# Module User Experience | Gaming

## Onderdeel Unity Practicum 2: Jump

## Auteurs

Paul Bergervoet (Gameplay en teamleiding)

Monique Dewanchand (C#)

Lars Tijsma (Unity)

Anton Visser (Gameplay en testen)



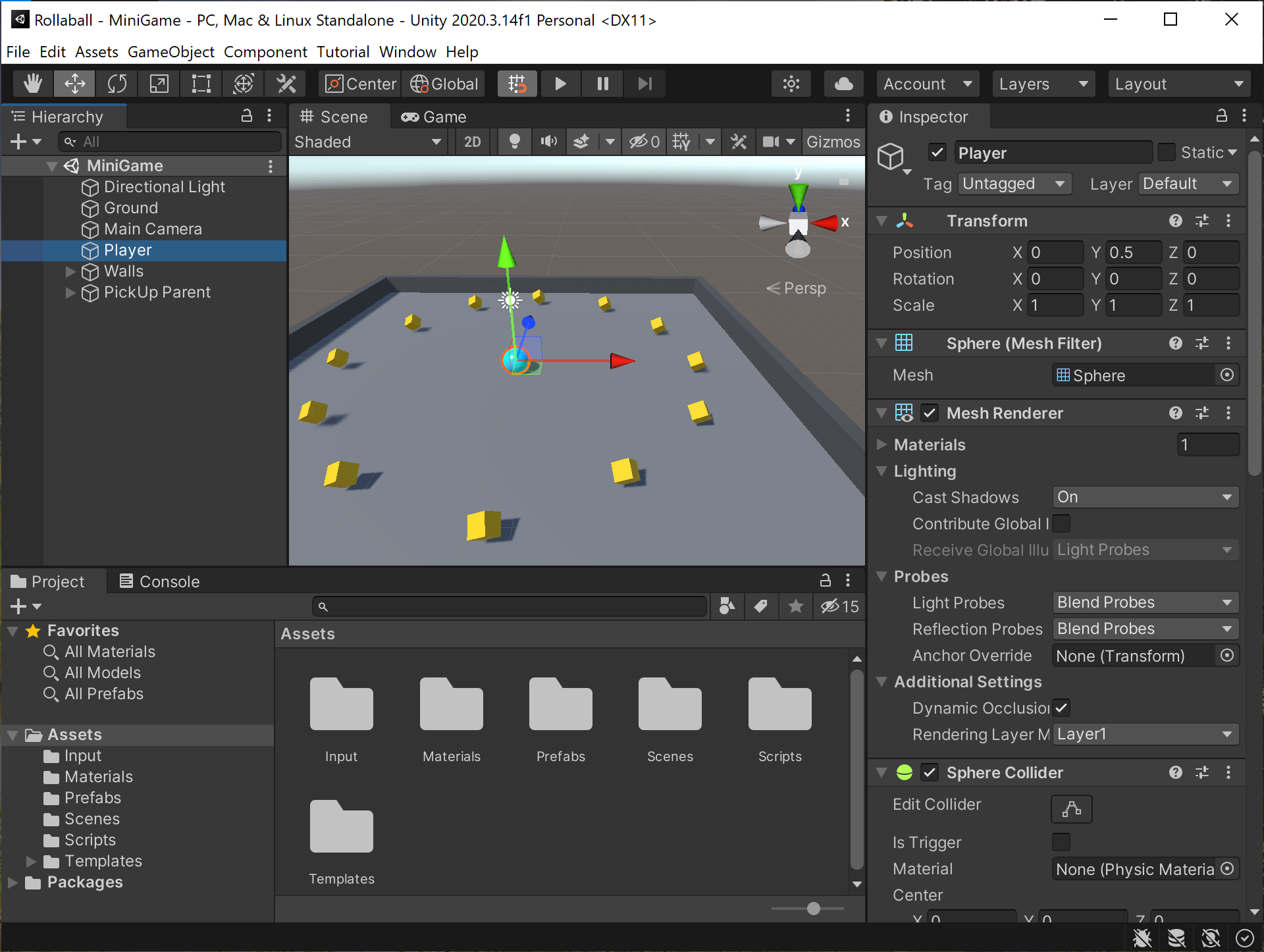
Unity-opdracht bij GD2: Jump

In deze opdracht ga je een “jump mechanic” toevoegen aan het spel. De bal moet kunnen springen als hij op de grond staat.

Bij dit practicum is er een startpunt, een Unity-project (Unity versie 2021) waarin je het practicum kunt maken. Je kunt ook je eigen Unity-project van Roll-a-Ball gebruiken en daarop voortbouwen.

## Start

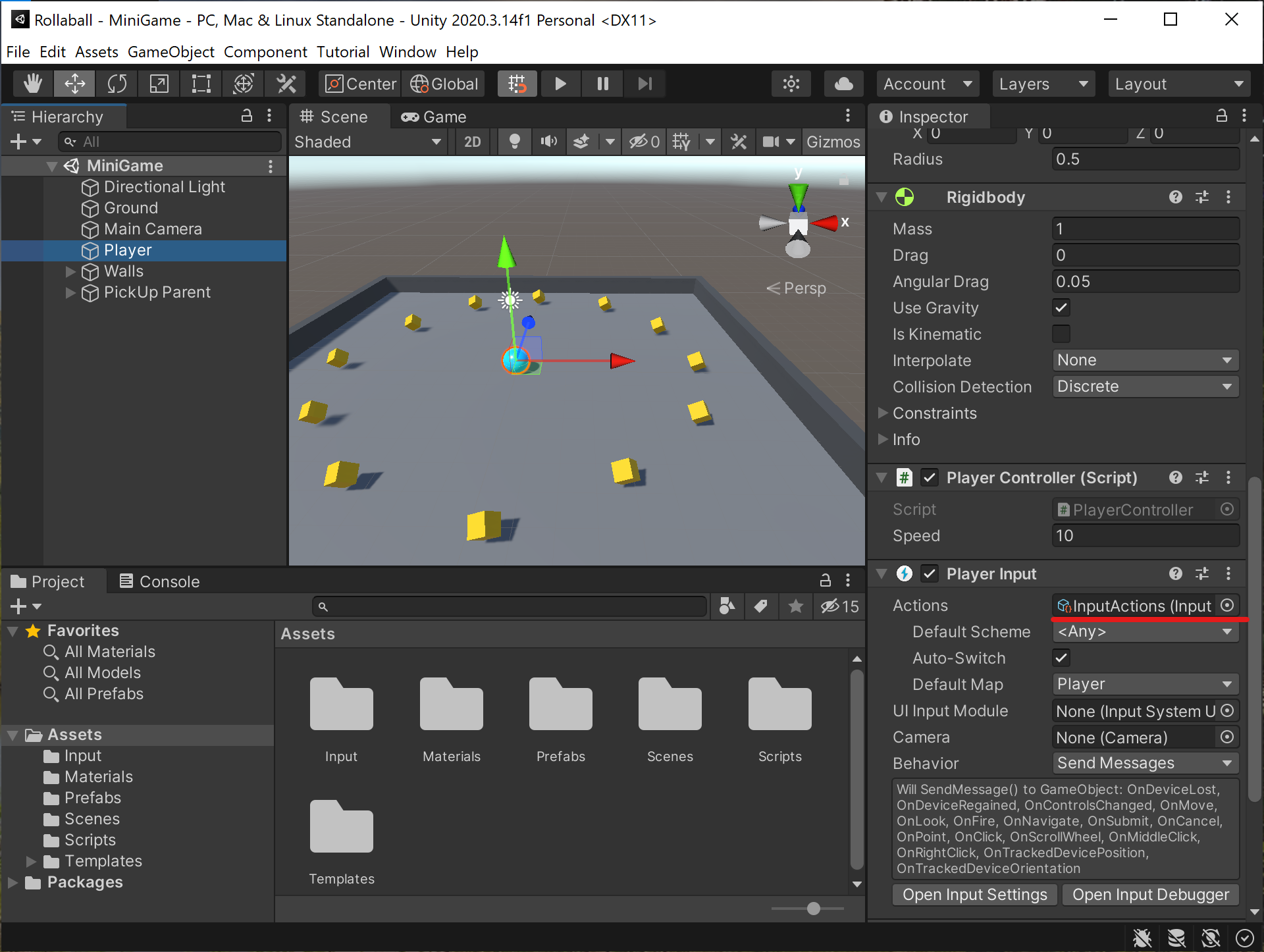
* Open de versie die je eerder hebt gemaakt, of open de versie in de map 02-jump en open de scene MiniGame en selecteer Player.
* Als het goed is ziet de Unity editor er nu ongeveer zo uit:



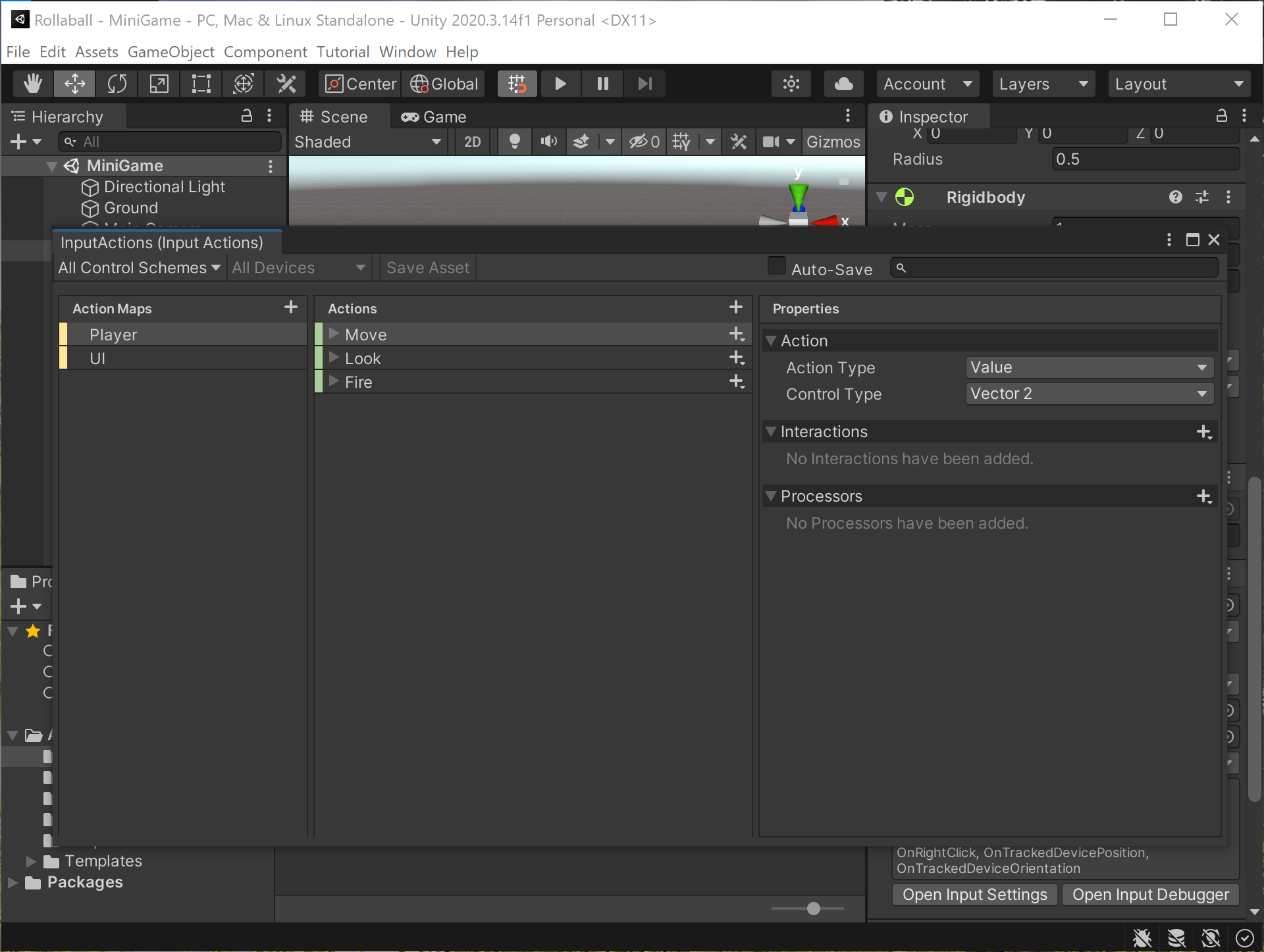
## Jump Action toevoegen

De speler moet de bal kunnen laten springen door op de spatiebalk te drukken. In de tutorial-video’s van Unity heb je gezien hoe je de bal kunt laten bewegen via het “Player Input”- component. We gaan aan dit component de actie Jump toevoegen.

* Open “InputActions” in het “Player Input”-component door erop te dubbelklikken:

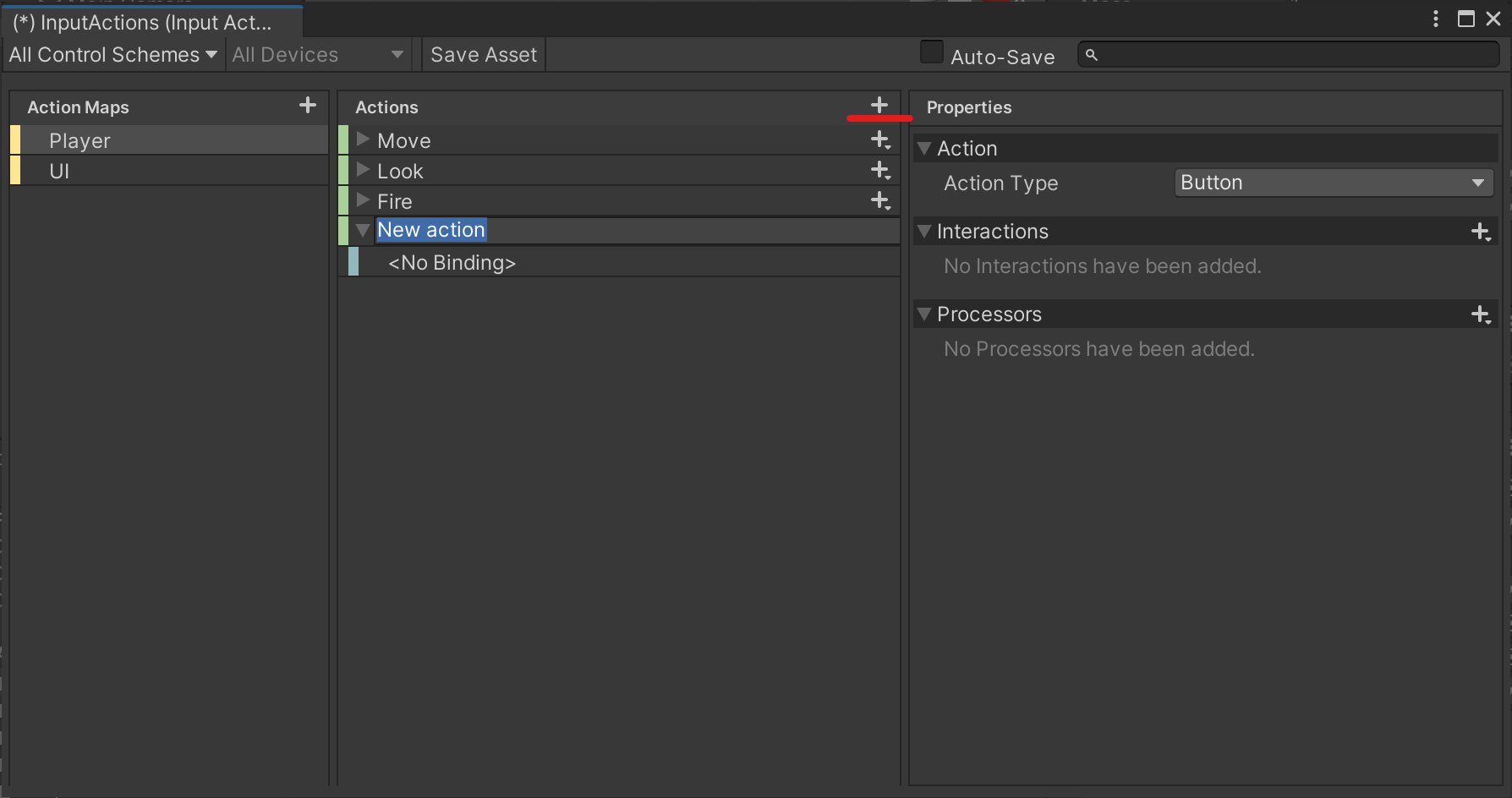


* Als het goed is, zie je dan het onderstaande venster:



De middelste kolom bevat alle Actions die op dit moment beschikbaar zijn. Deze zijn automatisch gegenereerd tijdens de Unity Roll-a-Ball-tutorial. Op dit moment gebruiken we alleen de Move-action. Je ziet dat er nog geen Action voor springen bestaat.

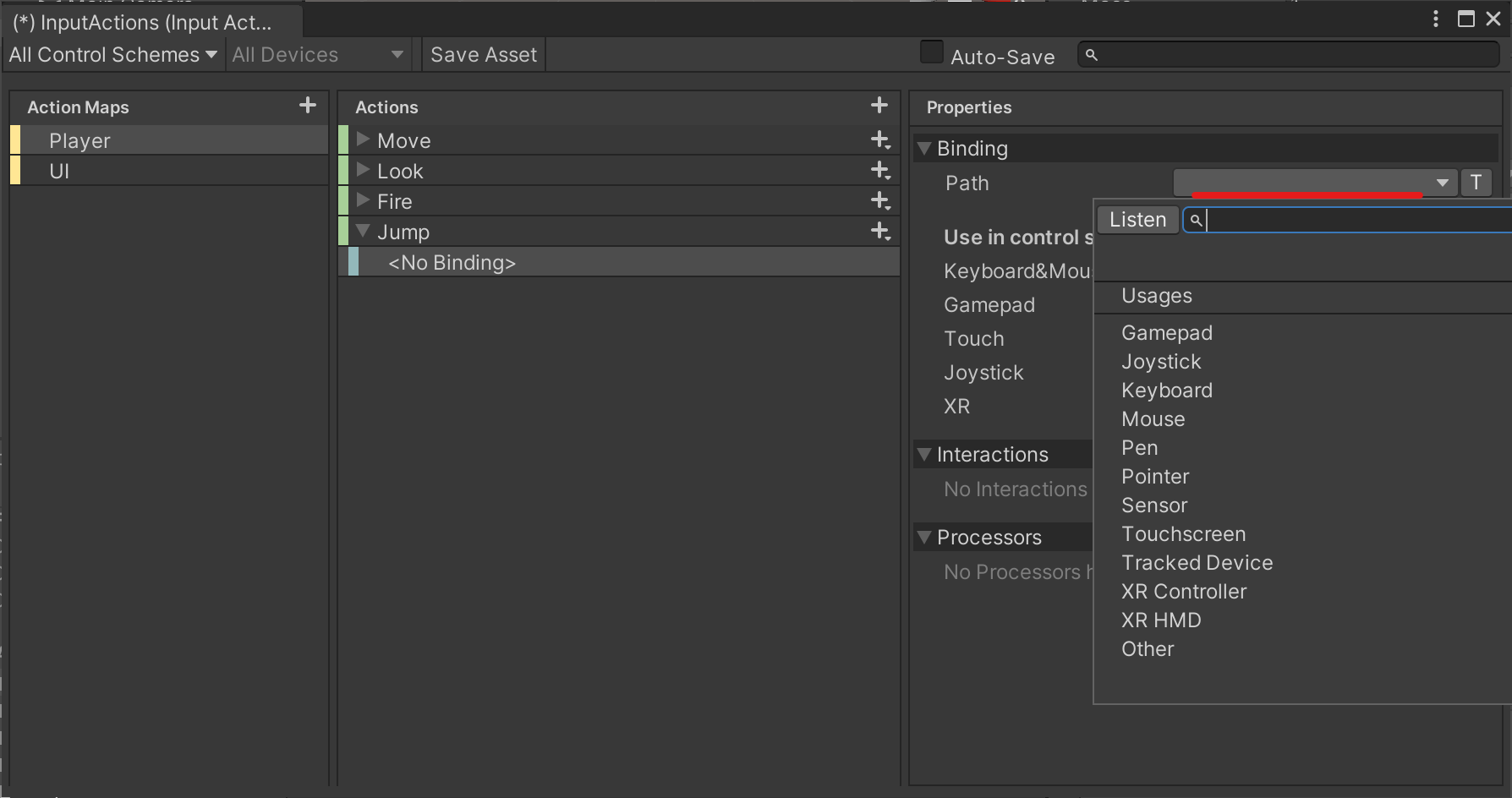
* Voeg een nieuwe Action toe door op de “+” te klikken naast het woord Actions
* Als het goed is verschijnt er een nieuw item in de lijst zoals hieronder te zien is:



* Verander de naam van New action in *Jump*.

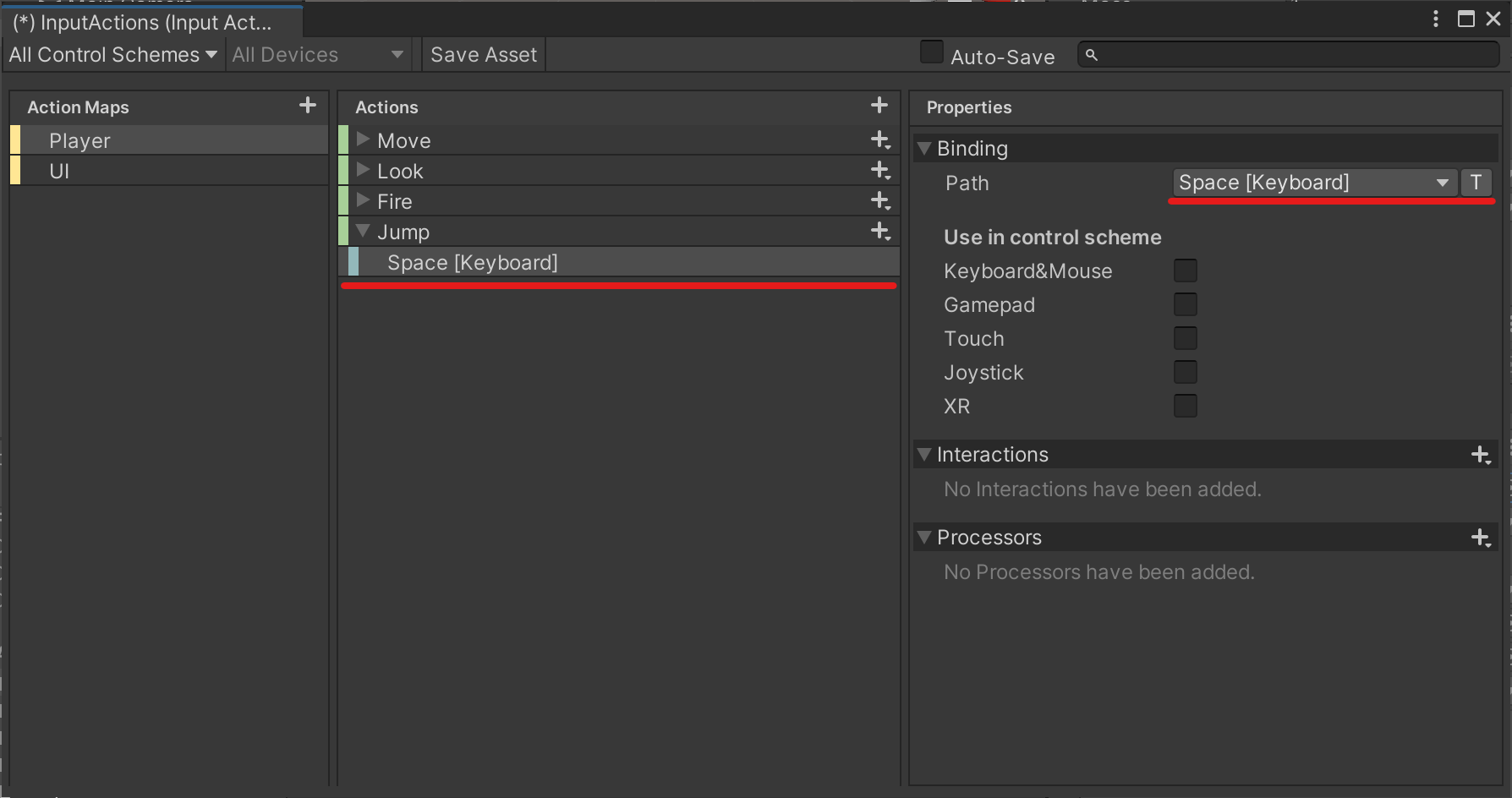
Je hebt nu de actie gedefinieerd, maar nog geen koppeling gemaakt met een druk op een knop. Deze koppeling wordt een binding genoemd en je een binding maken voor het drukken op de spatiebalk.

* Onder *Jump*, selecteer *<No Binding>* en klik op de lege knop naast het woord *Path*:



Je kunt nu de spatiebalk selecteren uit de lijst, of Unity automatisch de spatiebalk voor je vinden door op listen te klikken.

* Klik op de knop *Listen*, druk op de spatiebalk en selecteer Space [Keyboard]. Als het goed is ziet het venster er dan zo uit:



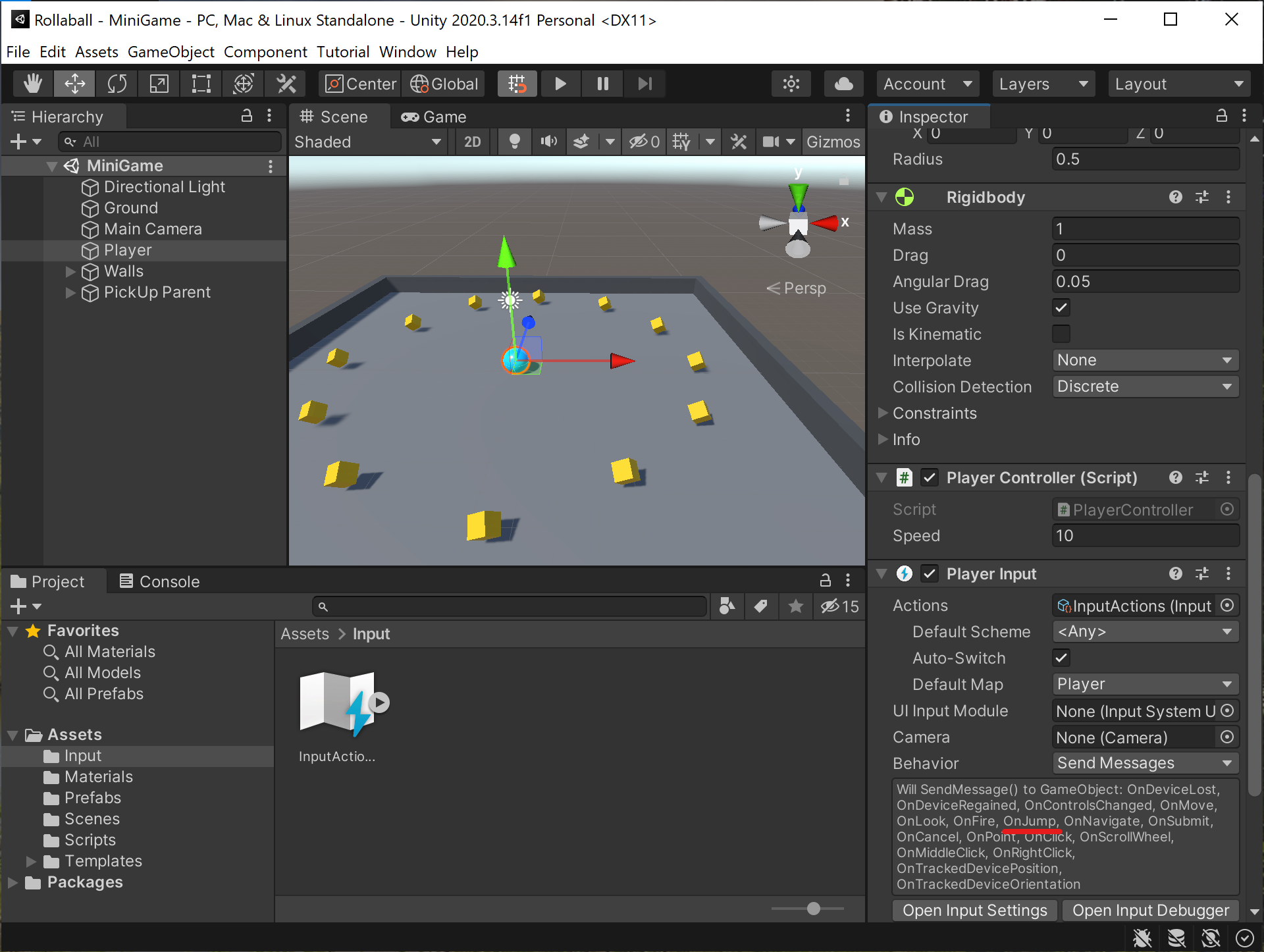
* Sla de wijzigingen op door bovenin op de knop *Save Asset* te klikken en sluit het venster.

**Tip:** Je kunt aan het sterretje, (\*), linksboven naast InputActions zien dat er nog niet-opgeslagen wijzigingen zijn.

Als je zeker weet dat je altijd alle wijzigingen wilt opslaan dan kun je ook een vinkje bij Auto-Save zetten.

**Optie:** Als je zelf een controller hebt, of een ander device waarmee je het spel wilt spelen, dan kun je hiervoor een extra binding maken door op het plusje te klikken naast Jump. Op deze manier kun je eenvoudig voor dezelfde Action verschillende knoppen maken zodat de speler kan kiezen welke manier van besturen hij, of zij wil gebruiken.

* Bekijk het Player Input-component weer in de Unity editor. Als het goed is zie je nu de het woord *OnJump* in de lijst onder Behavior staan:



Je kunt in alle scripts in het Player-object de functie *OnJump* toevoegen en Unity roept dan automatisch deze functie aan als de speler op de spatiebalk drukt.

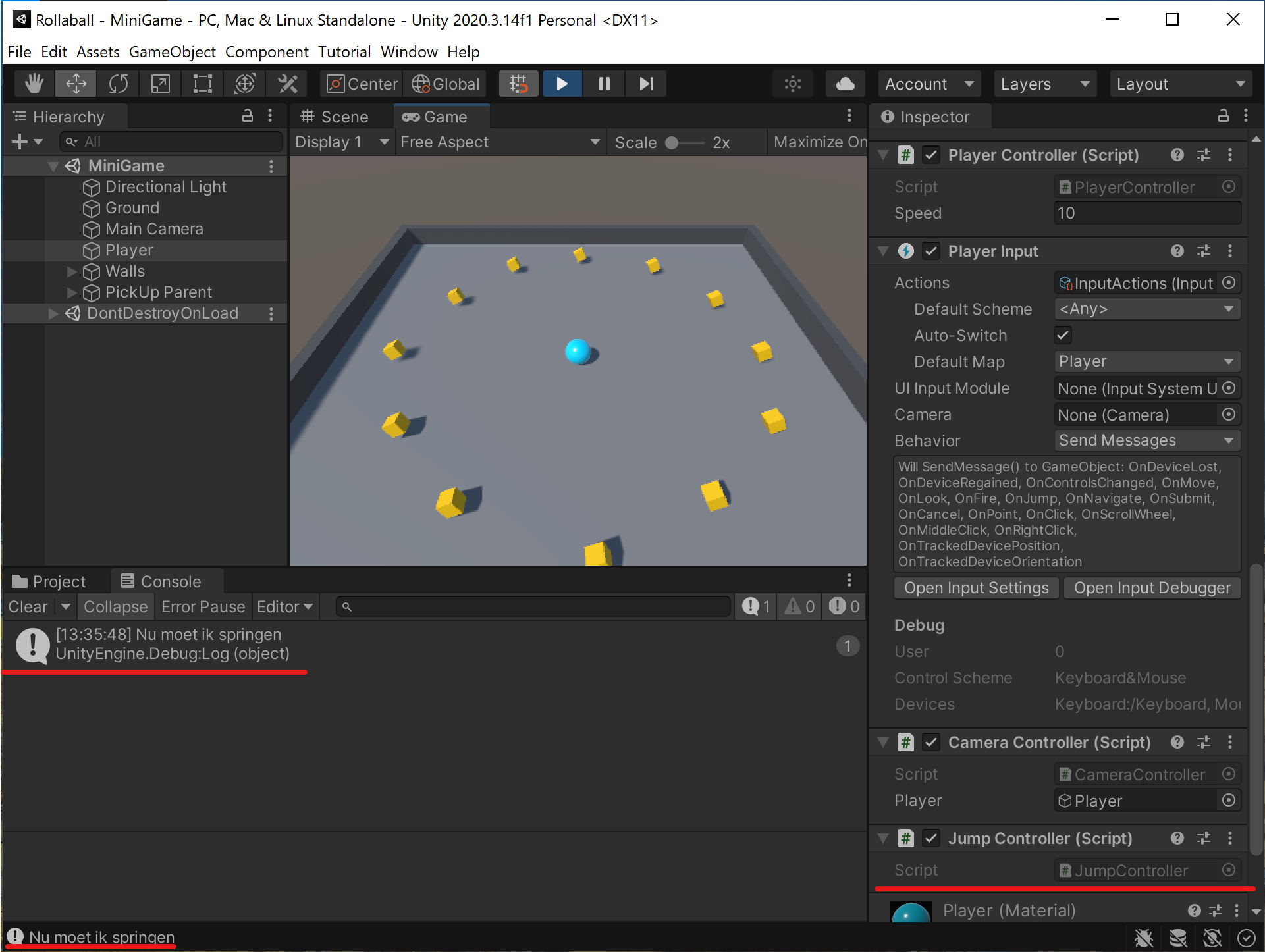
## Input opvangen in een nieuw script

Je kunt in Unity meerdere scripts aan een GameObject toevoegen. Daar maken we gebruik van door het spring-gedrag in een nieuw script te plaatsen.

* Maak een nieuw script in de scripts folder (rechtermuisknop > “Create” > C# Script).
* Noem dit script “JumpController”.
* Open het script en vervang alle standaard code door onderstaande code:

| using System.Collections; using System.Collections.Generic; using UnityEngine;  public class JumpController: MonoBehaviour {   private Rigidbody rb;    void Start()   {  rb = GetComponent<Rigidbody>();  }   void OnJump()   {   Debug.Log("Nu moet ik springen");  } } |
| --- |

* Voeg het script toe aan de het Player game object door het naar dit object te slepen.
* Test het spel. Als het goed is zie je de tekst “Nu moet ik springen” wanneer je op de spatiebalk drukt.



### Vraag

De code die er uiteindelijk voor gaat zorgen dat de bal kan springen staat nu in het nieuwe script JumpController. Deze code had echter ook kunnen worden toegevoegd aan het eerder gemaakte script PlayerController .

#### A)

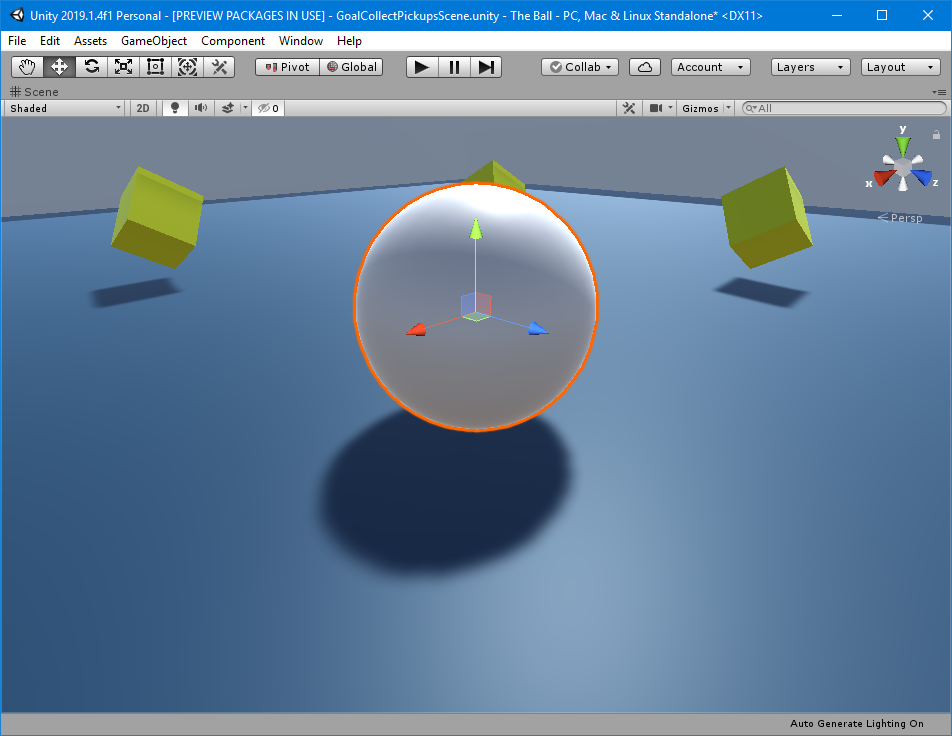
Noem een voordeel van het maken van een apart script voor het realiseren van het springen van de bal..

#### B)

Kun je ook een nadeel verzinnen?

## Springen

Door de bal een kracht omhoog te geven, kun je het springen realiseren. Zoals je eerder hebt gezien, kun je een game object een kracht geven door de methode AddForce aan te roepen op het RigidBody component van dat game object. In de afbeelding hieronder zie je dat de groene pijl de richting omhoog aangeeft.



* Voeg onderstaande regels toe aan de code in OnJump

| void OnJump()   {   Debug.Log("Nu moet ik springen");  Vector3 jumpForce = */\* Vul hier de vector in \*/*  rb.AddForce(jumpForce, ForceMode.Impulse);  } |
| --- |

* Vervang het commentaar in de code door een nieuwe *Vector3* voor de kracht omhoog.
* Test het spel en verander de grootte van kracht totdat je tevreden bent met de kracht waarmee de bal omhoog gaat.

### Vraag

Misschien valt je op dat de tweede parameter ForceMode.Impulse aan AddForce is toegevoegd. Zoek in de documentatie: <https://docs.unity3d.com/ScriptReference/ForceMode.Impulse.html> op in welke situatie je deze parameter gebruikt en leg deze situatie in eigen woorden uit.

## Alleen springen als je op de grond staat

* Test het spel en druk meerdere malen op de “Jump”-knop. Als het goed is, zie je dat de bal blijft springen ook als je niet op de grond staat.

De bal moet alleen kunnen springen als hij daadwerkelijk op de grond staat. We moeten dus een manier hebben om te kunnen detecteren dat er grond onder de bal aanwezig is.

* Voeg de volgende code toe aan de methode Update:

| void OnJump()   {   if (isOpDeGrond())  {  Debug.Log("Nu moet ik springen");  Vector3 jumpForce = */\* Vul hier de vector in \*/*  rb.AddForce(jumpForce, ForceMode.Impulse);  }  }   private bool isOpDeGrond()  {  *//Retourneert true als de bal zich op de grond bevindt*  } |
| --- |

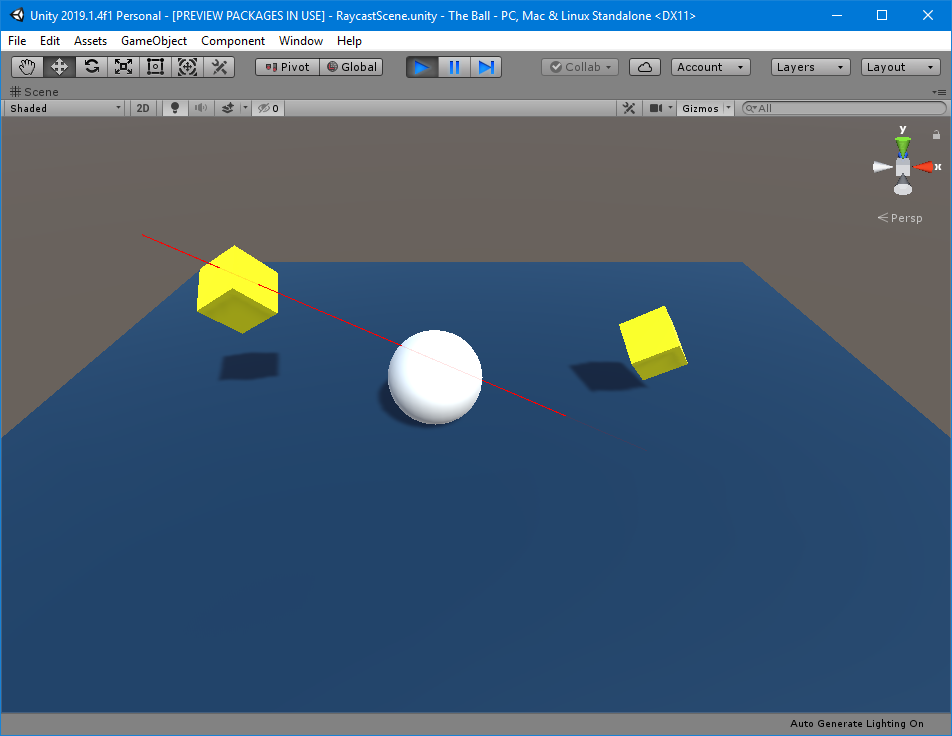
Je ziet dat we een extra methode hebben toegevoegd die true moet retourneren als de bal zich op de grond bevindt. Deze methode roepen we aan in het if-statement dat we ook voor het detecteren van de “Jump”-knop gebruiken.

### Raycast

Een veelgebruikte manier om te bepalen of zich objecten in een bepaalde richting bevinden, is het afschieten van een rechte lijn in die richting. Het afschieten van een lijn wordt een *raycast* genoemd.

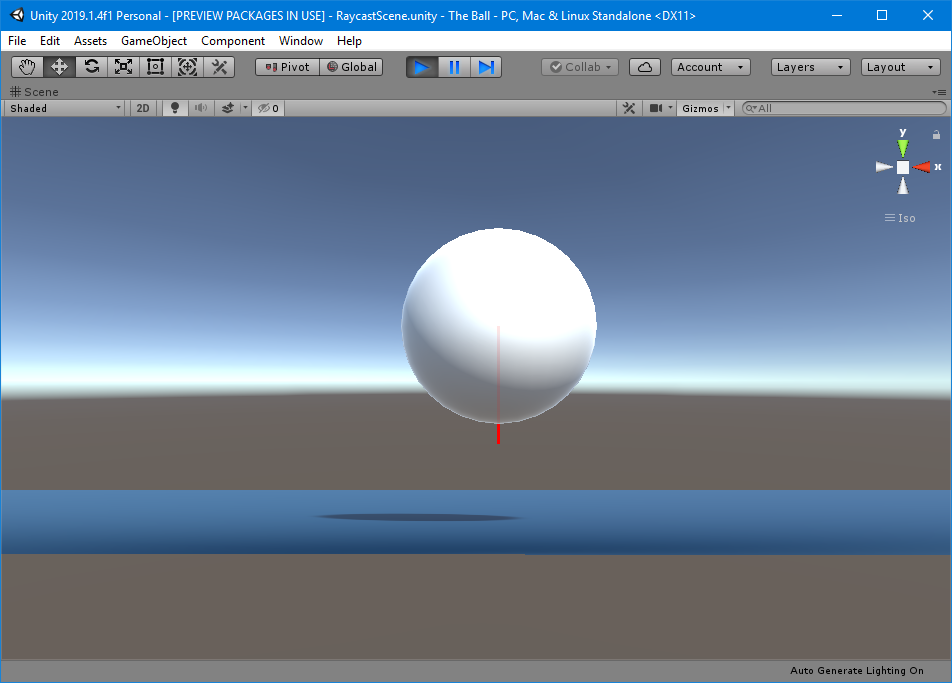
In Unity kunnen we de lijn afschieten met de functie Physics.RayCast. Als de afgeschoten lijn tegen een object aankomt, of er doorheen gaat, dan retourneert deze functie true.

In de afbeelding hieronder zie je een voorbeeld van een raycast.

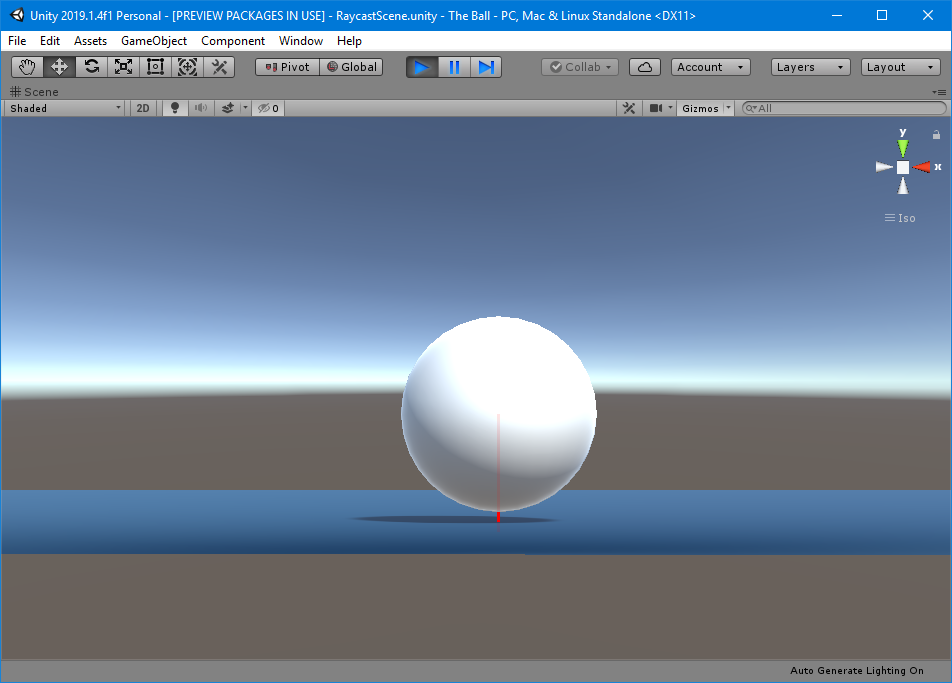


De rode lijn heeft als startpunt de linkerbovenhoek en beweegt zich naar rechtsonder. De linker pickup, de bal en de ondergrond worden geraakt, maar de pickup aan de rechterkant niet.

We kunnen lijn afschieten die net onder de bal uitkomt en als deze lijn een object raakt, dan weten we dat de bal mag springen. Onderstaande afbeeldingen laten dit principe zien.

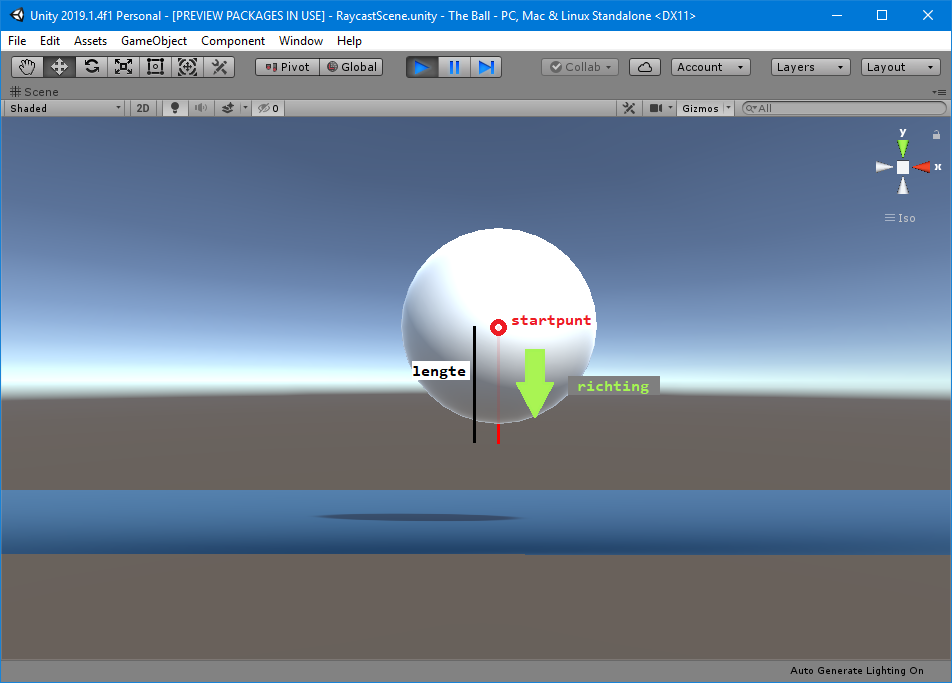


Het startpunt van de lijn is het midden van de bal en komt 0.1 meter onder de bal uit. De lijn raakt de grond niet en Physics.Raycast retourneert in dit geval false.



De lijn raakt de grond wel en Physics.Raycast retourneert true. Als je goed kijkt, dan zie je dat de bal een klein beetje boven de grond zweeft. Je zou dit kunnen oplossen door de lengte van de lijn korter te maken. Meestal is dit echter niet nodig en wordt het spel juist beter speelbaar als je kleine imperfecties toelaat.

De methode Physics.Raycast heeft drie parameters nodig: het startpunt van de lijn, de richting van de lijn en de lengte van de lijn. Deze parameters zijn in onderstaande afbeelding te zien:



* Voeg onderstaande code toe aan de methode isOpDeGrond:

| bool isOpDeGrond()  {  //Retourneert true als de bal zich op de grond bevindt  Vector3 startpunt = *// Vul hier het startpunt van de lijn in*  Vector3 richting = *// Vul hier de richting van de lijn in*  float lengte = 0.55f;   return Physics.Raycast(startpunt, richting, lengte);  } |
| --- |

* Gebruik de afbeelding hierboven om de juiste waarde voor de parameters startpunt en richting te bepalen. De variabele lengte is al voor ingevuld omdat het voor deze opdracht te omslachtig is om te bepalen wat grootte is van de bal.
* Test het spel. Als het goed is, springt de bal nu alleen als hij op de grond staat.

### Pas de wereld aan

Doordat de bal nu kan springen, kan de het onbedoeld het speelveld verlaten.

* Verhoog de randen om het speelveld op zo’n manier dat de bal niet meer kan ontsnappen.

# Uitbreidingen

## Vraag een testspeler

Een goed spel laat de speler zoveel mogelijk zelf ontdekken zonder dat hij, of zij een handleiding (of tutorial) nodig heeft.

* Pas het level zo aan, dat de een speler zelf gaat ontdekken dat de bal kan springen.
* Laat het spel door iemand spelen die het niet eerder heeft gezien, zonder dat je vertelt wat de bedoeling is en wat de bal allemaal kan. Kijk of de testspeler inderdaad uit zichzelf uitvindt wat de bedoeling is en dat de bal kan springen.

## Uitdaging: dubbele sprong

In sommige spellen kun je één keer extra springen als je in de lucht bent.

* Breid het script zo uit, dat de speler één extra sprong in de lucht kan maken.