Hint**: return the previously equipped DoorKey.
>
> If any other type, simply return the Item passed as a parameter.

Add a read-only Property that return true if the Hero's is alive and false if the Hero is dead.

> The Hero is alive as long as the current HP is greater than 0.

Overload the + operator for the Hero so that it follows the following guidelines:

> The format will be Hero + Monster.
>
> If the Hero's AttackSpeed is greater than the Monster's AttackSpeed, then the Hero will apply damage to the Monster first based on his/her AttackValue. If the Monster is still alive after the Hero hit it, the Monster will apply damage to the Hero based on the Monster's AttackValue.
>
> If the Monsters' AttackSpeed is greater than the Hero's AttackSpeed, then the Monster will apply damage to the Hero first based on its AttackValue. If the Hero is still alive after the Monster hit him/her, the Hero will apply damage to the Monster based on the Hero's AttackValue.
>
> If the AttackSpeed of both the Hero and the Monster are the same, they will both apply damage to each other based on their respective AttackValues. It will not matter who applies damage first because both will apply damage regardless.
>
> If the Hero is running away, the Hero cannot attack. If the Hero's AttackSpeed is greater than the Monster's AttackSpeed, then the Monster does no damage and the Hero gets away undamaged. If the Monster's AttackSpeed is equal to or greater than the Hero's AttackSpeed, then the Monster applies damage to the Hero based on the Monster's AttackValue.
>
> Return true if the Hero is still alive after all damage has been dealt. Return false if the hero is dead.

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

Game
Include the static Game class from Deliverable 5.

frmItem
Create a subform that will pop up anytime that the hero is in a MapCell that contains and Item.

The name of the Item found is shown to the user.

An “OK” buttons closes the form.

No other window in the application can be used until this window is closed.

frmMonster
Create a subform that will pop up anytime that the hero is in a MapCell that contains and Item.

The name of the Monster found is shown to the user.

An “OK” buttons closes the form.

No other window in the application can be used until this window is closed.

frmMain
The main window will need to contain the following:

A grid control to show the current contents of the map.

A method to populate the grid with controls to show the locations of any Item or Monster, the Door, the DoorKey, and the Hero. If the cell contains nothing and the Hero has discovered it, mark it with an X. We will call this drawing the map.

Buttons to move the hero up, down, left, or right. After moving the hero, you will need to call the method to redraw the map.

Whenever the map redraws, update the number of Monsters and Items on the map. Also you will update the percentage of the map discovered.

A form level event listener that listens for and handles the keypress event. It will need to handle the up, down, left, and right arrow keys. When pressed, fire the appropriate button listening event is called.

A button to refill the map. When pressed, call the ResetMap method on the Game and the method to redraw the map.

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.
- 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 or key press.
- The hero cannot go off of the map.
- When an empty cell is discovered, an x in placed in the cell. If a non-empty is discovered, the appropriate subform is displayed as a modal.
- Update the statistics each time the map is redrawn.

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