Blueprints

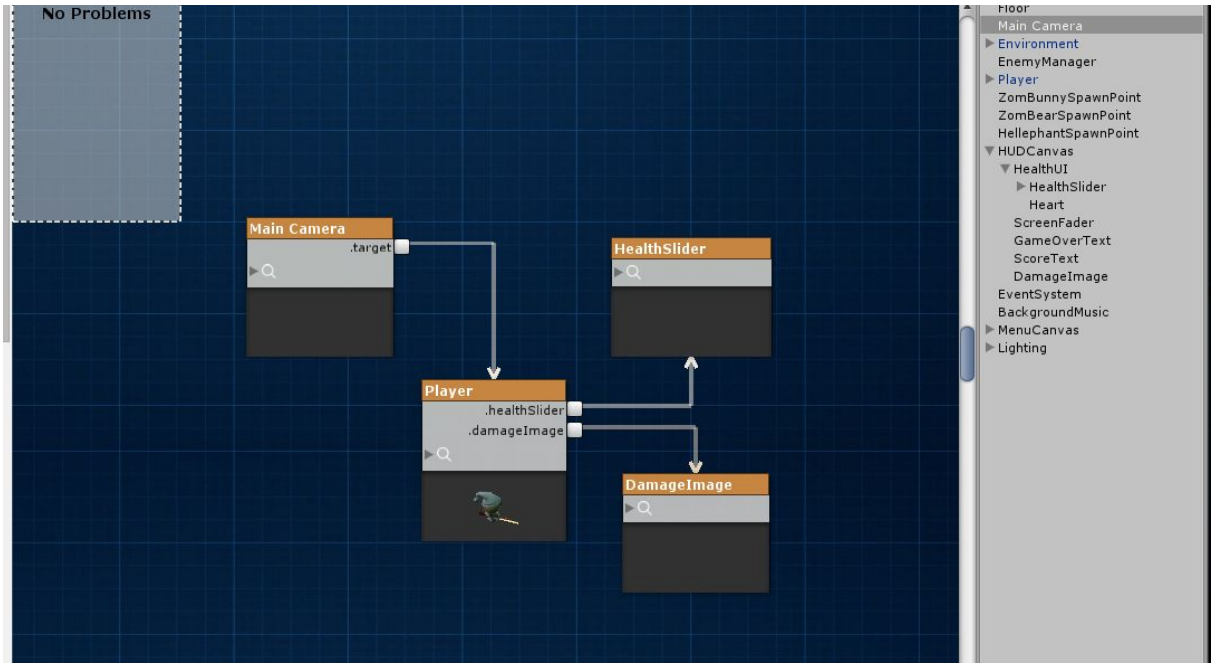
(tutorial)

Blueprints is a tool made to help the process of creating and maintaining unity projects. One thing always bothered me was having to manually update my levels every time I added a new feature. Having to keep track of what scenes were already fixed and what i needed to change and set for each object is just not productive. This is why I started creating something to help me develop faster and avoid having loads of bugs.

Basics

What are dependencies?

When we talk about dependencies we are usually referring to what an object needs in order to work correctly. For example in our unity survival example:



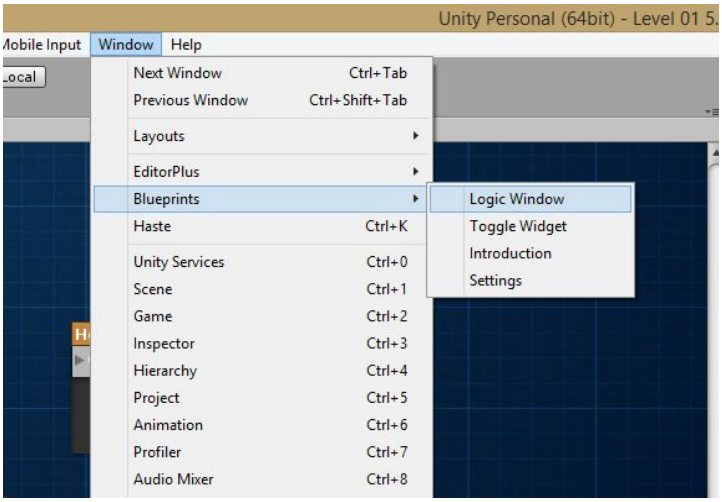
we can use the Logic window to see that our Main Camera has 1 dependency which is the target. It uses this dependency to know who to follow and point to.

How to declare dependencies in the Blueprints context?

Dependencies in the blueprints context are public variables of the GameObject type or anything extending Component. We chose to use public variables because it integrates well with unity's way of serializing objects in a scene. (we are going to extends this to use private Serializable variables in the future as well.)

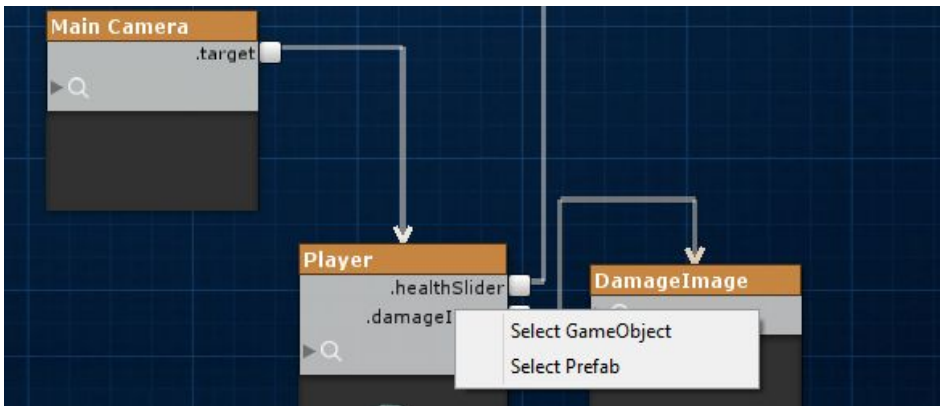
Logic Window

The Logic Window is where we display the game's inner Logic. Selecting an object in the scene will show the dependency graph starting from that object.



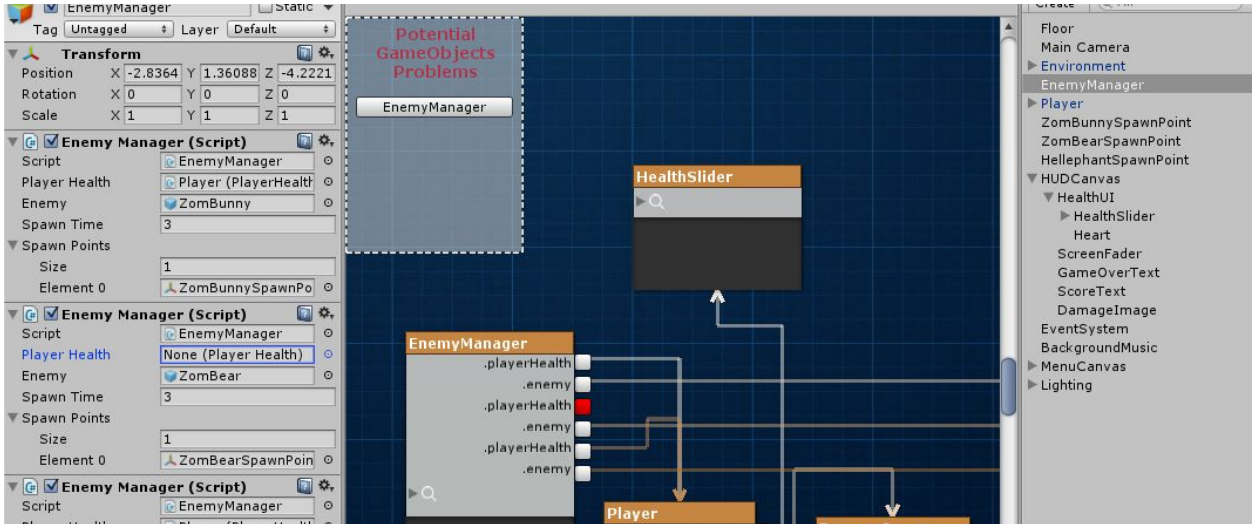
To open the Logic Window we go Window > Blueprints > Logic Window on the top menu.

In here we show objects as nodes and their dependencies as the connections to the other nodes. One can move the nodes around by dragging them with the left mouse button. Selecting them highlights their connections. In the bottom there is a zoom slider as well.



left clicking on an "object" a tooltip will show to help navigate the scene.

Another useful feature is that we highlight missing dependencies in the scene.
By clicking on the warnings in the left menu titled '*Potential GameObjects Problems'* we can navigate and select the objects in our scene that have missing dependencies.

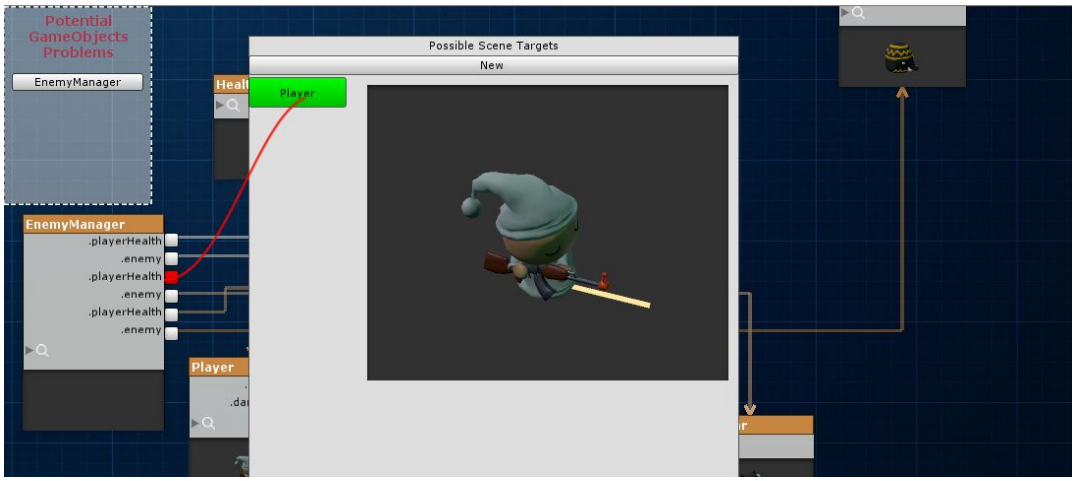


We can see that the EnemyManager has the playerHealth dependency missing

In play mode Logic window shows the private variables as well. This is to help debug while the game is running and spot problems easier.

Setting Dependencies

Clicking on the connection (small squares) opens the connection wizard which shows all the possible objects in the scene that can be the use to set the dependency.
Hovering on any of the targets shows their preview and when clicked sets the dependency to them.



Here we can see we are trying to set the playerHealth dependency

Game Example

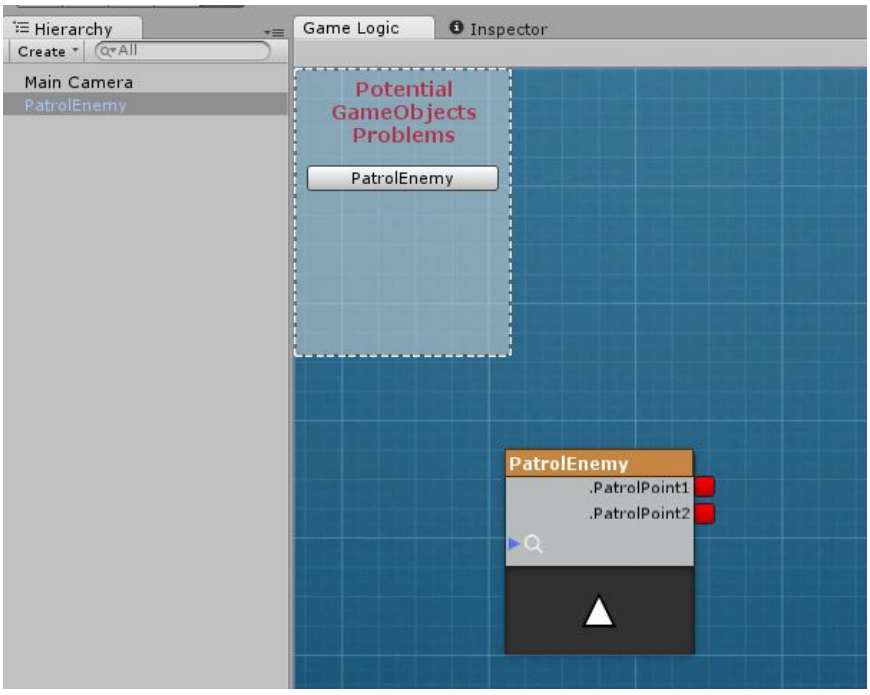
Using Logic Window

To learn how to fully use our tool we are going to make a simple 2d stealth game together.

For our game we need a hero so we create a sprite object in unity. In order to have it do anything we need to attach a behaviour of our own(let's call it HeroController).
Now that we have a hero that moves lets create some enemies to catch it.
Add a new sprite object and attach a behaviour to it (PatrolEnemy).
In order to have move around a bit let's make it patrol between 2 points defined by 2 objects. those 2 are going to be our only dependencies for now.

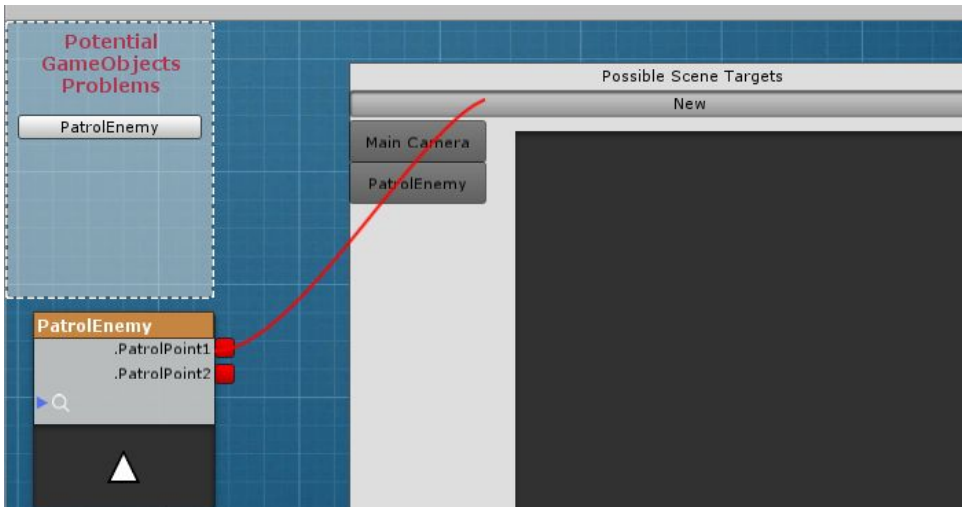
```
public class PatrolEnemy : MonoBehaviour {
      public GameObject PatrolPoint1;
      public GameObject PatrolPoint2;
...
}
```

To help us through this we are going to open our Logic Window.



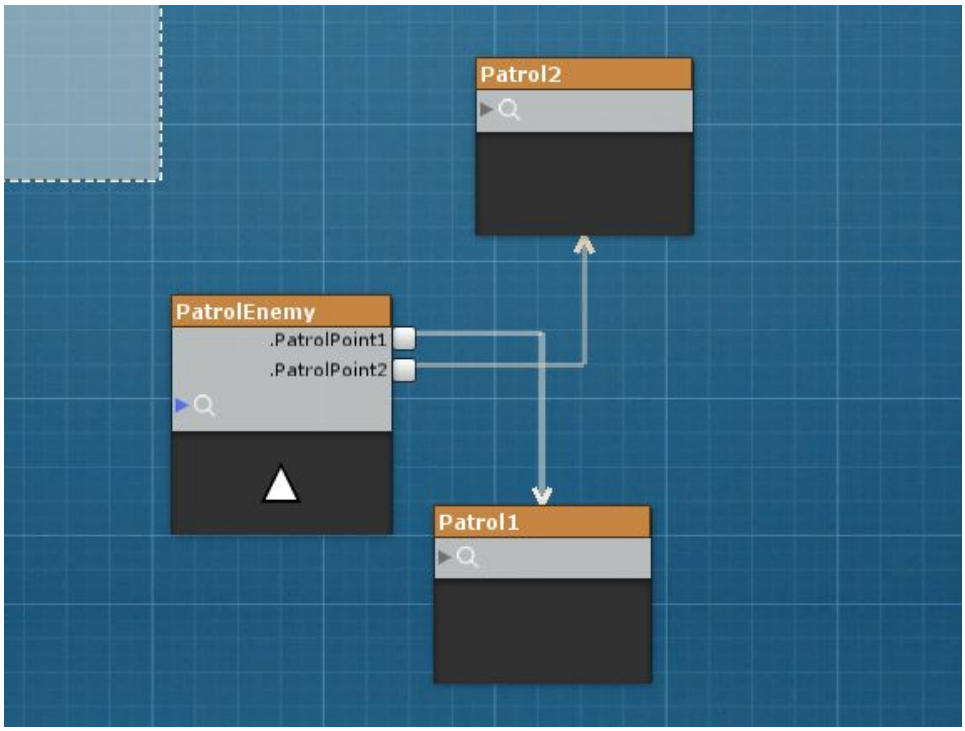
Lets look at our window:
We can see our object with 2 dependencies. They are red because they are missing, we haven't set them.

Because the dependencies are of type GameObject the 2 objects in the scene can be used. We need 2 new objects so let's click on New for both dependencies.



Clicking on the red connections we can start to set them.

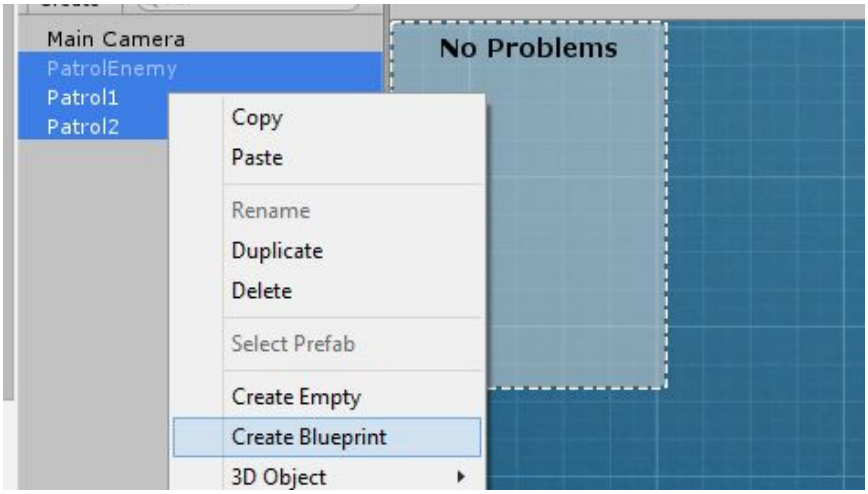
The dependencies are now set the new objects. Change their names to a more appropriate name like Patrol1 and Patrol2. At this point we have one enemy moving between the 2 Patrol objects but we need more if we want to create decent levels.



Creating a Blueprint

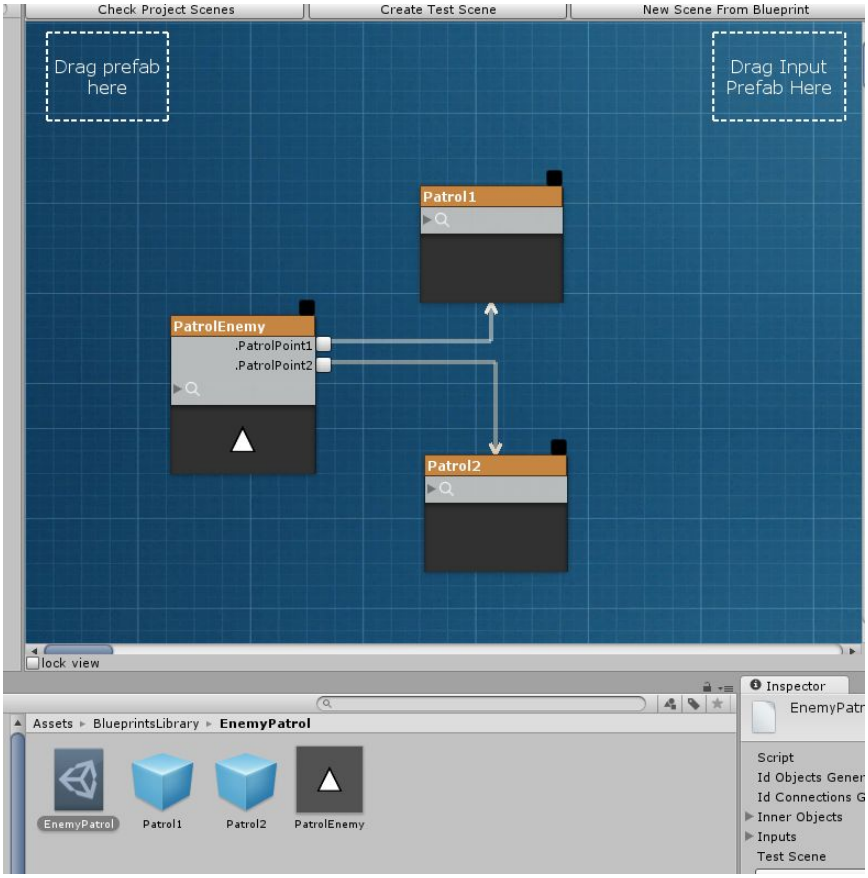
As we want more than 1 enemy let's create a Blueprint to clone more of them.

Selecting the 3 new object that compose our enemy we are now able create a pattern to instantiate multiple times. So select then
Right click > Create Blueprint



to

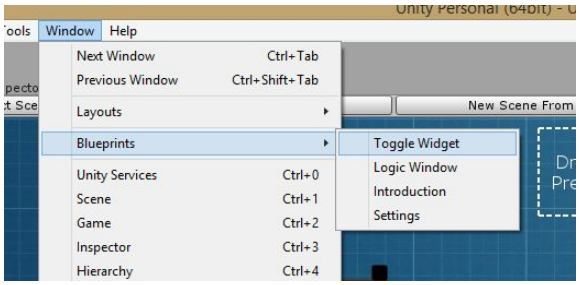
This is what a basic Blueprint looks like.
Its very similar to the scene graph we have because it is the same pattern. With this concept we can now instantiate multiple enemies on the scene.



Let's use the blueprint to add some enemies to our level. For that we need our scene widget.

Scene Widget

The scene widget is a helper we added to the SceneView to help create scenes. With it one can instantiate blueprints and navigate through the objects created with ease.



By going **Window > Blueprints > Toggle Widget** we can toggle its appearance on the scene view.

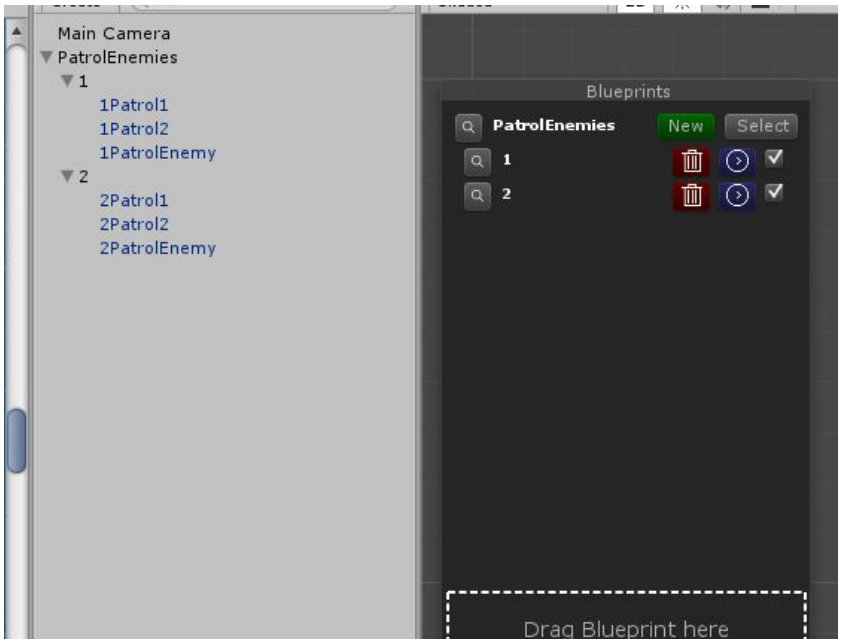
In order to create enemies we drag the new blueprint to the area bottom of it. After that a new type of object is added, we can see hierarchy by the name *PatrolEnemies* and it is our enemy

The point of having an instantiator object is to allow specific associated with the object it clones. By allowing multiple instantiators in a scene we can, for example, divide the enemies categories or have them point to different objects in a scene (like describe later).

in the
it in the
spawner.

settings

into
we are going



Our widget now has a new entry where we can create a new enemy by clicking on ***"New"***.
We can also select the instantiator with ***"Select"*** and we can show all the enemies we already instantiated by revealing them with the *"magnify glass"* button.

We can select each enemy as well or delete it by clicking on the corresponding button. Also we can disable/enable the whole pattern by pressing the toggle button.

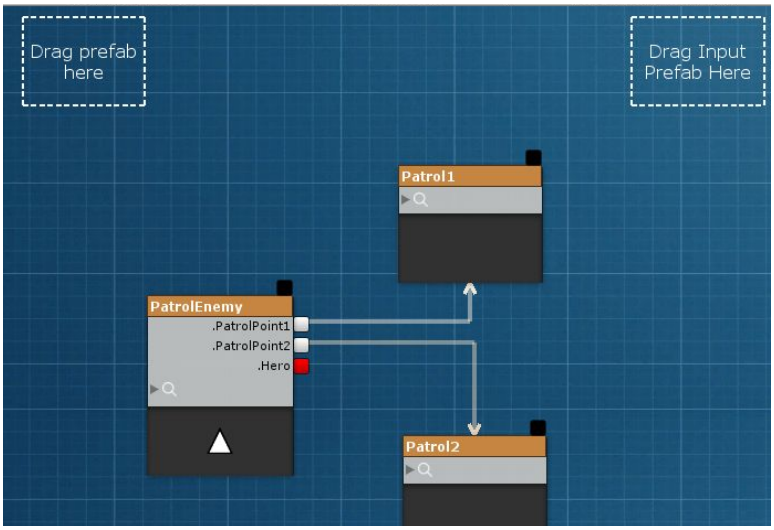
External Blueprint Dependencies

Having enemies walk around the level is fun but it would be good to have them do something. Let's add the ability to reload the level if they see the player.
We first add a new Object called Player to the scene and add a new behaviour named *HeroController*.
Now we add the new Dependency in our enemies to the player.

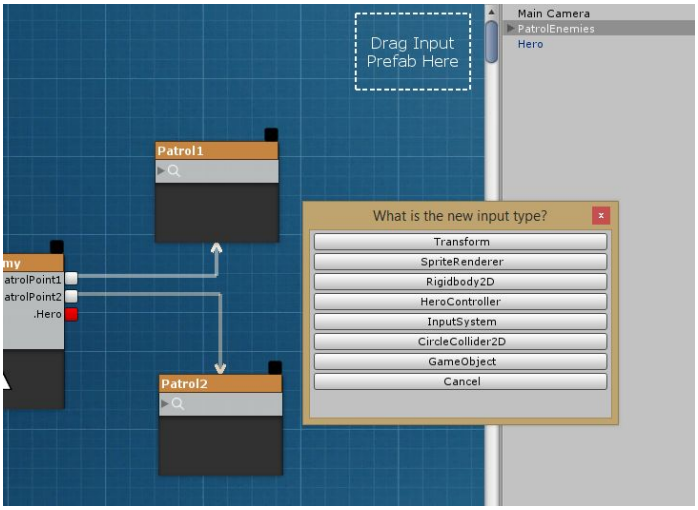
```
public class PatrolEnemy : MonoBehaviour {

      public GameObject PatrolPoint1;
      public GameObject PatrolPoint2;
      public HeroController Hero;
...
}
```

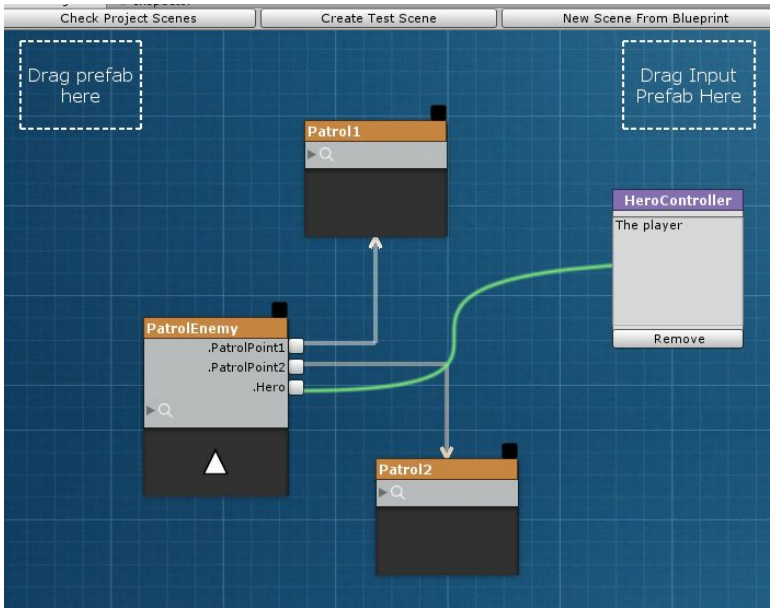
The new dependency “Hero” appears in the blueprint as missing (red color). To set it we need to introduce a new concept to blueprints called ***External Dependency***.



An External Dependency is whenever we want the same objects in all the clones to point to the same object in the scene. In this case we want all enemies to have hero point to our Hero.

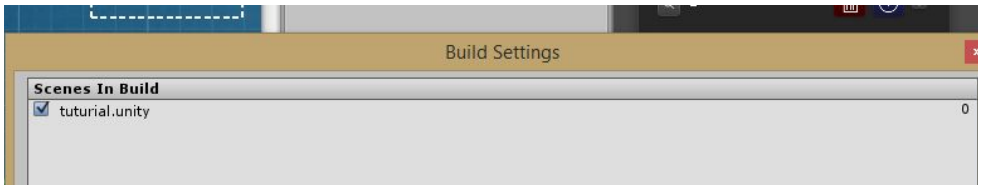


To add an external dependency we drag any component or object to the right green area on the top titled *“Drag Input Prefab here”. When we drag our Hero object a popup will appear asking what type of dependency we want to add, chose HeroController.*

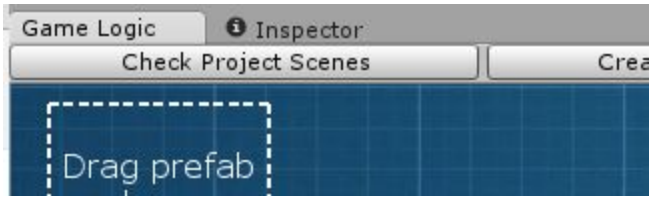


Now connect our hero dependency to the new formed external dependency node (represented by the color purple).
With this we are saying that whenever we instantiate a blueprint we need give it an object in the scene of the type HeroController.

Now that we have changed our blueprint we need to update the scenes it is being used in. To do that we need to add the current scene to the project settings.**Only scenes in the project build are updated when we changed a blueprint.** (for now)

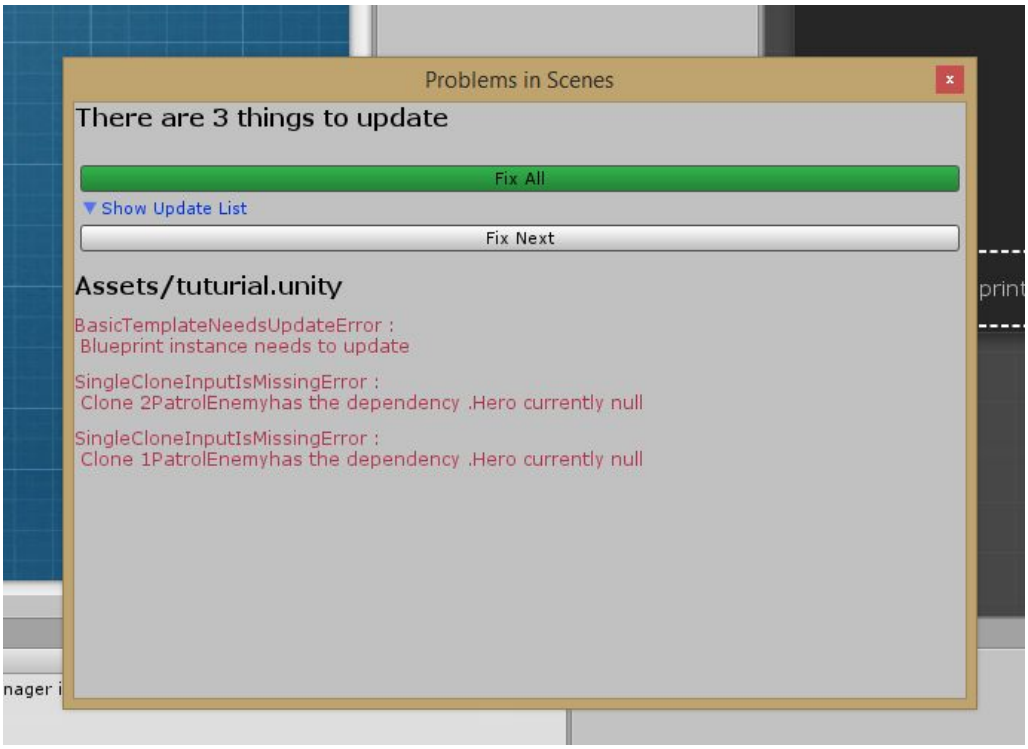


add the scene to the build settings



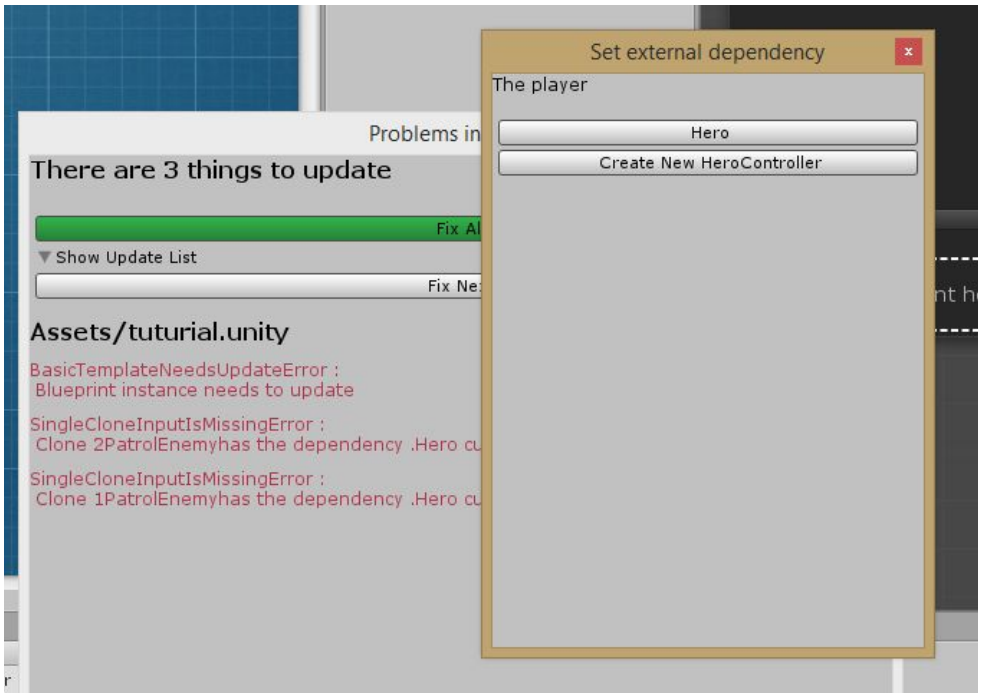
Now that we added the scene we can update it by clicking on the top button ***"Check Project Scenes".***

With this we are going through all the scenes and where we find an instantiator of that blueprint we check what needs to be updated and show it on a popup.



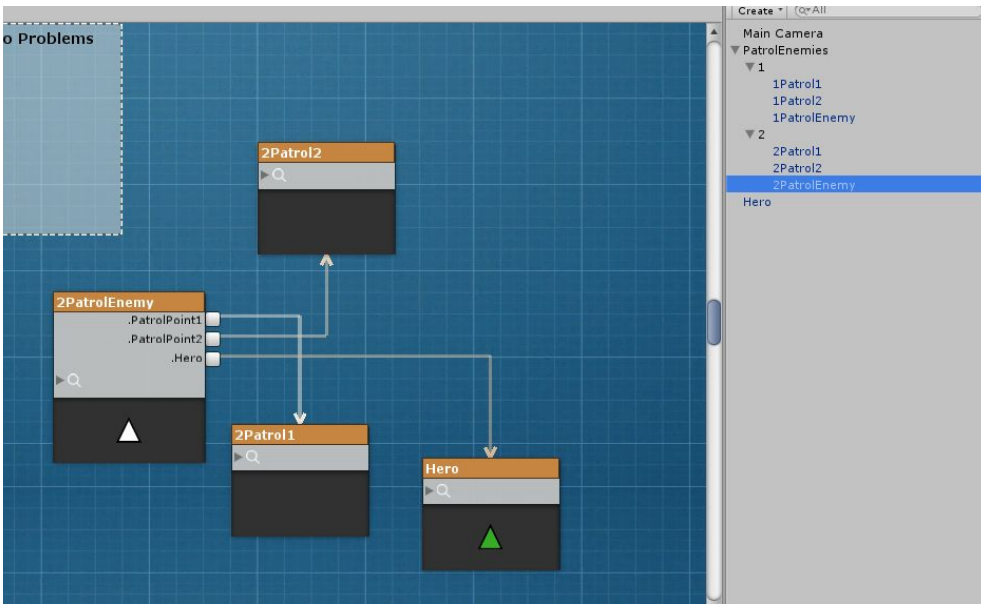
We can go each update with *Fix Next* or all at once with *Fix All*

Click on **Fix All**..
Now, because we added a new external dependency to the blueprint, our instantiator will be asking for an object of the type HeroController.
A popup will appear with the possible objects in scene that can fit into the possibilities, chose our Hero object by clicking on the first option.



Note that the external dependency description (“The player”) appears is in the title to help

After clicking fixing all the changes we can see all enemies created earlier are now pointing to our hero.



Scenes from Blueprints

Now that we have the basis of a level we can add some walls and play it and fun. Nevertheless wouldn't it be better to have more than one level?

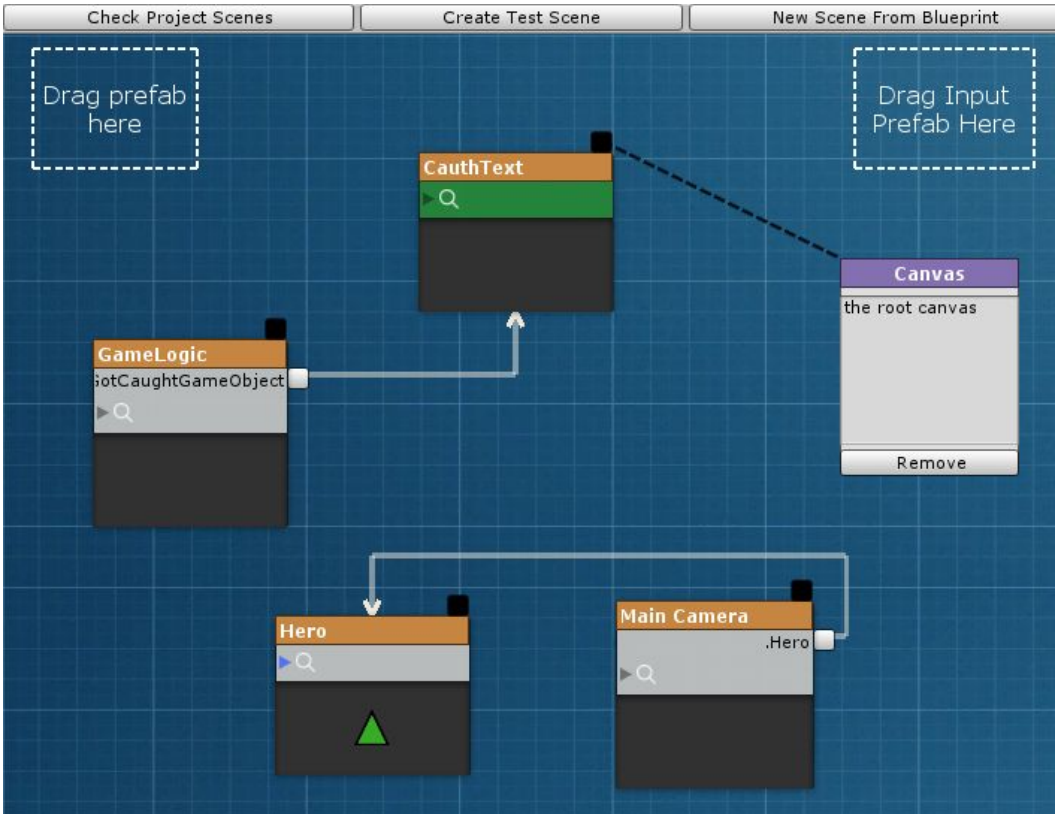
Now to do this what we need to do is define what we need to exist in all levels. We already know our enemies depend on a hero being there, but we are going to add one more thing. Let's add a GameLogic object to control what happens when we get caught, showing a UI text telling the player that he has been caught and for about 3s. So we need a dependency on this new Text object.

```
public class GameLogic : MonoBehaviour {
      public Text YouGotCaughtText;
…
}
```

Also we want to make the main Camera standard across levels so we always have the same view. Adding a Dependency to the player to make the camera follow the player as well.

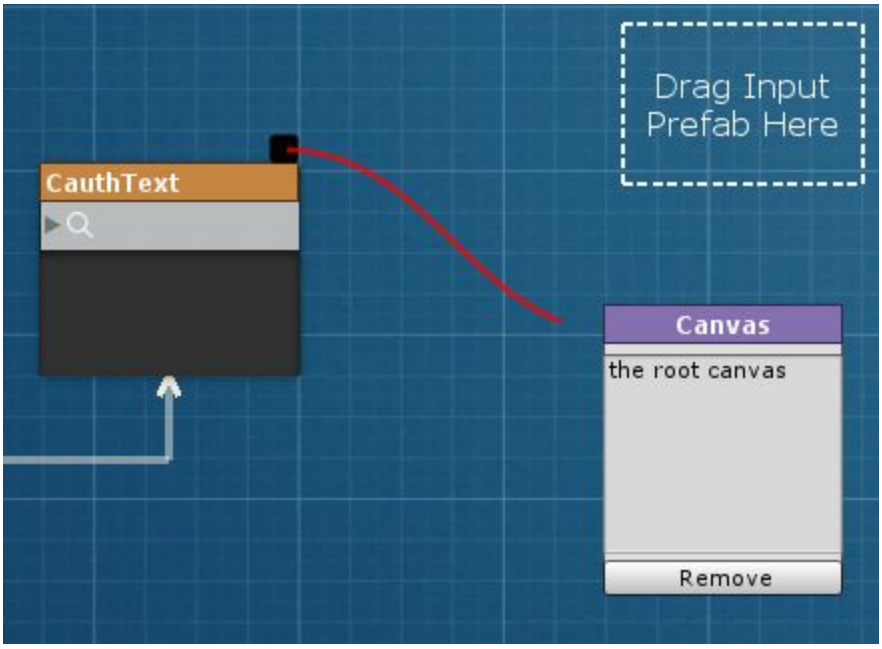
```
public class CameraController : MonoBehaviour {
      public HeroController Hero;
…
}
```

Let's create a blueprint representing this. We need to add a Hero, a MainCamera, am UI CaughtText and a GameLogic to control the everything.



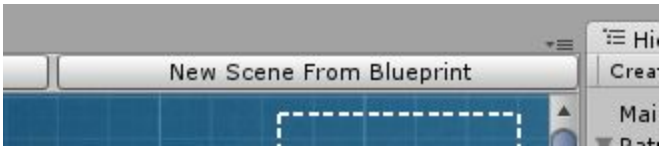
After adding the objects to our scene, creating the new blueprint it should look like this.

In order to have the UI text unity demands that it must be a child of a canvas object.
To represent this we introduce a new feature, setting **blueprints objects hierarchy**.
An object's parent is shown by the dotted line going from the CaughtText to the external Canvas dependency. To change an object's hierarchy click on the black square and connect it to either another object or an external dependency.



Now that we have our blueprint it is time to have it create some levels for us.

Click on the "***New scene from Blueprint"*** button on the top to create a new scene. It will create a new scene with the exact pattern described by the blueprint.



The new scene should have the objects shown here. Now every time we change our level blueprint all the scenes instantiated by it can be updated automatically as well.