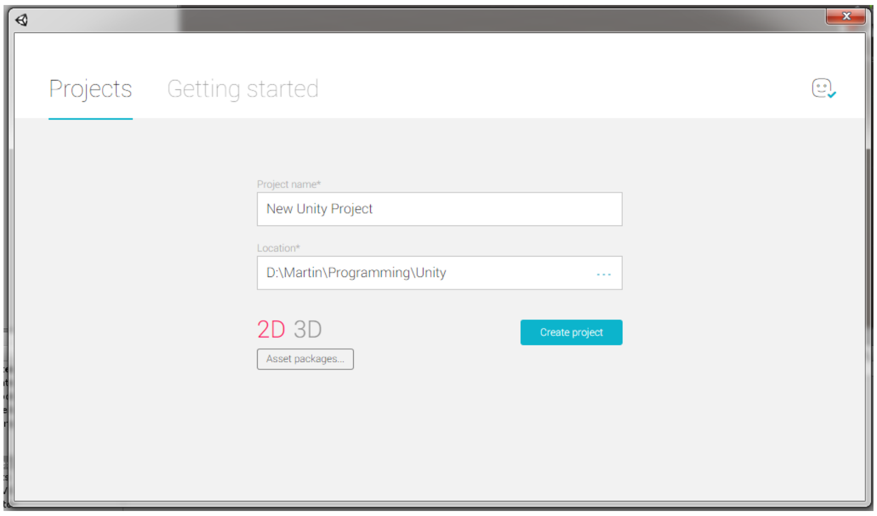
Unity First Project

Based on video tutorial from Unity3D:

https://unity3d.com/learn/tutorials/projects/roll-ball-tutorial

First, we have to create the project itself (***File -> New Project***)



Project name*: New Unity Project
Location*: D:\Martin\Programming\Unity

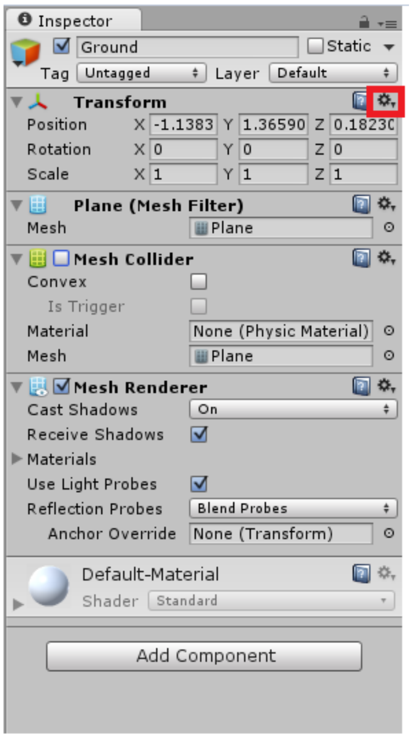
Once we have created project, Unity will create empty scene for us. This scene should be saved at first, before we start to adding some stuff into it. Go to ***File -> Save Scene*** and save scene inside Assets directory. It is a good habbit to save scene into the seperate directory, lets called it „Scenes“.

Inside Assets directory, create also directories: Materials, Scripts, Prefabs. We will use them later.

Create ground

Our scene is currently empty. Well, not really empty, there is a camera and a directional light, but no objects that can be seen if we run our project. Most of the time, if creating 3D scenes, there should be a ground. Even if ground would not be later visible, it is a good practice to put a ground into the scene. It prevents objects with physics to fall into the infinity and its easier to debug the scene with solid ground.

Go to ***GameObject -> 3D Object -> Plane***. This will create simple plane in your scene ***Hierarchy***. Select your plane there and in Inspector, you will see its properties. Change its name to „Ground“. Since this is a ground plane, reset its coordinates to put the center of the plane in point [0,0,0]. This can be done in



Ground ☑ | Static ☐
Tag: Untagged | Layer: Default

Transform
Position: X -1.1383 Y 1.36590 Z 0.18230
Rotation: X 0 Y 0 Z 0
Scale: X 1 Y 1 Z 1

Plane (Mesh Filter)
Mesh: Plane

☐ **Mesh Collider**
Convex: ☐
Is Trigger: ☐
Material: None (Physic Material)
Mesh: Plane

☑ **Mesh Renderer**
Cast Shadows: On
Receive Shadows: ☑
Materials
Use Light Probes: ☑
Reflection Probes: Blend Probes
Anchor Override: None (Transform)

Default-Material
Shader: Standard

Transform section by clicking „the gear icon“ and selecting „Reset“ (this will also reset any scale or rotation changes).

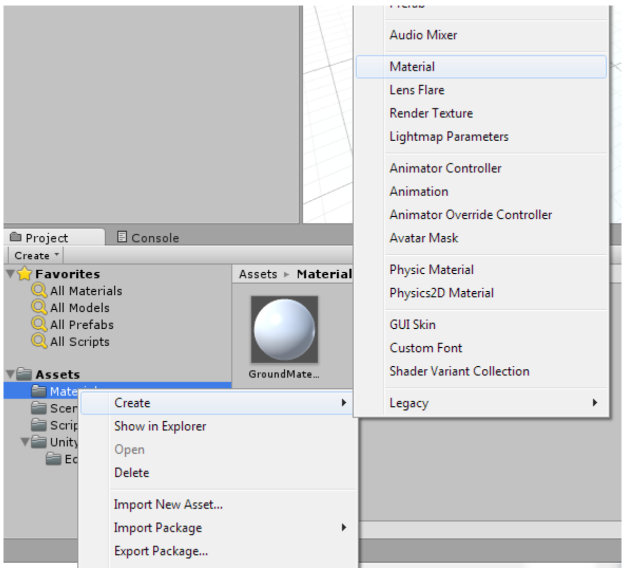
To enlarge the area, scale it a bit in X a Z axis. Y axis will do nothing, because plane has no height (only if you use -1, it will flip the plane).

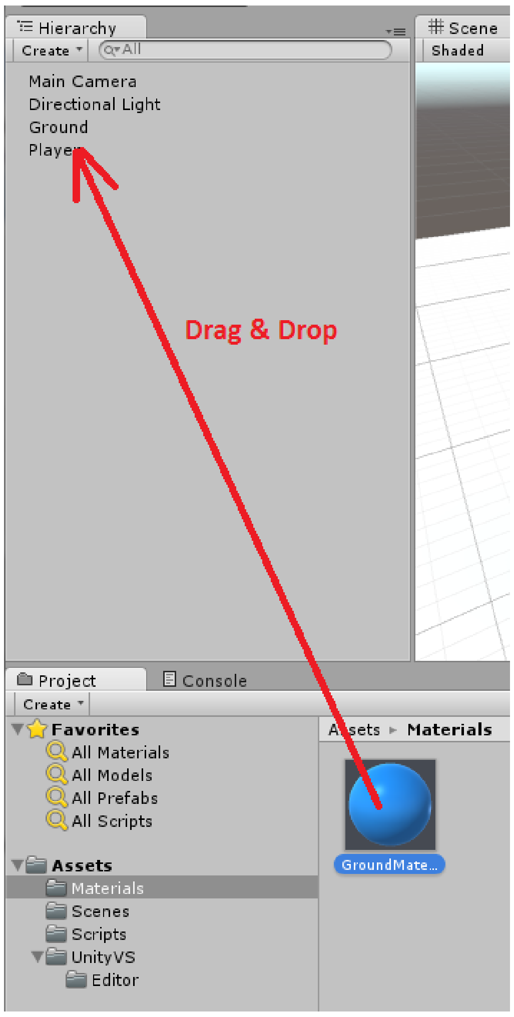
As the last step, enable Mesh Collider. Without it, all object with gravity will fall through your plane.

Adding Material

Our ground has a white color. It is not very nice, so we will add some color to it. Inside the directory Materials, create new material. Right click on the Material folder in the ***Project*** inspector.

After material has been created, select it and in the Inspector, you will see its properties. To assign material to the ground, Drag & Drop it to the Ground object.

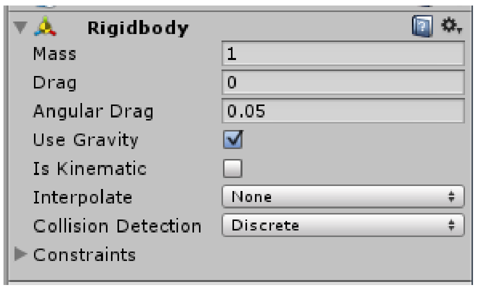




The Player

In the next step, we will add a player to the scene. Start with something simple and add a sphere the same way, the ground plane was added. Reset its position to be [0,0,0].

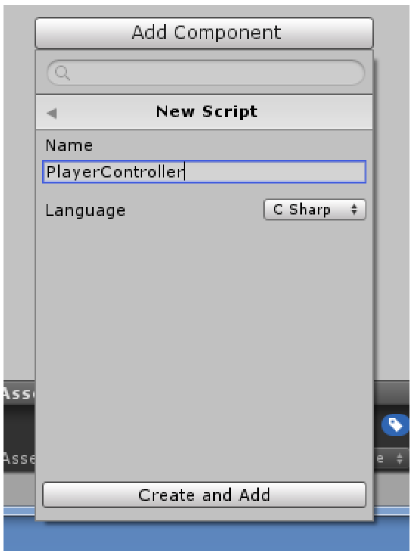
To control the sphere, in this tutorial, the physics based controlling is used. For this, Physics component needs to be added to the sphere. So, select sphere in the scene Hierarchy and go to ***Component -> Physics -> RigidBody***. New RigidBody component will appear in your object Inspector.



Rigidbody

Mass	1
Drag	0
Angular Drag	0.05
Use Gravity	✓
Is Kinematic	
Interpolate	None
Collision Detection	Discrete
Constraints	

Player has to be controlled by the, well, player. For this, we will use scripts. The easiest way to create and add a script to the scene object is this: Select scene object in the Hierarchy and in Inspector, use Add component button and add New Script. We will use C#.



Add Component

New Script

Name	PlayerController
Language	C Sharp

Create and Add

Our script is created inside Assets directory. To better maintain the project, move the script inside Scripts directory.

Double click on your script will open it in predefined editor (default is Mono). There are some precreated methods for us, but we can delete them.

Methods:

Start() – called in first frame, only once

Update() – called once per frame just before rendering

FixedUpdate() – called before physics update (can be more than once per frame ?)

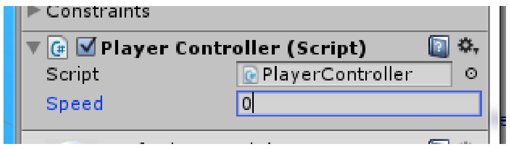
LateUpdate() – run after Update() just before rendering

Lets see sample code for Player sphere controlling.

```
public class PlayerController : MonoBehaviour {

        public float speed;

        private Rigidbody rb;

        void Start()
        {
                this.rb = this.GetComponent<Rigidbody>();
        }

        void FixedUpdate()
        {
                float moveHorizontal = Input.GetAxis("Horizontal");
                float moveVertical = Input.GetAxis("Vertical");

                Vector3 movement = new Vector3(moveHorizontal, 0, moveVertical);

                this.rb.AddForce(movement * speed);

        }
}
```

As you can observer, variable speed is public and has no value (so default 0 will be used). That is not what we want. This variable, because it is marked as public, can now be set from within Unity itself.

In the ***Inspector*** of the sphere (player), in component with script, new editable value appear with name of our variable. Now you can set value directly from Unity. Set some value and experiment yourself.



Now you can test your project, hit the Play button in Unity above your scene.

As you can see from test run, the camera is not set very friendly. This is because camera has been set to observe the scene, not the player.

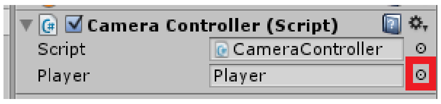
If you want to experiment:

Drag & Drop camera to the player (it became its children). In the camera transform section, move or rotate it a bit until you are satisfied with te result. You can see preview of your current settings inside Scene. Now test your scene. After you have moved the player, WTF just happened. The camera is rolling up and down, even if are not changing its transformation. This is caused by the player. The camera is attached to it and copy its transforms (with its own transform offsets). Player is rolling ball, the camera is rolling as well. Well, this is not the solution, so Drag & Drop the camera back, where it was.

To move camera together wit the player, we have to create its controll script. Create CameraController the same way, as we have created the controll script for the player (select ***Camera -> Add Component -> New Script*** (use C#)).

```
public class CameraController : MonoBehaviour {

        public GameObject player;

        private Vector3 offset;

        // Use this for initialization
        void Start () {
                this.offset = this.transform.position - this.player.transform.position;
        }

        // Update is called once per frame
        void LateUpdate () {
                this.transform.position = this.player.transform.position + this.offset;
        }
}
```

Beware! Dont forget to attach GameObject inside Unity (similar way as we have changed the speed). Click on the marked „circle with dot“ and in window, that appears, select ***Scene -> Player***.



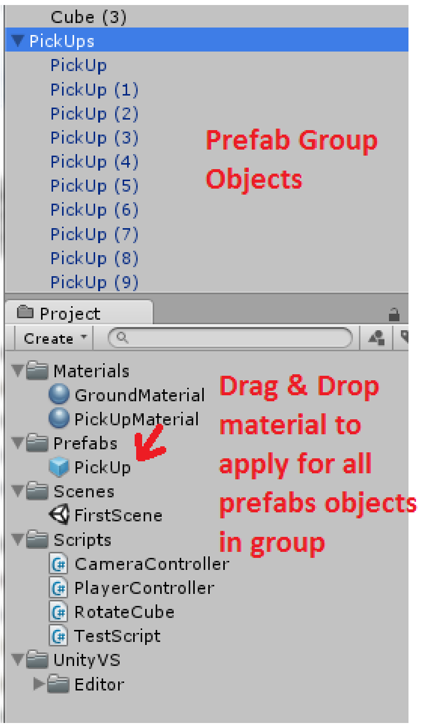
Camera Controller (Script)

Script	CameraController
Player	Player

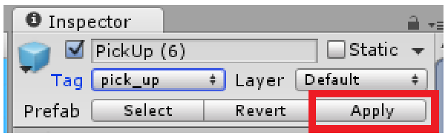
Other scene objects

Now create walls, to prevent player from falling from the area at the ground ends. ***Create GameObject -> Create Empty*** and rename it to Walls. Reset its position. Put a single cube inside and again, reset its position. Now scale it and/or rotate it as you like and move it to the ground end. We need three more „walls“. You can create them the same way, but if you want to spare some work, duplicate the first cube (***Edit -> Duplicate***) and then move and/or change the duplicate wall.

We have now created main game area. Next step is to put some collectables inside. We will use rotating cube. Create new cube in scene ***GameObject -> 3D Object -> Cube*** (reset it and name it PickUp). To rotate it, you will need a script. We will need more of those, with only one pickable object, it wont be much of a game. We can create each cube manually the same way, but it will be too much work. Instead, we will use „Prefabs“ – they contain single object with all its properties and can put them into scene many times, using the same settings and scripts. Just Duplicate the same object within the prefab group. We can also create a material and assign it to all objects within the Prefab Group. See the image.



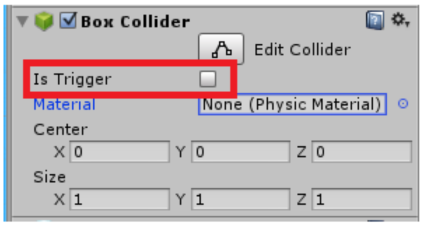
When doing something with Prefab and select one object that is part of the prefab, to update all objects, you have to click „Apply“ button after each change to commit changes to every object in prefab.



Detecting collisions

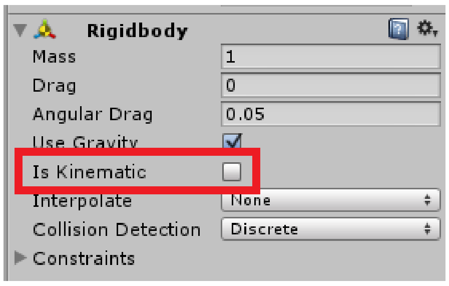
There are two basic types of collision objects – static and dynamic. Static objects are used for non-movable parts of the world and they are optimized for this task (using cache etc.). Dynamic, however, are used for movable elements.

Each object, that we want to be part of the physics, have Collider. Collider response to collision with another object can be double – first type reacts to collision the classic way by pushing objects apart, the second is called Trigger and only detects collision, but do nothing with objects itselves. To enable triggers, check appropriate setting in Unity. *„OnTriggerEnter“* method is called in our C# scripts after collision with trigger object.



Box Collider
Edit Collider
Is Trigger ☐
Material None (Physic Material)
Center X 0 Y 0 Z 0
Size X 1 Y 1 Z 1

If our object is movable, we have to tell it to Unity. Dynamic objects are the objects, that have RigidBody component attached to them. Now, we can change RigidBody properties to something, that suits out needs. For example, we want our objects to stay put in the air. By only disable gravity, they dont fall down, but they will still react to forces. To disable objects forces altogether, set the object as a Kinematic one. Those objects can be directly positioned instead of using forces.

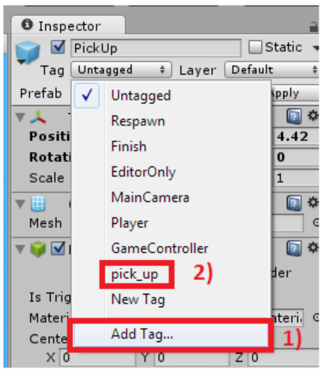


Rigidbody
Mass 1
Drag 0
Angular Drag 0.05
Use Gravity ✓
Is Kinematic ☐
Interpolate None
Collision Detection Discrete
Constraints

To use collider inside our code and test somethign on it, we can use this

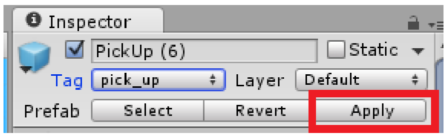
```
void OnTriggerEnter(Collider other)
{
        if (other.gameObject.CompareTag("pick_up"))
        {
                other.gameObject.SetActive(false);
        }
}
```

Now, one important thing. Objects are compared using Tags. Here, we have used tag „pick_up“. This tag has to be set for every object, we want to test with this code. First, create custom tag by „Add Tag...“ and then set this tag to the object.



!!! Remember !!!

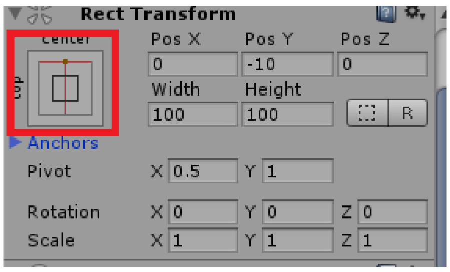
When doing something with Prefab and select one object that is part of the prefab, to update all objects, you have to click „Apply“ button after each change. This goes for the Tag, as well as other setting in colliders



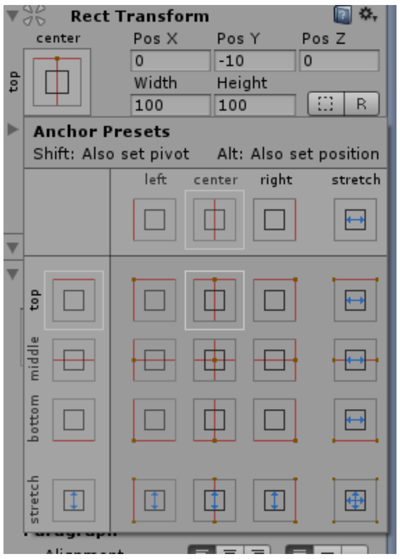
Inspector
PickUp (6) Static
Tag pick_up Layer Default
Prefab Select Revert Apply

GUI elements

To show elements in the GUI, we will use canvas. Add simple Text element via ***GameObject -> UI -> Text***. This element is placed in the scene and inside Hierarchy, its inside Canvas parent. It must stay there! If you wat to change position of the newly created Text, you can use anchors. Click on highlighted element and anchor settings will show.



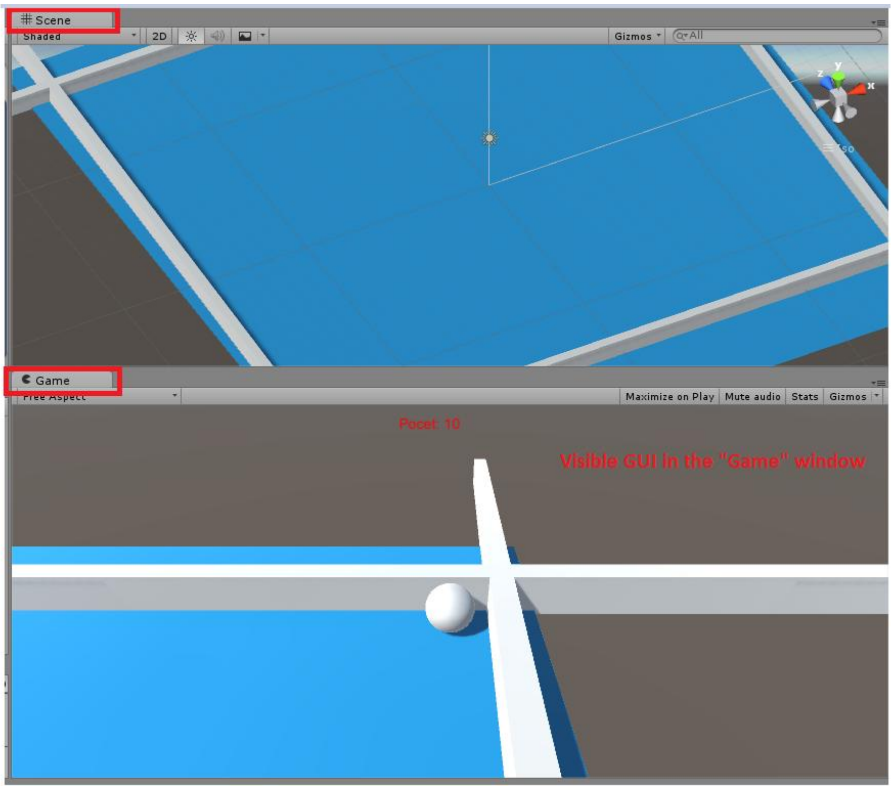
```
Rect Transform
center      Pos X   Pos Y   Pos Z
top         0       -10     0
            Width   Height
            100     100
Anchors
Pivot       X 0.5   Y 1
Rotation    X 0     Y 0     Z 0
Scale       X 1     Y 1     Z 1
```



```
Rect Transform
center      Pos X   Pos Y   Pos Z
top         0       -10     0
            Width   Height
            100     100
```

There you can position your GUI Text element on the screen. If you press Shift and/or Alt, another options of anchors will show in the visible menu.

GUI is not visible in the „Scene“ window, instead it is visible in „Game“ window. This is the windows, that tells you, how the game look in the current setting of the camera and other stuff from „Scene“ window.



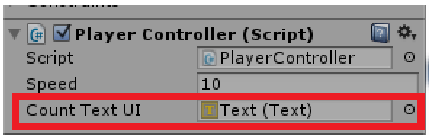
To use GUI elements inside your code, add

```
using UnityEngine.UI;
```

and inside class

```
public Text countTextUI;
```

and connect it from the Unity the same way, as we did before with the GameObject for the Camera.



Player Controller (Script)

Script: PlayerController

Speed: 10

Count Text UI: Text (Text)

Now you can manipulate text inside GUI Text element

```
this.countTextUI.text = "Some text ";
```

This is all for this simple tutorial. You have now created basic Unity app, that uses many aspects, that will be used in your future projects.