CIS 1280 C# / .NET I

Cowboys and Aliens Game

# Inheritance, Interfaces and Polymorphism

# Turn in requirements:

1. Please name your Program LastnameP7, such as NelsonP7.
2. Please staple a printout of your Form1.cs file to this page to hand in.
3. Check program into source control.

# Program Requirements:

1. **3 pts** Write your name, email address and file name at the top of your source code in a comment.
2. **5 pts.** Put your name and program title in the Text Property of the Form.
3. **5 pts** Use good C# programming style and formatting for your program.
4. **5 pts** Use appropriate comments to explain what you are doing. All public members of your class should use XML commenting (triple slashes). Group all your fields, properties, methods and constructors under #region/#endregion tags.

This is a group project. Assign tasks to members of your group. All members should have an equitable number of tasks. You will be graded on your individual checkins to the Visual Studio project.

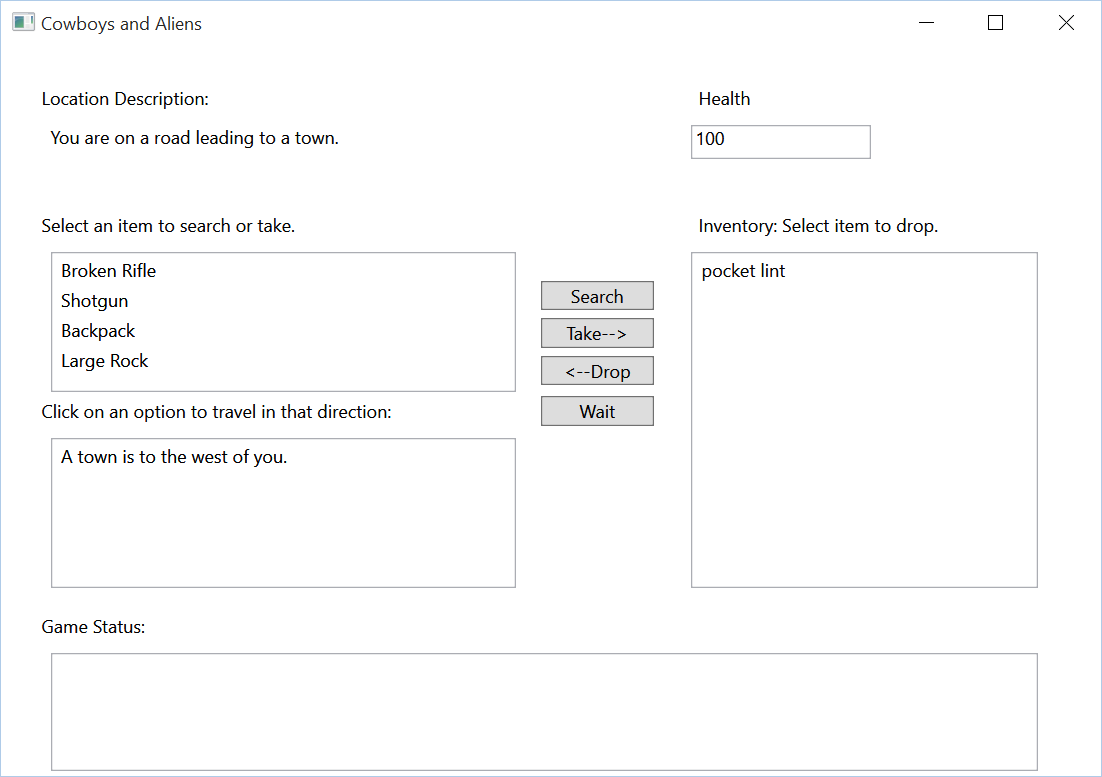
We are going to write a program that simulates the action in the movie, ***Cowboys and Aliens.*** It is 1873 in a small town in New Mexico and the aliens have arrived to invade the earth and take it over for their own nefarious purposes. There are seven aliens on a recon mission to gather intelligence for the initial invasion. Your mission is to take out the seven aliens. There is no one else.

There are four types of weapons in this game: a Colt 45 Peacemaker (Six-shooter), a Sharps “*Big Fifty*” rifle, a large spear, and a stolen alien-manufactured gauntlet that shoots death rays. The aliens have superior weapons and are stronger, but they can be killed with your weapons, but they must be hit twice.

The aliens are approaching your town. If you can find them and take them out, the aliens will assume that the earth is too hard a target and they will leave.

Create a new C# WPF Application

The MainWindow will look like:



Items on the form include:

A Location Description TextBox that describes the location your are at.

An Items ListBox that lists the items at that location you can take or search.

A Travel Options ListBox displaying the travel options from that location.

A Game Status ListBox displaying Game Status messages.

An Inventory ListBox displaying the items in the player’s Inventory.

A Health TextBox displaying the player’s health.

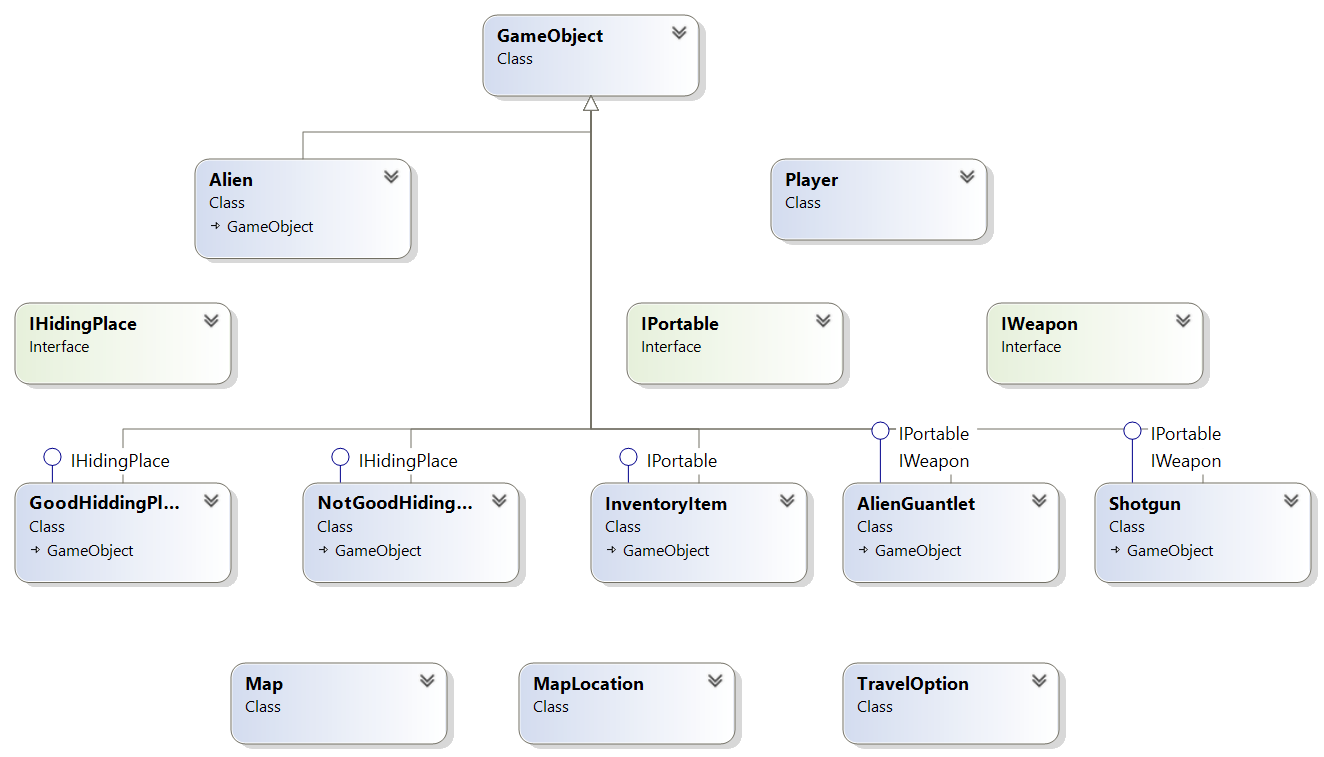
A search button which allows the player to search the item selected in the Items ListBox.

A take button which allows the player to take the items selected in the Items ListBox and add them to their inventory.

A drop button which allows the player to drop items from their inventory (they will then show up in the Items ListBox for the location.

A Wait button which allows the player to do nothing for one turn.

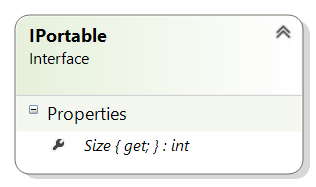
You will write the following classes:



Game Object:

This is the base class for all objects in the game. All items must have a Description.

IPortable Interface:

The IPortable interface must be implemented by objects that can be taken into the player’s inventory. It requires the implementing class to have a Size property that consists of an int that indicates the number of inventory slots taken by the item.

InventoryItem:

InventoryItem implements the IPortable interface and is basically used for generic items that can be taken into a players inventory. Things that might become InventoryItems could be things like pocket lint, backpack, broken rifle etc.

IWeapon:

IWeapon is an interface that requires implementing classes to have an Attack method that takes the following parameters: A double for range, a double for wind, a double for light. Attack returns an int which corresponds to damage.

Weapon classes:

Each weapon class implements IWeapon and IPortable. Each weapons attack method should implement a different attach method.

The attack method returns number of points of damage as an int.

Range is the range in yards. Wind is a double between 0 and 1 that represents the winds impact on the probability of hit. Light is a double between 0 and 1 that represents the light’s impact on the probability of hit.

The formula for probability of being hit is range probability x (1-wind) x (1-light).

Use the following chart for range probabilities:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Range (yards)** | **Six Shooter** | **Rifle** | **Alien Gauntlet** | **Spear** |
| **5** | 0.99 | 0.99 | 0.70 | 0.99 |
| **25** | 0.80 | 0.80 | 0.90 | 0.80 |
| **50** | 0.65 | 0.70 | 0.90 | 0.50 |
| **100** | 0.40 | 0.60 | 0.85 | 0.30 |
| **500** | 0.30 | 0.50 | 0.80 | 0.20 |
| **1000** | 0.20 | 0.40 | 0.70 | 0.10 |

IHidingPlace:

Classes implementing IHidingPlace must have a HiddenObject property that contains a GameObject. They also must have a search method that yields the HiddenObject (either randomly or right away).

NotGoodHiddingPlace:

This class inherits from GameObject and implements IHidingPlace. NotGoodHiddingPlace’s Search method just returns hiddenObject then sets hiddenObject to null.

GoodHiddingPlace:

This class inherits from GameObject and implements IHidingPlace. In addition to having a HiddenObject as required by IHidingPlace, this class has a Probability property which is a double between 0 and 1 that is the propbability of finding the hidden object if the player searches this hiding place.

GoodHidingPlace’s Search method gets a random number. If that random number is less than probability then it returns hiddenObject and sets it to null. Otherwise it returns null.

Alien:

Alien is a class to represent the aliens. moveProbabiilty is a double between 0 and 1 that represents an alien’s probability of changing it’s location on any given turn. Health is an int representing the Aliens health. Location is a MapLocation object pointing to the Aliens location on the Map. Note: you should always point this to an existing map location and not use new to create a new location not on the map! Weapon is an IWeapon object which represents the weapon the alien is equipped with.

The Alien class constructor will take one parameter (a MapLocation for the alien’s starting location). The aliens Location property will be set to the location passed in to the constructor. IMPORTANT: Also add the Alien to that location’s Items list so that the player will see the alien at that location. The constructor will initialize the Alien’s description (You can make it Alien or randomly assign a name), set it’s weapon to a new AlieenGuantlet. Set Health to a starting value and set the alien’s moveProbability.

The Alien will have an Update() method that will be called by the MainWindow each turn. Update() will randomly determine if the Alien chooses to change it’s position based on the Alien’s move probability. If the Alien chooses to move randomly pick a travel option from that MapLocation’s list of travel options, change the Alien’s location to that new location. Remove the alien from its existing location and add the alien to the new location’s list of items (if you don’t remove it from the current location you will end up having a trail of ghost aliens left all along your map).

Player:

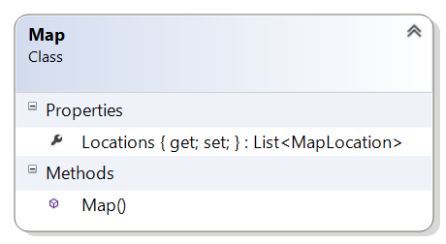
The player class has a Health property that keeps track of the player’s health level. The Inventory property has a List of GameObjects. The player has a Location property which points to a MapLocation. The MaxHealth property is the maximum health Health can be set to. Weapon property has the Weapon currently readied from inventory.

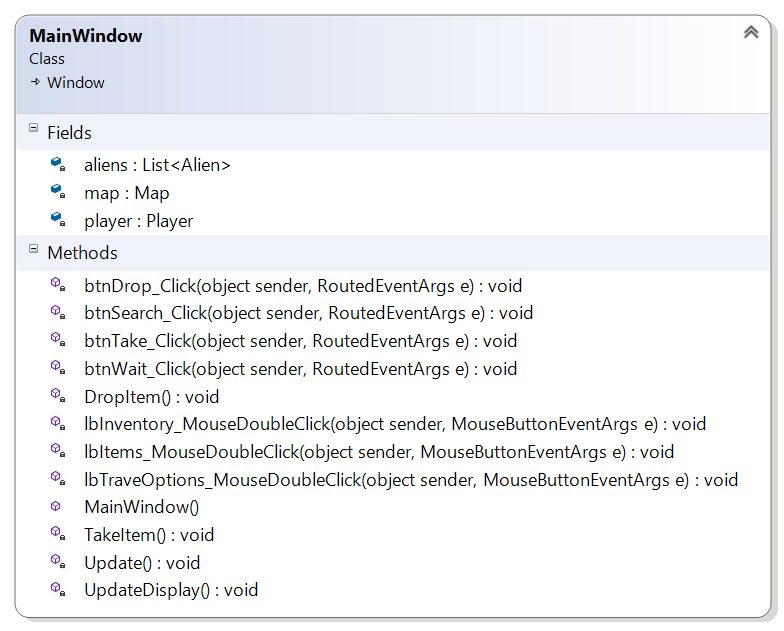
The player’s update method restores a random number of health points to the player each turn up to the player’s MaxHealth.

The player’s constructor takes one parameter, a MapLocation which is used to set the player’s starting location. The constructor should set starting values for all remaining properties.

Map, MapLocation and TravelOptons Classes:

These classes are the same as the ones you created in the previous assignment:



MainWindow:

MainWindow has three items: A List of aliens, a map and a Player object.

MainWindow’s constructor instantiates a new Map, instantiates a new player setting the player’s starting map location, instantiates all the aliens. When you instantiate the aliens you can set their starting location to the same location or randomly scatter them around the map.

The Update method Calls the player’s and each aliens update methods. It then will go through all the GameObjects in the player’s location’s items list. If any of them are Aliens, it will instantiate a fightDialog window so the player can fight that Alien. If there are multiple aliens you can have the player fight each alien one by one or have the fight dialogue fight all at once, your choice. The update method will need to check the outcome of the fight dialogue. If the player wins, remove that alien from the list of aliens. If the player loses display a player loses window. If the result is a draw do nothing. If there are no Aliens left display a player wins window.

The UpdateDisplay updates all the controls in the window with current status.

The Travel options listbox should set the players location to the location clicked on by the player then call the update method.

The Take button should check to make sure the item selected in the location items list box is an IPortable item and that there is sufficient room in inventory then move that item into the player’s inventory, remove it from the location’s list of objects then call update.

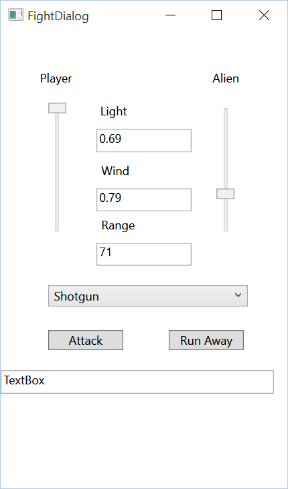
The Drop button should remove the item selected in the player’s inventory and add it to the locations list of items.

The Search button should check to see if the item selected in the location’s object list is an IHidingPlace then call that hiding places search method. If an item is returned by the search method add it to the locations objects list, then call update.

The wait button just displays a message that you wait for a period of time then calls update.

You can implement the list box double click methods to do appropriate action methods if you wish.

FightDialogue:

The fight dialogue needs to be passed the Alien and Player object that will fight (alternatively you can pass a list of aliens if the you are implementing the option where the player has to fight all aliens at once).

You can pass these in to the dialogue through the constructor or with properties or both.

Set light, wind and range to random values in the constructor of the window.

The attack button will pass the Light, wind and range values to the players and the alien(s) attack methods. Subtract the points that return from the attack methods from the player and alien health. Check health at the end of each attack. If the player goes to 0 or all aliens are at 0 close the dialogue. You can use the Outcome string to set the results. The Run Away button sets the results to a draw and closes the Fight Dialogue.

.