

QuickBaseGameManager

The *QuickBaseGameManager* is basically the Main script of any VR application. It offers a base skeleton of the application that you can extend on a subclass. There are four main steps:

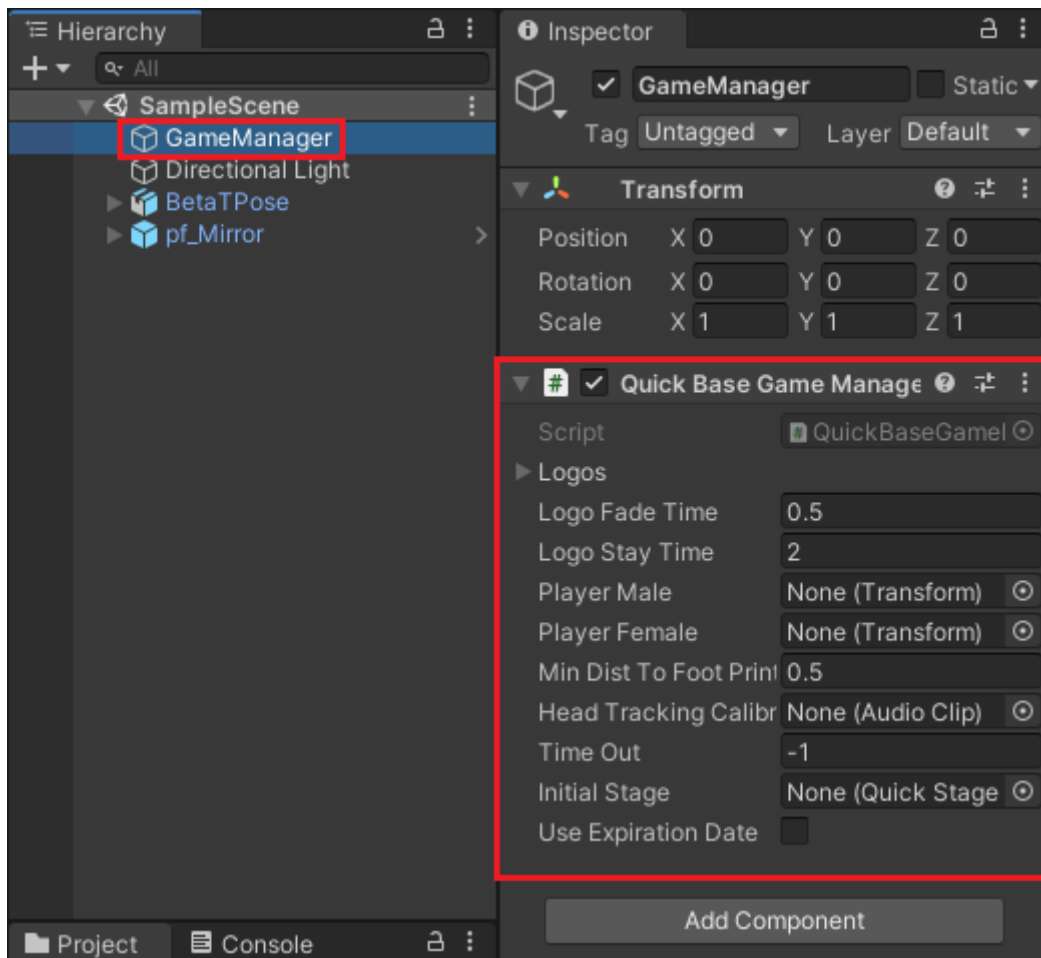
AvatarConfiguration: In this step we configure the virtual avatar. So for example, we can add via script, any component needed by the virtual avatar at the beginning. In order to do so, there are provided two functions that can be overridden in a subclass. The *AwakePlayer* is called on the *Awake* of the *Monobehaviour*, whereas the *StartPlayer* is called on the *Start*.

UserCalibration: Next the user is calibrated. In this step we make sure that the user has the HMD correctly adjusted by showing some text. If the text appears blurry, the user must adjust the HMD until it can read it clearly. Then we require the user to look forward in order to take a reference of the forward direction of the user in the real world.

Run: Here goes the main logic of the application. Usually we pass the control to a *QuickStage*, which we are going to introduce next.

Finish: Finally in this step we set is executed when the game is finished. Here we want to set any action we want to do before the application is closed, for example, saving some results to a file.

So let's try it out. Create a new *GameObject* on your scene and name it *GameManager*. Add the component *QuickBaseGameManager*. Press play, put on your HMD and follow the instructions appearing on the screen. Once the calibration process is finished, you will appear on the scene, but this time you are already calibrated.



QuickStages

The *QuickStage* approach allows us to subdivide the whole logic of the application, which can be arbitrarily complex, into different independent stages. There is a useful set of common stages already implemented, so we can program the logic of the application by simply configuring and concatenating those stages.

Let's see an example to clarify concepts. Imagine that we want to spawn a cube once the application starts, just after the calibration process finishes. Add a new *GameObject* and make it child of *GameManager*. Name it *SpawnCube*.

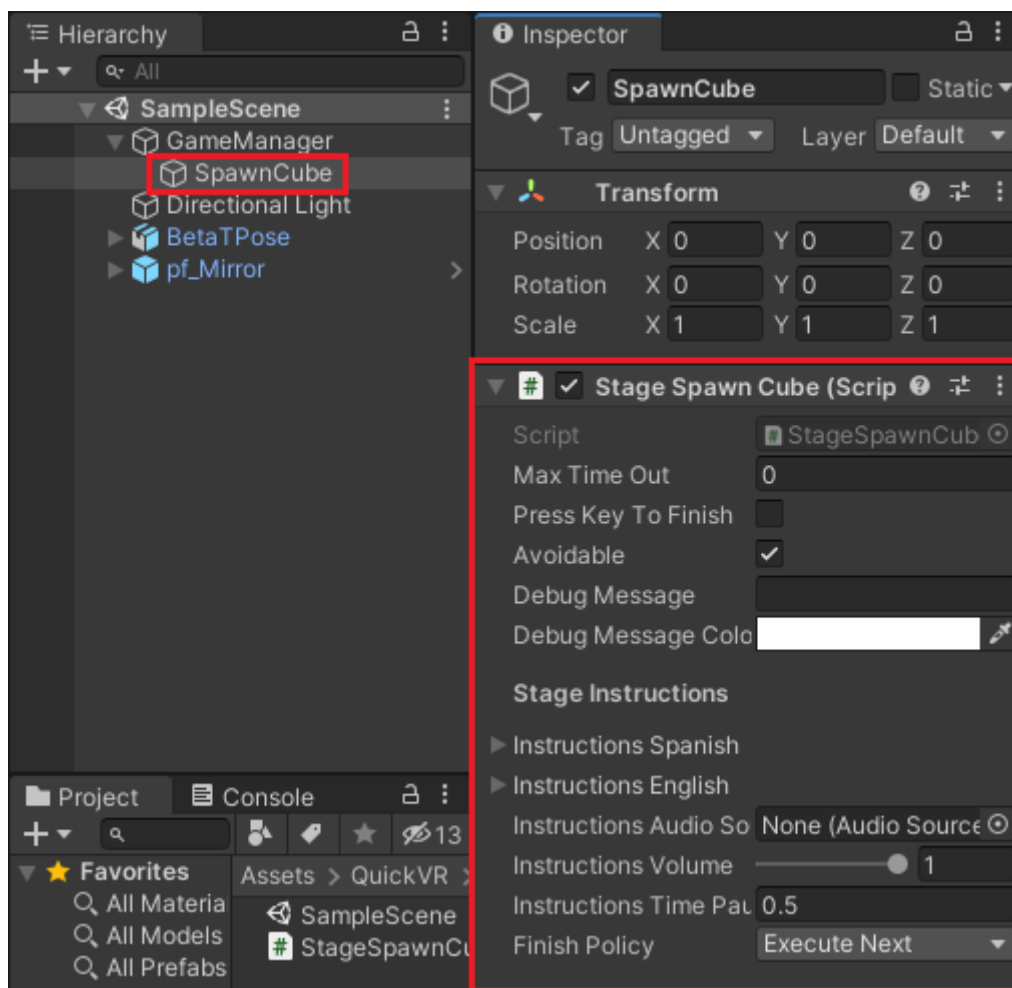
Now add a new script component and name it *StageSpawnCube*. This will be a subclass of *QuickStageBase*.

```

1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4
5  using QuickVR;
6
7  public class StageSpawnCube : QuickStageBase
8  {
9
10
11
12
13

```

At this point, your hierarchy should look like this:



Now let's implement the logic of this stage. We want to spawn a cube when this stage starts. So override the function *Init* of *QuickStageBase* as follows.

```

1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4
5  using QuickVR;
6
7  public class StageSpawnCube : QuickStageBase
8  {
9
10     public override void Init()
11     {
12         base.Init();
13
14         GameObject cube = GameObject.CreatePrimitive(PrimitiveType.Cube);
15         cube.transform.parent = transform;
16         cube.transform.localScale = Vector3.one * 0.25f;
17     }
18 }
19
20

```

Press play and you will see that a cube appears after the fade in, once the calibration process is finished.

Now suppose that you want to repeat this stage 5 times. Of course you could do a *for loop* in order to spawn 5 cubes, but let's do it using stages, just to show the power of what can you do with this.

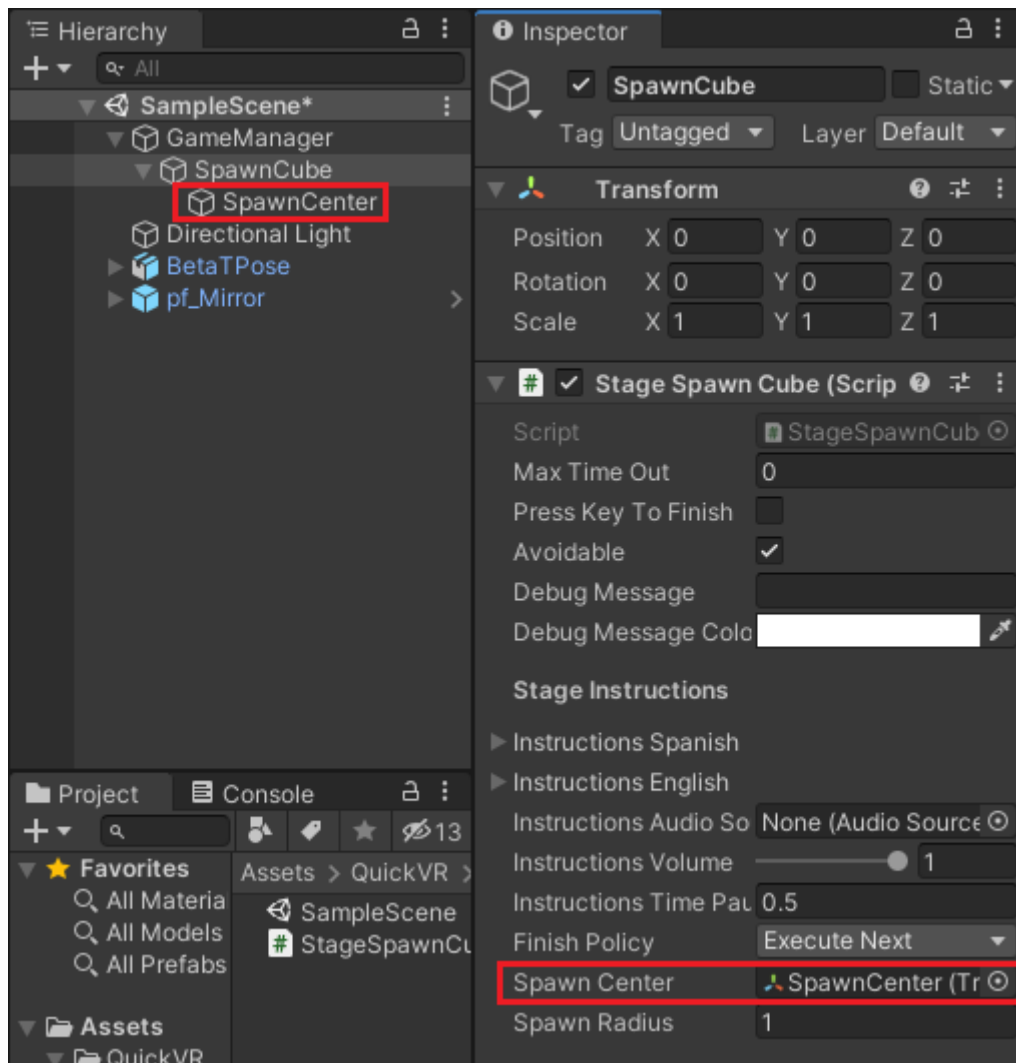
First of all, create two new public variables, one called *_spawnCenter* which will be a *Transform*, and another one called *_spawnRadius* of *float* type.

```

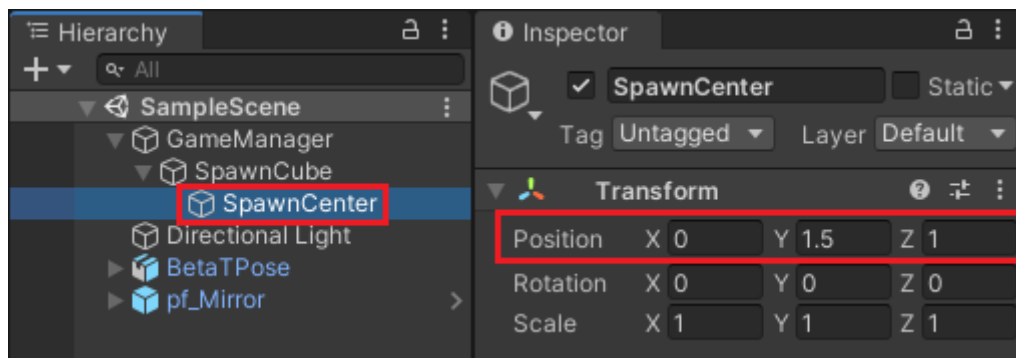
1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4
5  using QuickVR;
6
7  public class StageSpawnCube : QuickStageBase
8  {
9
10     public Transform _spawnCenter = null;
11     public float _spawnRadius = 1.0f;
12
13     public override void Init()...
14 }
15
16
17
18
19
20
21
22
23

```

Back on the scene hierarchy, create a new child *GameObject* of *SpawnCube*. Name it *SpawnCenter* and drag it to the new *Spawn Center* field that has appeared in the component *StageSpawnCube*.



Move the *SpawnCenter* object to (0, 1.5, 1).



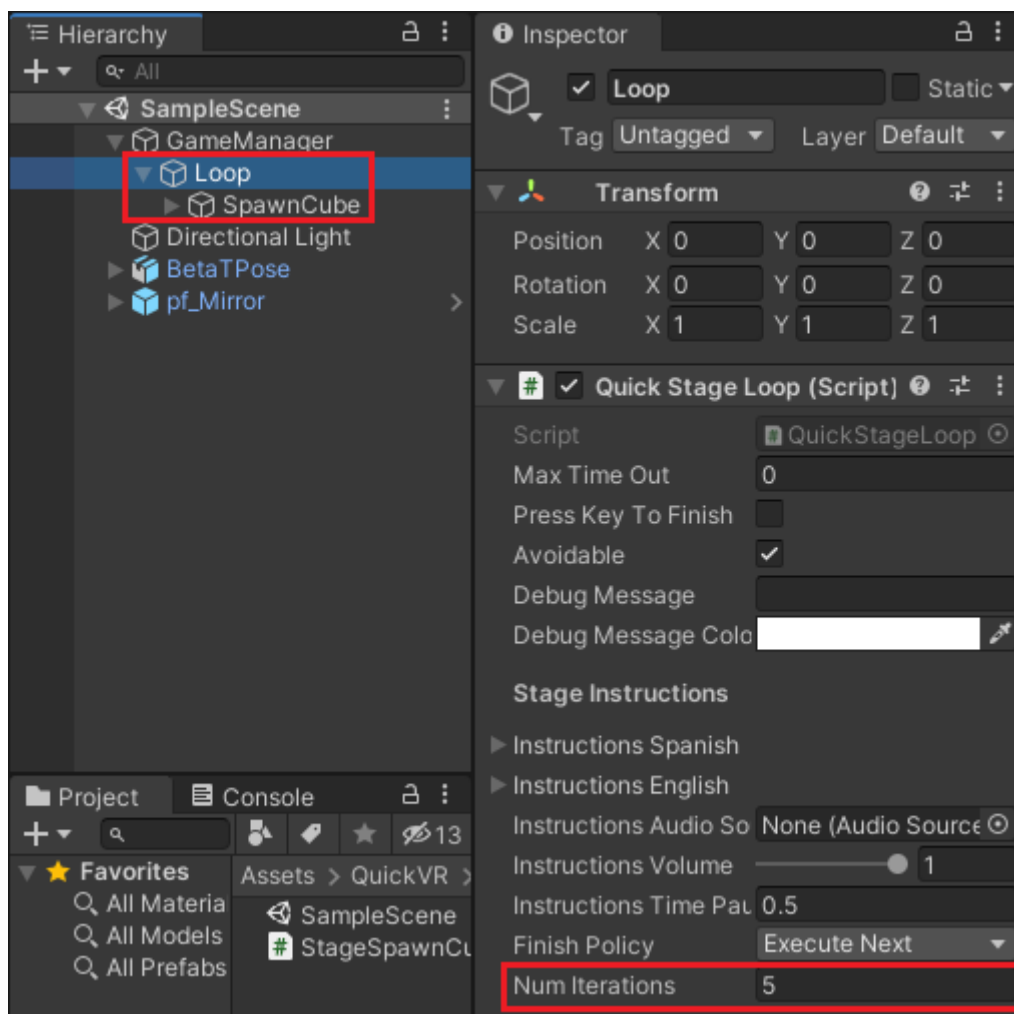
Now modify the script, so the cube spawned appears in a random position inside a sphere with center at *_spawnCenter* and radius *_spawnRadius*.

```

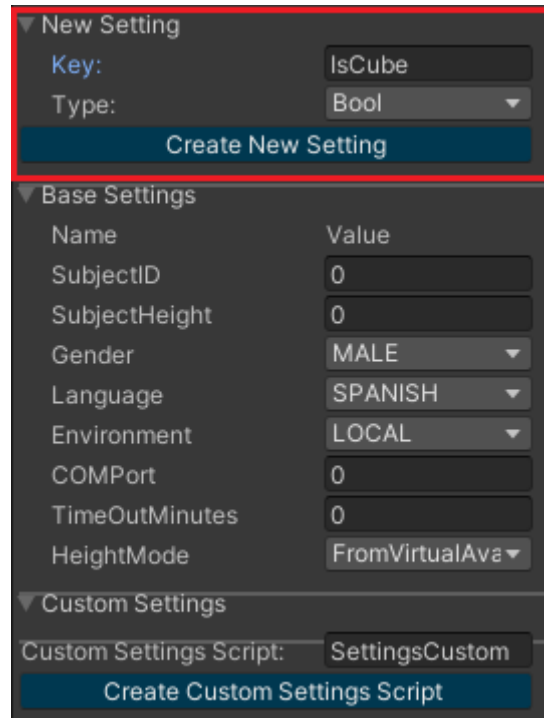
1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4
5  using QuickVR;
6
7  public class StageSpawnCube : QuickStageBase
8  {
9
10     public Transform _spawnCenter = null;
11     public float _spawnRadius = 1.0f;
12
13     public override void Init()
14     {
15         base.Init();
16
17         GameObject cube = GameObject.CreatePrimitive(PrimitiveType.Cube);
18         cube.transform.parent = transform;
19         cube.transform.localScale = Vector3.one * 0.25f;
20
21         cube.transform.position = _spawnCenter.position + (Random.insideUnitSphere * _spawnRadius);
22     }
23 }
24
25

```

Now on the scene hierarchy, create a new child object of *GameManager*. Name it *Loop* and add the component *QuickStageLoop* on it. Set the *Num Iterations* field to 5. Finally, move the object *SpawnCube* to be child of *Loop*. Press play and you will see that 5 cubes will be created.



So we have learned how to make use of *QuickStages* to create a *loop* block. Now we will see that it is also possible to create an *if else* block using *QuickStages*. First of all, open the *PlayerPrefs* window (on the top menu, select *QuickVR > PlayerPrefs*). Create a new setting with *Key* as *IsCube* and *Type* of *Bool*.



The screenshot shows the 'PlayerPrefs' window with a dark theme. The 'New Setting' section is highlighted with a red border. It contains two input fields: 'Key:' with the value 'IsCube' and 'Type:' with a dropdown menu set to 'Bool'. Below these fields is a blue button labeled 'Create New Setting'. The 'Base Settings' section is visible below, showing a list of settings with their names and values. The 'Custom Settings' section is also visible, showing a 'Custom Settings Script' field set to 'SettingsCustom' and a blue button labeled 'Create Custom Settings Script'.

| Name | Value |
|----------------|----------------|
| SubjectID | 0 |
| SubjectHeight | 0 |
| Gender | MALE |
| Language | SPANISH |
| Environment | LOCAL |
| COMPort | 0 |
| TimeOutMinutes | 0 |
| HeightMode | FromVirtualAva |

If it is done correctly, the newly created setting should appear under *Custom Settings*. Now we need to create the settings script. On the *Custom Settings Script* field, set *SettingsTestVR* and click on *Create Custom Settings Script*.

▼ New Setting

Key:

Type: String ▼

Create New Setting

▼ Base Settings

| Name | Value |
|----------------|------------------|
| SubjectID | 0 |
| SubjectHeight | 0 |
| Gender | MALE ▼ |
| Language | SPANISH ▼ |
| Environment | LOCAL ▼ |
| COMPort | 0 |
| TimeOutMinutes | 0 |
| HeightMode | FromVirtualAva ▼ |

▼ Custom Settings

| Name | Value |
|--------|----------------------------|
| IsCube | <input type="checkbox"/> - |

Custom Settings Script: SettingsTestVR

Create Custom Settings Script

Now we can use the newly created setting in our scripts. Create a new *C# Script* and name it *StageCheckIsCube*. Open this new script and make it a subclass of *QuickStageCondition*. In this case, it is mandatory to offer an implementation of the abstract member *Condition*. So do it as follows:

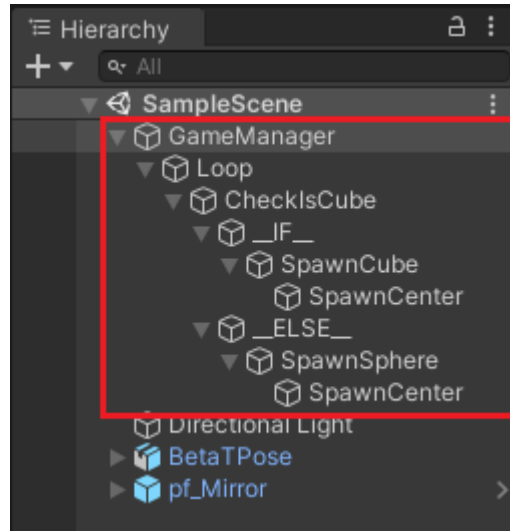
```

1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4  ...
5  using QuickVR;
6
7  public class StageCheckIsCube : QuickStageCondition
8  {
9
10     protected override bool Condition()
11     {
12         return SettingsTestVR.GetIsCube();
13     }
14 }
15
16

```

On the scene hierarchy, create a new child of *Loop* and Name it *CheckIsCube*. Add the previously created component *StageCheckIsCube*. You will see that two childs are automatically created, named *__IF__* and *__ELSE__*. As you may guess, on the *__IF__* branch we will set all the stages to be executed if the condition defined at *StageCheckIsCube* is true. On the contrary, the stages on the *__ELSE__* branch will be executed.

So move the object *SpawnCube* into the `__IF__` branch. Now we will create a new stage to have something to do in the `__ELSE__` case. Following the same process that has been described before, create a new *QuickStage* that spawns a sphere instead of a cube. Add this component in a new child of the `__ELSE__` object and name it *SpawnSphere*. The final hierarchy structure should look like this:



Finally press play to test it. Now we have spawned spheres instead of cubes. This is because the setting *IsCube* is set to false. Open the *PlayerPrefs* window and change the value of *IsCube* to test the other branch.