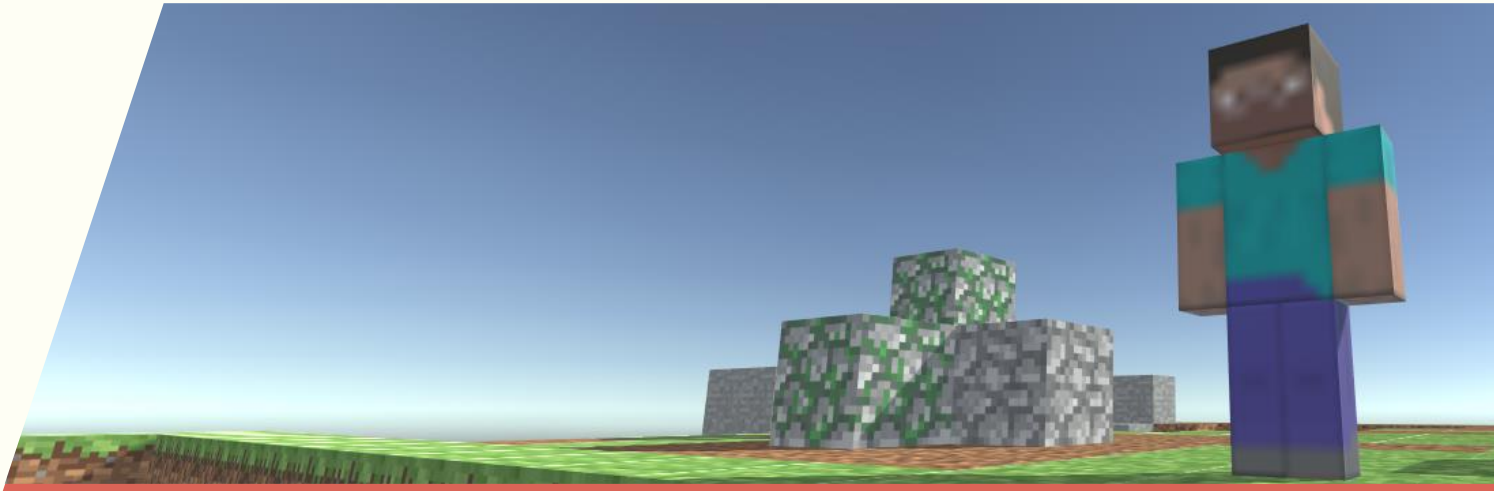


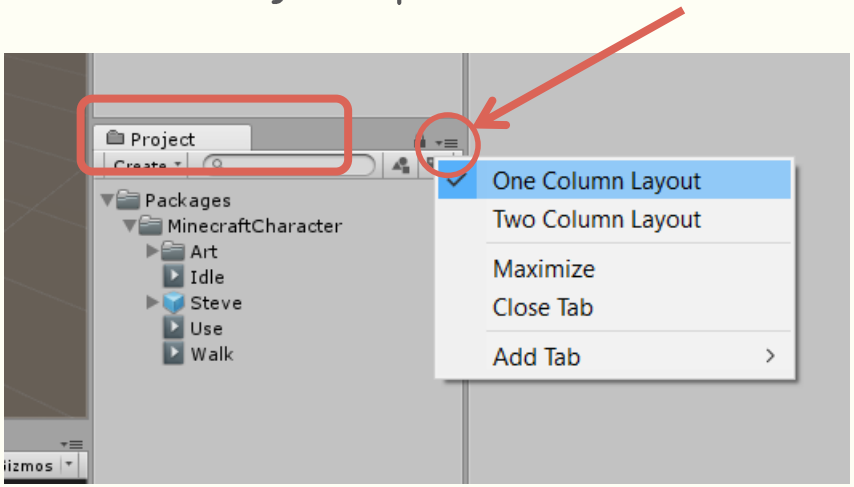
Bouw Minecraft in Unity

frederikwindey.blog

Wat we gaan maken

We beginnen eraan!

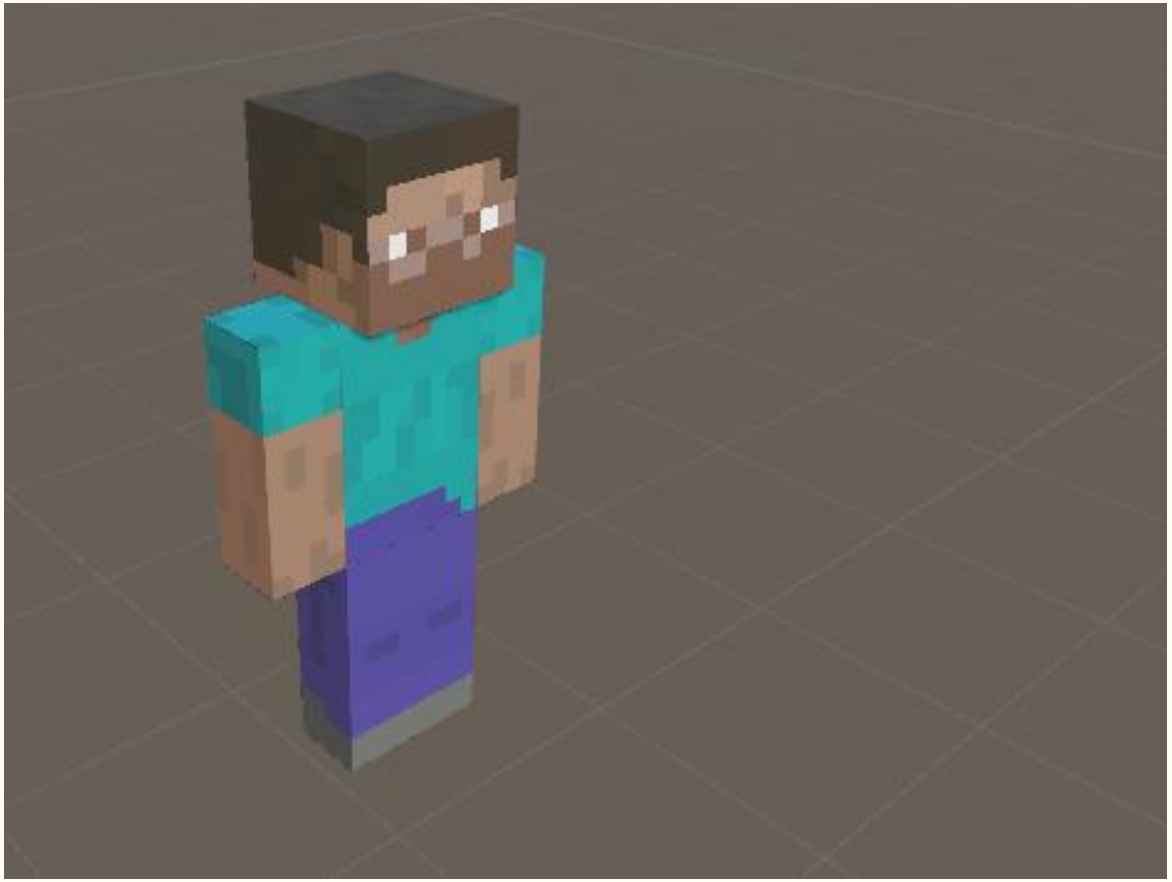
- Start Unity.
- Maak een nieuw 3D-project aan, en noem het *UnityCraft*.
- Switch de kolom layout van het **Project**-paneel naar **One column layout** door te klikken op het symbooltje zoals aangeduid op de figuur.

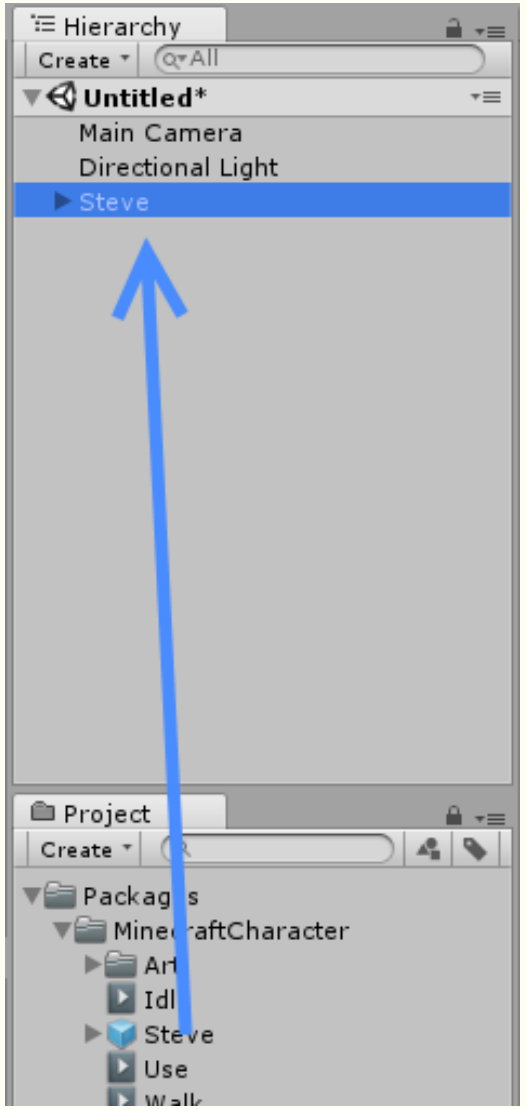


- Heb je de **Assets** ook gedownload voor deze tutorial? Je vindt ze op dezelfde pagina waar je deze tutorial hebt gedownload. Deze bevatten **unitypackages** die je zal nodig hebben.
- Importeer van deze Assets het **MinecraftSteve** unitypackage:
  - Assets → Import Package → Custom Package

Steve in de scene plaatsen

- Sleep de Steve prefab in the scene



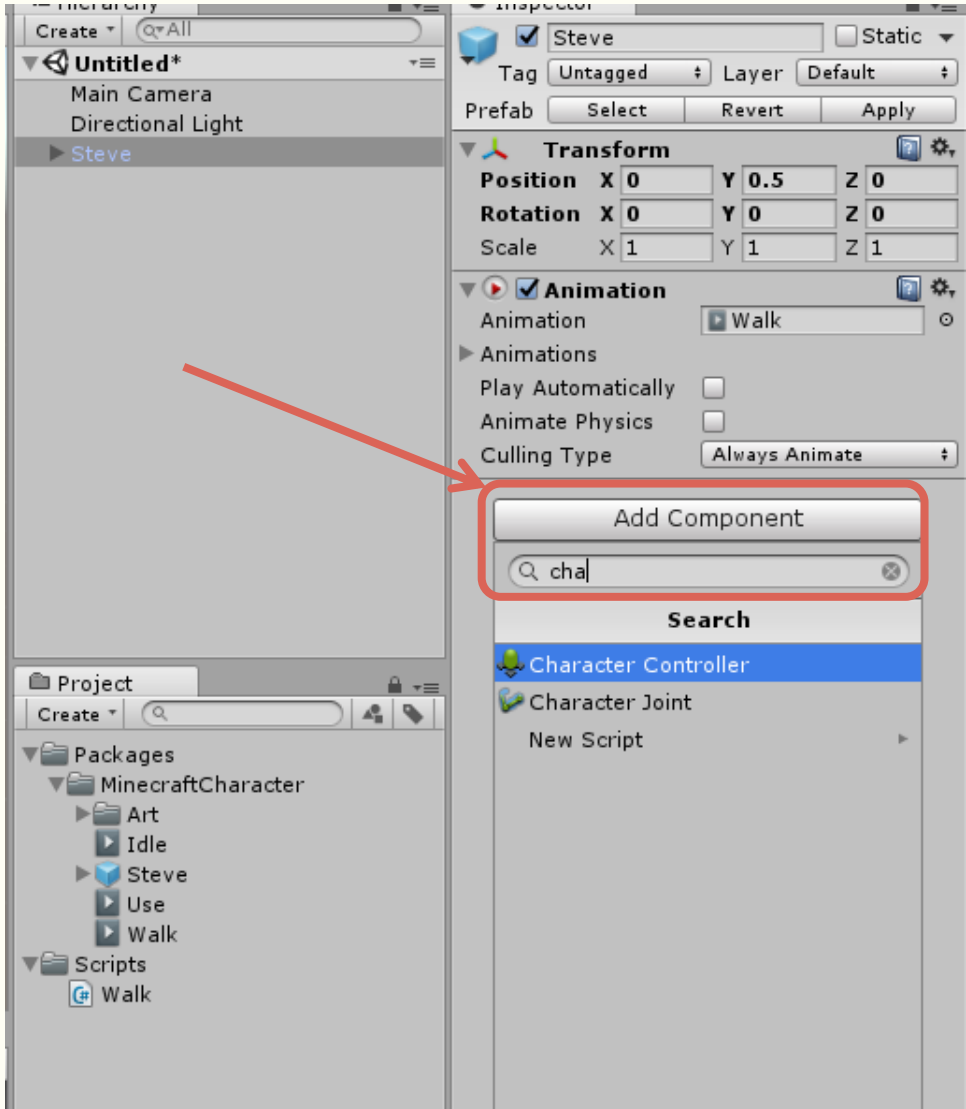


Character Controller toevoegen

- De **Character Controller** component laat ons toe om Steve te laten voortbewegen in de wereld zonder dat hij doorheen muren kan stappen of door de grond valt.

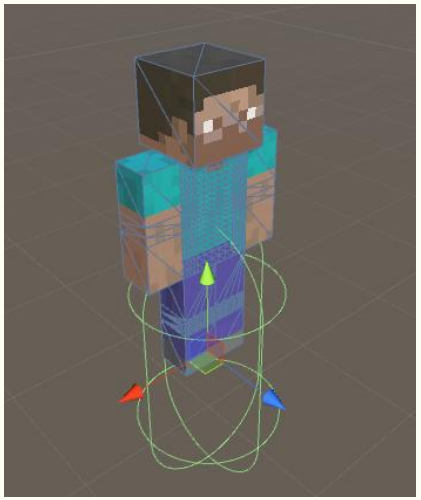
Character Controller toevoegen

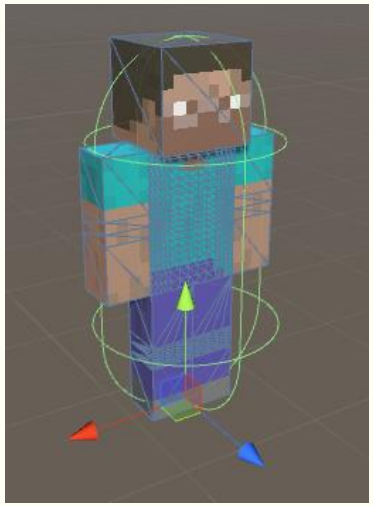
- Selecteer Steve in de scene.
- Kies **Add Component** in de **Inspector.**
- Typ de eerste letters van de component in: **cha**
- Voeg de **Character Controller** toe.

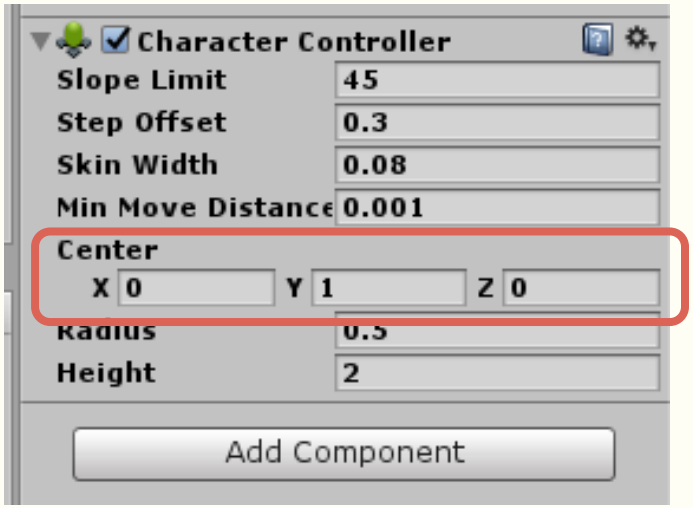


Character Controller toevoegen

- Je zal zien dat de Character Controller niet mooi gealigneerd is met Steve.
- Pas de waarden van de Character Controller in de Inspector aan zodat Steve mooi in de Character Controller past.

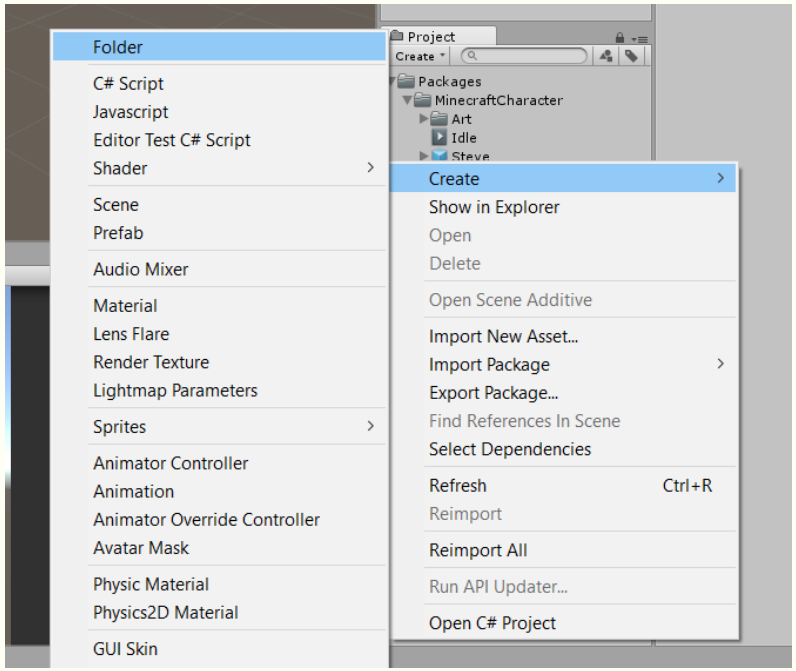


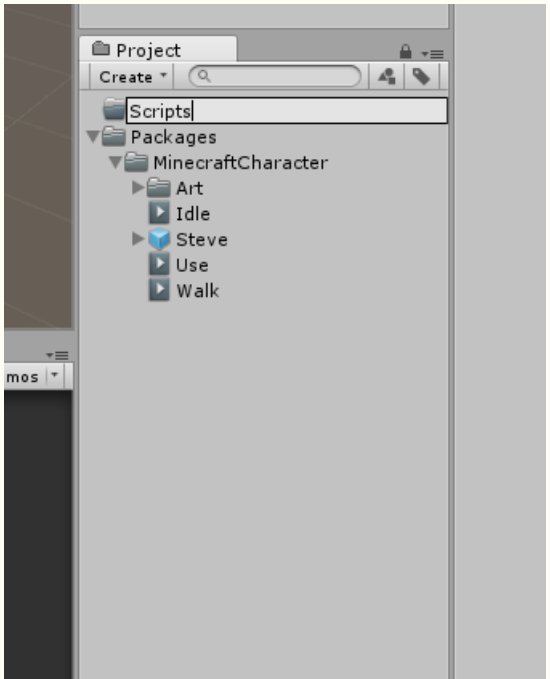




Steve vooruit laten bewegen

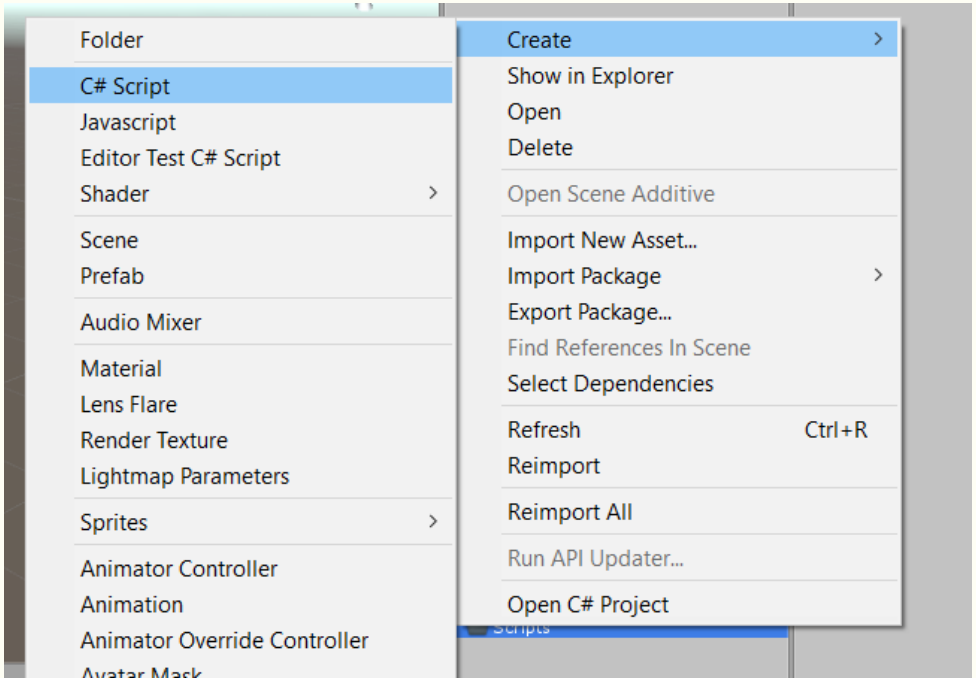
- Maak een nieuwe folder aan in het project-paneel met de naam **Scripts**:
  - Rechtermuisknop in het projectpaneel, **Create → Folder**

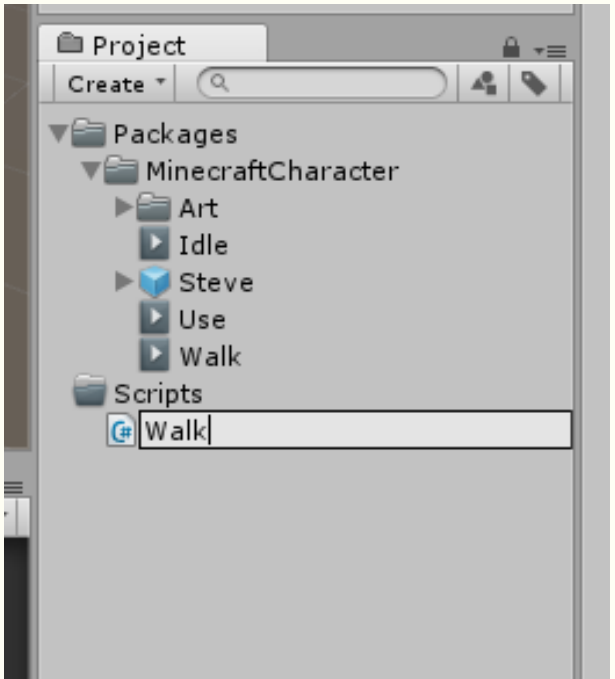




Steve vooruit laten bewegen

- Maak een nieuwe C# script met de naam **Walk** in deze folder:
  - Rechtermuisknop op de **Scripts** folder, **Create → C# Script**





Steve vooruit laten bewegen

- Dubbelklik op het **Walk** script om het te openen.
- We hebben de **Character Controller** nodig om Steve te laten bewegen.
- De eerste stap is dan ook om er een referentie naar te maken.
- We gaan de Character Controller opvragen als de game start, dus in de **Start** functie.

Steve vooruit laten bewegen

```
public class Walk : MonoBehaviour {

    private CharacterController controller;

    void Start ()
    {
        controller = GetComponent<CharacterController>();
    }

    void Update ()
    {

    }
}
```

- Een referentie naar de Character Controller wordt bijgehouden in de **controller** variabele bij de start van de game.

Steve vooruit laten bewegen

```
public class Walk : MonoBehaviour {

    public float WalkSpeed;

    private CharacterController controller;
    private Vector3 moveDirection;

    void Start ()
    {
        controller = GetComponent<CharacterController>();
    }

    void Update ()
    {

    }
}
```

- We hebben nog 2 extra variabelen nodig.
  - Een publieke **WalkSpeed** zodat we de snelheid kunnen aanpassen waarmee Steve wandelt
  - **moveDirection** zal gebruikt worden om de beweging die Steve moet maken door te geven aan de **Character Controller**

Steve vooruit laten bewegen

```
public class Walk : MonoBehaviour {

    public float WalkSpeed;

    private CharacterController controller;
    private Vector3 moveDirection;

    void Start ()
    {
        controller = GetComponent<CharacterController>();
    }

    void Update ()
    {

    }
}
```

- **WalkSpeed** is een variabele van het type **float**. Een float is een decimaal getal.
- Naast float bestaan er nog andere types van getallen, zoals de **int**, wat staat voor integer. Integers zijn gehele getallen, dus zonder cijfers na de komma.

Steve vooruit laten bewegen

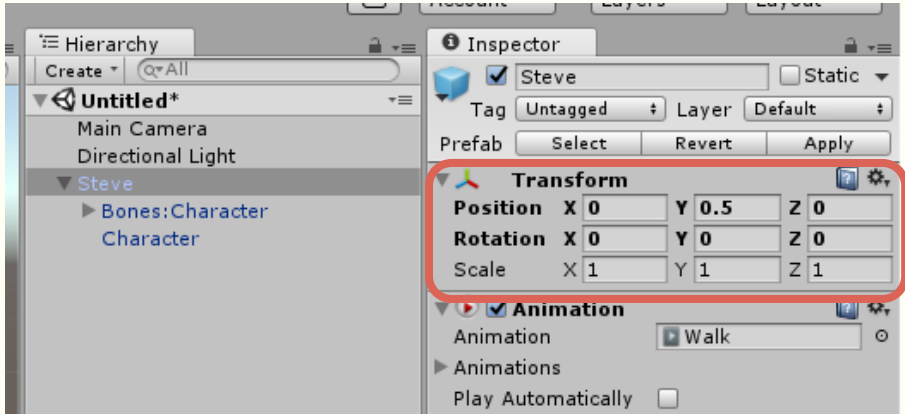
```
public class Walk : MonoBehaviour {

    public float WalkSpeed;

    private CharacterController controller;
    private Vector3 moveDirection;

    void Start ()
    {
        controller = GetComponent<CharacterController>();
    }

    void Update ()
    {

    }
}
```

- **moveDirection** is een variable van het type **Vector3**.
- Dat betekent dat deze een x, y en z component heeft.
- moveDirection kan je dan ook zien als de **bewegingsvector** van Steve, die aanduidt hoeveel hij beweegt in de richting van de x-as, y-as en z-as.

Steve vooruit laten bewegen

```
void Update ()
{
    if (Input.GetKey(KeyCode.UpArrow))
    {
        moveDirection = transform.forward * WalkSpeed * Time.deltaTime;
    }
}
```

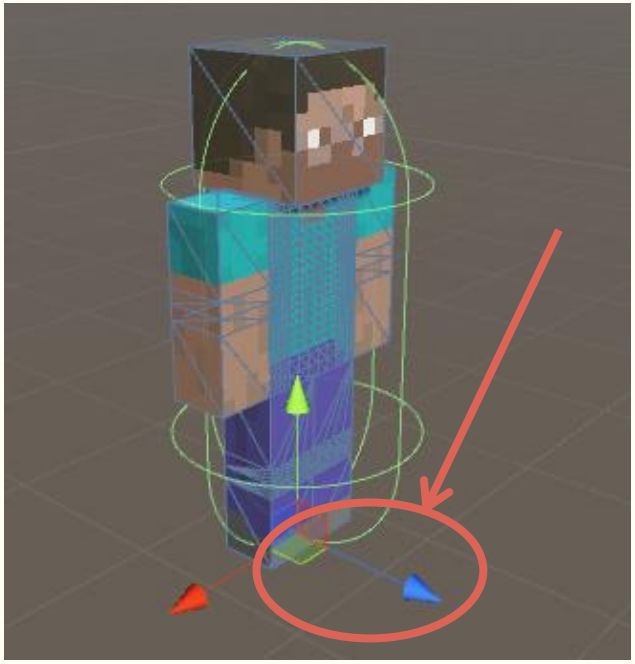
- We voegen onze eerste code toe aan de Update functie:
  - Als de gebruiker **Pijltje Omhoog** (UpArrow) indrukt, willen we Steve laten bewegen in de richting naar waar hij kijkt.
  - Als je Steve selecteert in je Hierarchy paneel zal je zien dat deze een Transform component heeft.
  - Deze Transform component houdt de positie, rotatie en schaal bij van Steve.



Steve vooruit laten bewegen

```
void Update ()
{
    if (Input.GetKey(KeyCode.UpArrow))
    {
        moveDirection = transform.forward * WalkSpeed * Time.deltaTime;
    }
}
```

- We voegen onze eerste code toe aan de Update functie:
  - Als je Steve selecteert in de scene, zal je zien dat hij in de richting van de blauwe as kijkt (de Z-as).
  - Deze komt overeen met **transform.forward** Dit is een conventie in Unity.
  - We willen Steve dus laten bewegen in de richting van deze as als we Pijltje Omhoog indrukken.



Steve vooruit laten bewegen

```
void Update ()
{
    if (Input.GetKey(KeyCode.UpArrow))
    {
        moveDirection = transform.forward * WalkSpeed * Time.deltaTime;
    }
}
```

- We voegen onze eerste code toe aan de Update functie:
  - Op het einde van de lijn code zie je ook nog **Time.deltaTime** staan.
  - Dit is een waarde die groter wordt als je computer trager is, en kleiner wordt als je computer sneller is. Of om het in technische termen te zeggen: deze waarde is omgekeerd evenredig met de frame rate van je game.
  - Dit zorgt ervoor dat Steve even snel gaat bewegen op een trage of snelle computer.
  - Stel je dus voor dat je die waarde niet zou toevoegen, dan zou heel het spel anders spelen op een oude of nieuwe computer!
  - Je ziet: je moet met heel wat rekening houden bij het maken van games!

Steve vooruit laten bewegen

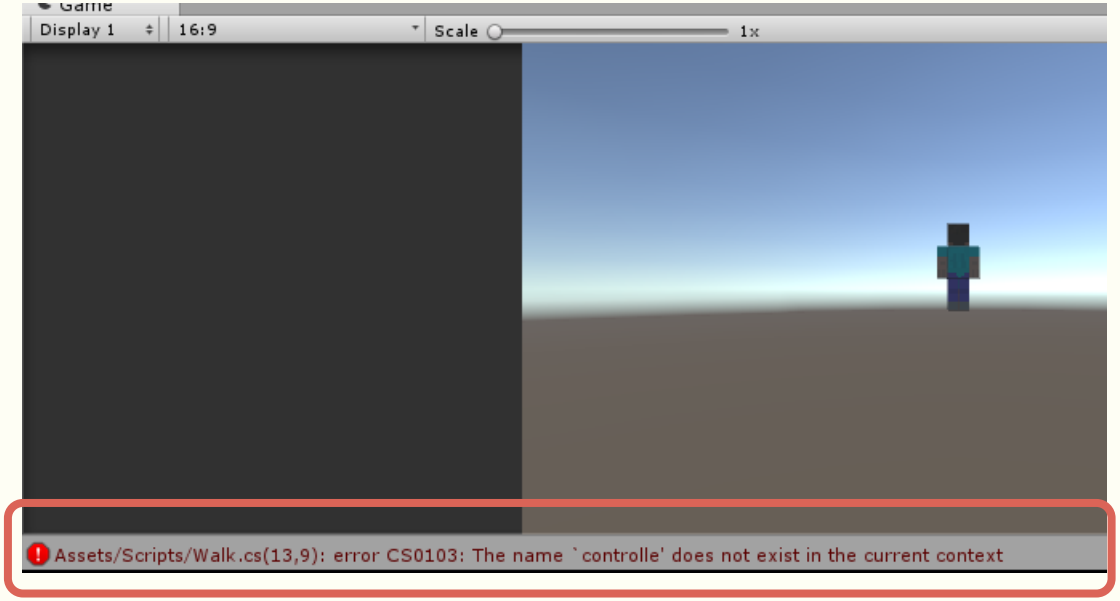
```
void Update ()
{
    if (Input.GetKey(KeyCode.UpArrow))
    {
        moveDirection = transform.forward * WalkSpeed * Time.deltaTime;
    }
}
```

- We voegen onze eerste code toe aan de Update functie:
  - Nu zal er nog niets gebeuren, want we moeten de **moveDirection** nog toepassen op de Character Controller.
  - Dit doe je aan de hand van de Move functie van de Character Controller, zoals onderstaand stukje code:

```
void Update ()
{
    if (Input.GetKey(KeyCode.UpArrow))
    {
        moveDirection = transform.forward * WalkSpeed * Time.deltaTime;
    }

    controller.Move(moveDirection);

}
```

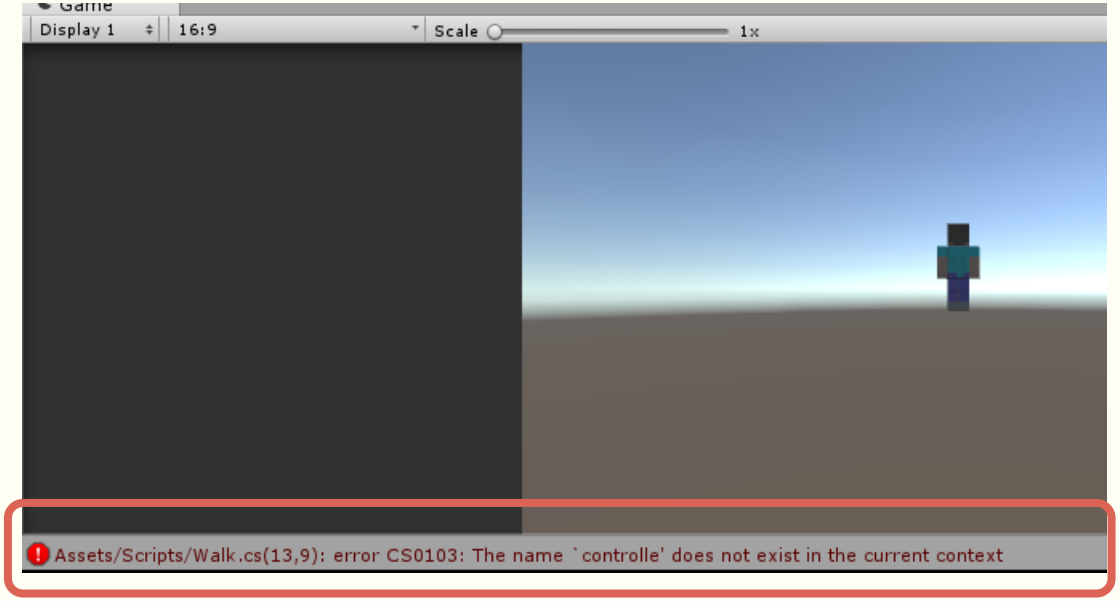
Steve vooruit laten bewegen

- Ga nu terug naar Unity.
- Het script zal automatisch compileren, wat betekent dat Unity eerst gaat kijken of er foutjes in de code staan, waarna de code wordt vertaald in machinecode die de computer kan begrijpen.
- Als er foutjes in je code staan, zal je dat zien in de **Console** onderaan je scherm.



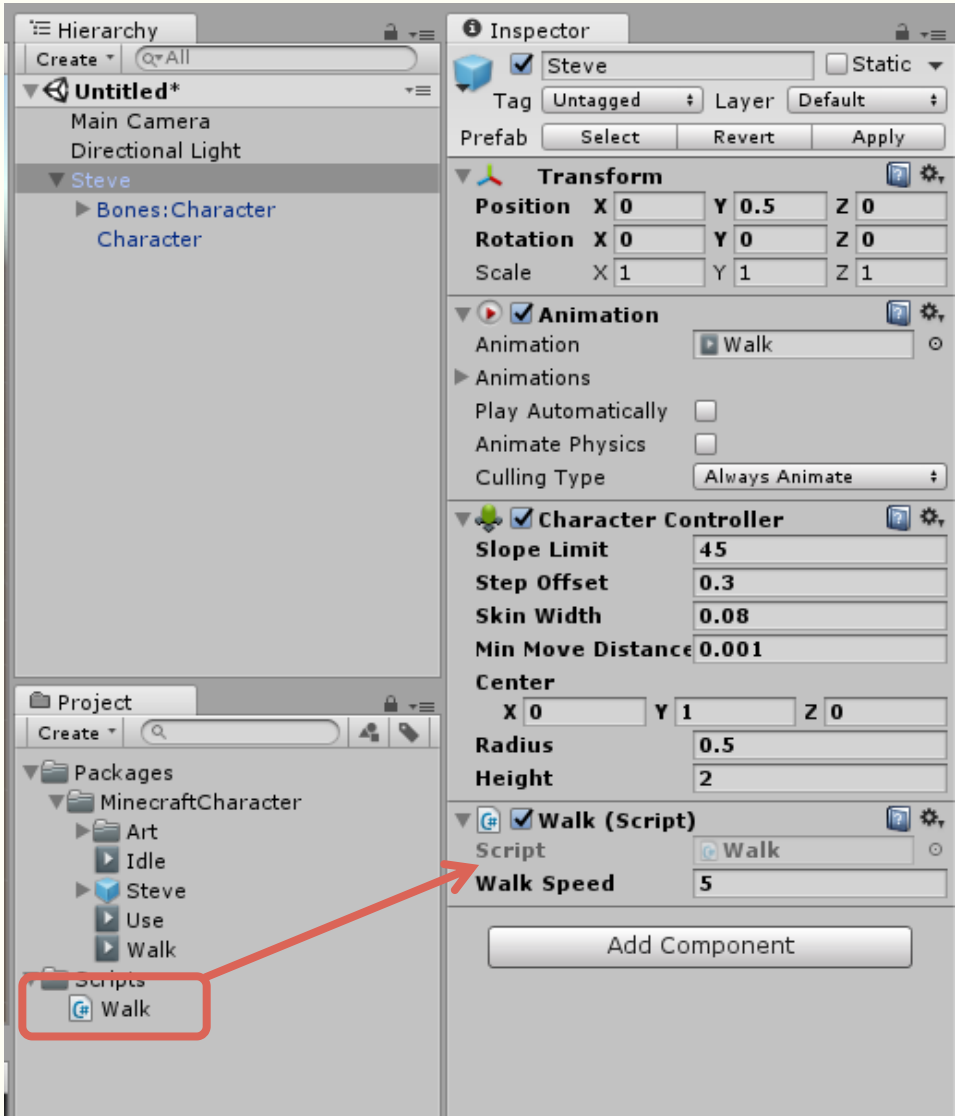
Steve vooruit laten bewegen

- Probeer eens vrijwillig een foutje in je code te plaatsen, door bijvoorbeeld een letter te verwijderen uit een stukje code.
- Ga dan terug naar Unity, en kijk of er een fout verschijnt in de Console.
- Corrigeer je code terug, en kijk of je code nu geen foutjes meer bevat door terug naar Unity te gaan en te kijken naar de Console.



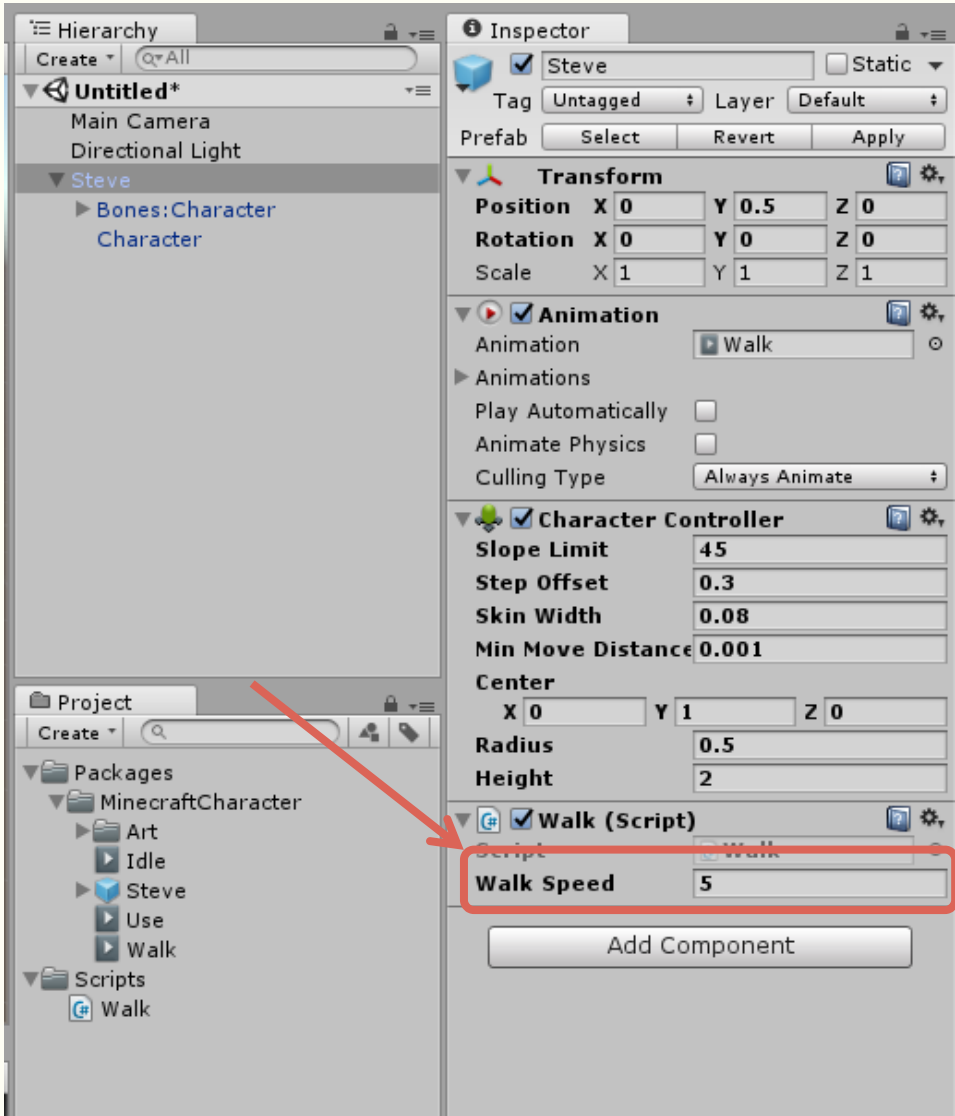
Steve vooruit laten bewegen

- De laatste stap om Steve te laten bewegen is het script aan hem toe te wijzen.
- Je kan dit doen door eerst Steve te selecteren en dan je Walk script te slepen op Steve in de Inspector.
- Probeer je nieuwe script nu uit door op Play te drukken!



Steve vooruit laten bewegen

- Er zal nog niets gebeuren!
- Je moet eerst de variable WalkSpeed in de Inspector een waarde geven verschillend van nul, bijvoorbeeld 5.
- Herinner je dat we de variable WalkSpeed in de code **public** hadden gemaakt? Wel, dat doen we zodat deze dan zou verschijnen in de Inspector, zoals dat bij WalkSpeed gebeurt.
- Op deze manier kunnen we heel gemakkelijk ons game gaan optimaliseren door variabelen aan te passen terwijl je je game test, zonder je daarvoor steeds terug de code moet induiken. Dit noemen we ook het **tweaken** van variabelen.



Steve vooruit laten bewegen

- Als je Pijltje Omhoog duwt, zal Steve vertrekken… maar ook niet meer stoppen!
- Dit komt omdat we de variabele **moveDirection** niet op nul zetten als er geen toets is ingedrukt.
- Dat zullen we even oplossen:

```
void Update ()
{
    if (Input.GetKey(KeyCode.UpArrow))
    {
        moveDirection = transform.forward * WalkSpeed * Time.deltaTime;
    }
    else
    {
        moveDirection = Vector3.zero;
    }

    controller.Move(moveDirection);
}
```

- Als de gebruiker op Pijltje omhoog drukt, zetten we de bewegingsvector van Steve op **vooruit**, en als er geen toets wordt ingedrukt, zetten we de bewegingsvector op de **nulvector**, wat betekent: geen beweging volgens de x-, y- of z-as.

Steve vooruit laten bewegen

- Probeer uit of het nu wel werkt.
- Nu is het aan jou om Steve achteruit te laten bewegen als we Pijltje Omlaag intoetsen.
- Probeer het eerst zelf te zoeken, voordat je naar de oplossing op de volgende slide gaat kijken.

Steve vooruit en achteruit laten bewegen

```
void Update ()
{
    if (Input.GetKey(KeyCode.UpArrow))
    {
        moveDirection = transform.forward * WalkSpeed * Time.deltaTime;
    }
    else if(Input.GetKey(KeyCode.DownArrow))
    {
        moveDirection = -transform.forward * WalkSpeed * Time.deltaTime;
    }
    else
    {
        moveDirection = Vector3.zero;
    }

    controller.Move(moveDirection);
}
```

- Wat ontbreekt er nog om Steve goed en wel door de wereld te laten stappen?

  Naar links en rechts draaien!

Maar eerst opslaan!

- Druk Ctrl + S om je scene op te slaan.
- Geef de scene de naam **UnityCraft**.

Steve laten draaien

- Als je al wat ervaring hebt met Unity kan je nu proberen om zelf Steve te laten ronddraaien, zonder te kijken naar de oplossing.
- Probeer dezelfde werkwijze te volgen als bij het vooruit- en achteruitbewegen.
  - Voorzie een public variabele zodat we de draaisnelheid van Steve makkelijk kunnen **tweaken**.
  - Voor het ronddraaien heb je de Character Controller niet nodig. Steve zal met niets kunnen botsen tijdens het draaien, en heeft de Character Controller dan ook niet nodig om mogelijke botsingen te detecteren.
  - Je kan voor het ronddraaien de methode Rotate gebruiken van de Transform component, bijvoorbeeld:

```
transform.Rotate(0, RotationSpeed * Time.deltaTime, 0);
```

  - Hou net zoals bij het vooruitbewegen rekening met de **deltaTime**. Anders gaat Steve veel sneller ronddraaien op een snellere computer!

Steve laten draaien

- Heb je het zelf geprobeerd om Steve te laten ronddraaien met alle nodige richtlijnen op de vorige slide?
- Kijk dan naar de volgende slide voor de oplossing!

Steve laten draaien

```
public class Walk : MonoBehaviour {

    public float WalkSpeed;
    public float RotationSpeed;

    ...

    void Update ()
    {

        ...

        if (Input.GetKey(KeyCode.LeftArrow))
        {
            transform.Rotate(0, -RotationSpeed * Time.deltaTime, 0);
        }
        else if (Input.GetKey(KeyCode.RightArrow))
        {
            transform.Rotate(0, RotationSpeed * Time.deltaTime, 0);
        }

    }
```

- Vergeet niet om de **RotationSpeed** variabele een waarde te geven in de inspector, zoals je gedaan hebt met de WalkSpeed variabele.
  160 is een goede waarde, maar je beslist natuurlijk zelf hoe goed Steve is in het ronddraaien!

Goed gedaan!

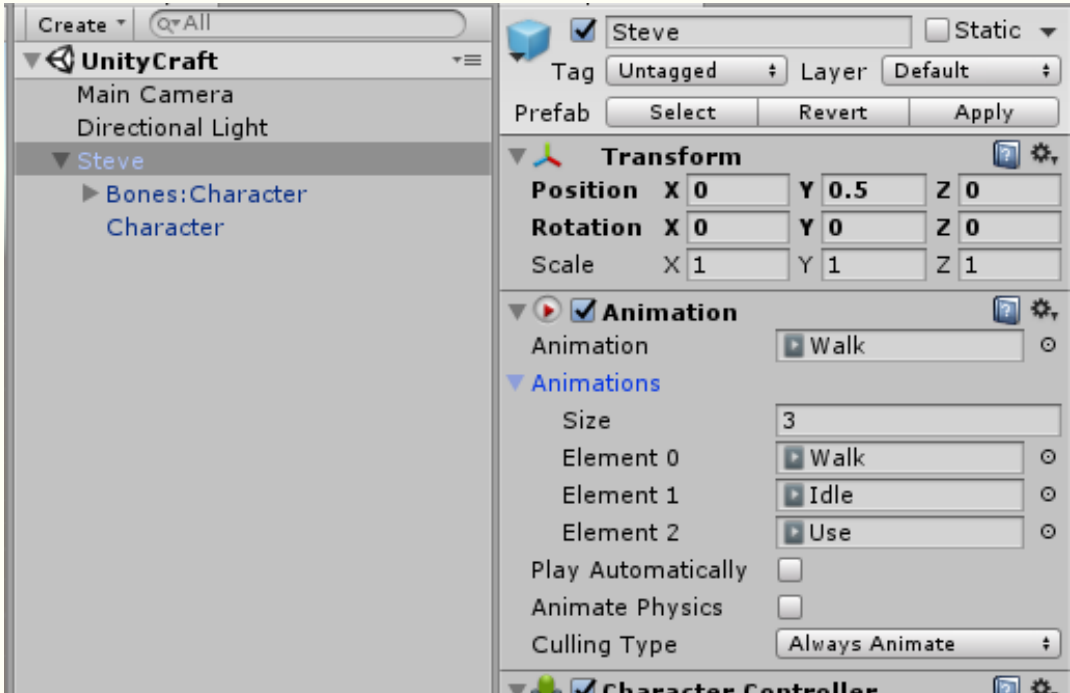
- We hebben nu alvast een Steve die kan rondbewegen!
- In de volgende stap gaan we Steve een beetje meer tot leven brengen door hem te **animeren** als hij rondstapt.
- Animeren is een heel groot onderdeel bij het maken van games waarbij men bijvoorbeeld de ledematen van het hoofdkarakter (in dit geval Steve) laat bewegen als die rondloopt, iets werpt of opneemt, valt, springt, …
- Op die manier brengt men de virtuele wereld van de game veel meer tot leven.

Maar eerst opslaan!

- Druk Ctrl + S om je scene op te slaan.

Steve animeren

- In Unity kan je ook animaties maken, maar dat zou ons te ver leiden tijdens deze oefening.
- Steve heeft al een aantal animaties toegewezen gekregen, zoals je kan zien als je hem selecteert in het hierarchy paneel.
- Er zijn drie animaties gemaakt voor Steve:
  - Walk: wandelen
  - Idle: stilstaan
  - Use: gebruiken



Steve animeren

- Je ziet in de Inspector in Unity dat de animaties zijn ondergebracht in de **Animations** component.
- We moeten deze dan ook eerst opvragen en opslaan als een referentie in onze code, willen we toegang krijgen tot de animaties:

```
public class Walk : MonoBehaviour {

    public float WalkSpeed;
    public float RotationSpeed;

    private Animation animations;
    private CharacterController controller;
    Vector3 moveDirection = Vector3.zero;


    void Start()
    {
        animations = GetComponent<Animation>();
        controller = GetComponent<CharacterController>();
    }

    void Update()
    {
        ...
```

- Je ziet dat we hier exact dezelfde techniek gebruiken als voor de Character Controller om een referentie naar de Animation component op te slaan

Steve animeren

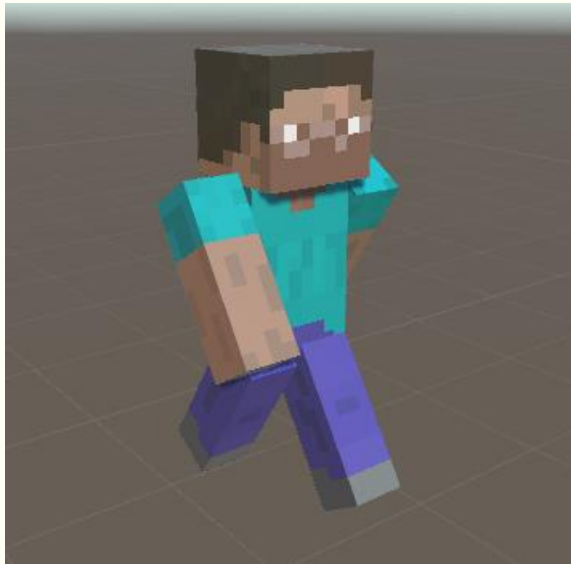
- Eens we de Animation component hebben, is het eenvoudig om de animaties op Steve toe te passen.
- Je kan bijvoorbeeld de **Walk** animatie afspelen met de volgende code:

```
animations.CrossFade("Walk");
```

- Probeer nu zelf om dit stukje code op de juiste plaats (of meerdere plaatsen!) te zetten, zonder naar de volgende slide te kijken!

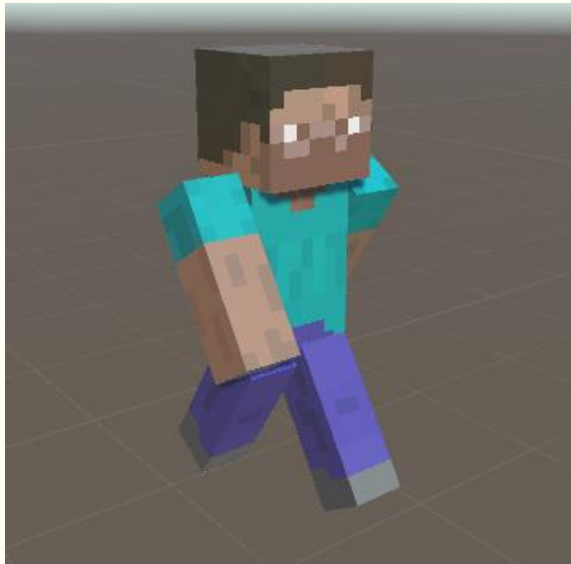
Steve animeren

```
void Update ()
{
    if (Input.GetKey(KeyCode.UpArrow))
    {
        animations.CrossFade("Walk");
        moveDirection = transform.forward * WalkSpeed * Time.deltaTime;
    }
    else if(Input.GetKey(KeyCode.DownArrow))
    {
        animations.CrossFade("Walk");
        moveDirection = -transform.forward * WalkSpeed * Time.deltaTime;
    }
    else
    {
        moveDirection = Vector3.zero;
    }

    if (Input.GetKey(KeyCode.LeftArrow))
    {
        transform.Rotate(0, -RotationSpeed * Time.deltaTime, 0);
    }
    else if (Input.GetKey(KeyCode.RightArrow))
    {
        transform.Rotate(0, RotationSpeed * Time.deltaTime, 0);
    }

    controller.Move(moveDirection);
}
```



Steve animeren

- Wow! Dat ziet er toch al veel interessanter uit als Steve rondloopt, niet?
- Probleem is nu dat Steve niet wil stoppen met bewegen als we de pijltjestoetsen loslaten!
- Daarvoor hebben we dus de **Idle** animatie nodig Deze doet eigenlijk niets, gewoon stilstaan. Maar je zou evengoed een interessante Idle animatie kunnen voorzien, zoals rondkijken, aan zijn hoofd krabben, …



Steve animeren

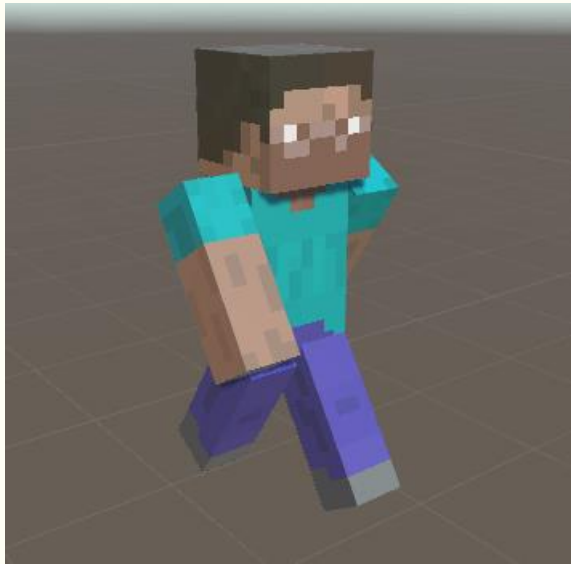
- Wanneer moeten we de Idle animatie afspelen?
  - Als er geen invoer is van de gebruiker!
- Hoe gaan we dat controleren? Heb je zelf een idee?

Steve animeren

- Er zijn verschillende mogelijkheden om te detecteren of de gebruiker een toets heeft ingedrukt. Hier is een tip hoe je het zou kunnen aanpakken:

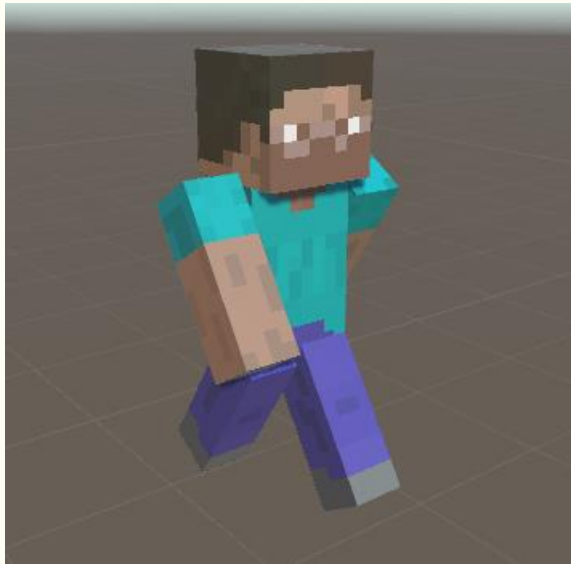
```
bool isMoving = false;

if (Input.GetKey(KeyCode.UpArrow))
{
    isMoving = true;
    animations.CrossFade("Walk");
    moveDirection = transform.forward * WalkSpeed * Time.deltaTime;
}

...

if (!isMoving)
{
    ...
```

- Zie je al waar we naartoe gaan? Probeer de nodige code aan te vullen aan de hand van de bovenstaande stukjes code, zonder op de volgende slide te kijken!



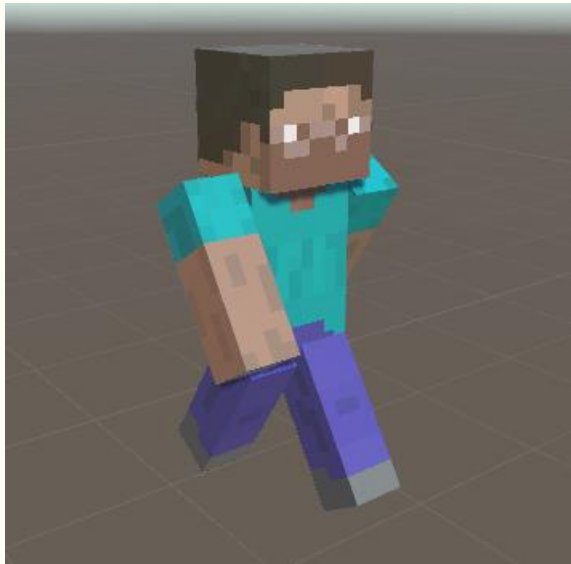
Steve animeren

```
void Update ()
{
    bool isMoving = false;

    if (Input.GetKey(KeyCode.UpArrow))
    {
        isMoving = true;
        animations.CrossFade("Walk");
        moveDirection = transform.forward * WalkSpeed * Time.deltaTime;
    }
    else if(Input.GetKey(KeyCode.DownArrow))
    {
        isMoving = true;
        animations.CrossFade("Walk");
        moveDirection = -transform.forward * WalkSpeed * Time.deltaTime;
    }
    else
    {
        moveDirection = Vector3.zero;
    }

    if (Input.GetKey(KeyCode.LeftArrow))
    {
        transform.Rotate(0, -RotationSpeed * Time.deltaTime, 0);
    }
    else if (Input.GetKey(KeyCode.RightArrow))
    {
        transform.Rotate(0, RotationSpeed * Time.deltaTime, 0);
    }

    if(!isMoving)
    {
        animations.CrossFade("Idle");
    }

    controller.Move(moveDirection);
}
```



Steve animeren

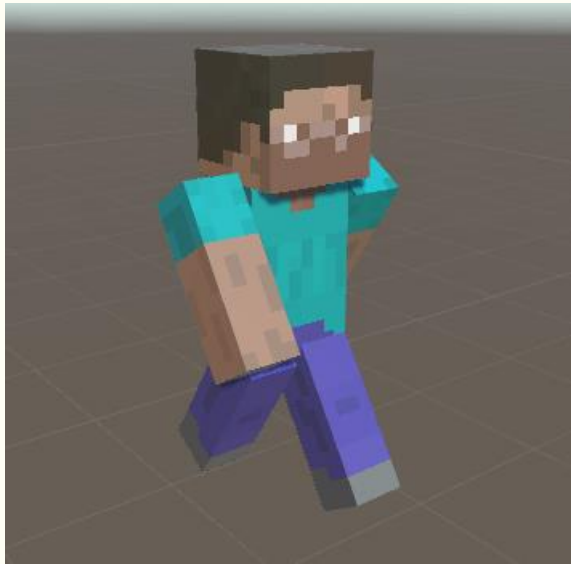
- We kunnen de code ook een beetje vereenvoudigen, nu we de **isMoving** variable hebben. Heb je een idee hoe?
- Het heeft iets te maken met het afspelen van de **Walk** animatie.



Steve animeren

- We plaatsen het afspelen van de **Walk** animatie mooi naast het afspelen van de **Idle** animatie!

```
void Update ()
{
    bool isMoving = false;

    if (Input.GetKey(KeyCode.UpArrow))
    {
        isMoving = true;
        moveDirection = transform.forward * WalkSpeed * Time.deltaTime;
    }
    else if(Input.GetKey(KeyCode.DownArrow))
    {
        isMoving = true;
        moveDirection = -transform.forward * WalkSpeed * Time.deltaTime;
    }
    else
    {
        moveDirection = Vector3.zero;
    }

    if (Input.GetKey(KeyCode.LeftArrow))
    {
        transform.Rotate(0, -RotationSpeed * Time.deltaTime, 0);
    }
    else if (Input.GetKey(KeyCode.RightArrow))
    {
        transform.Rotate(0, RotationSpeed * Time.deltaTime, 0);
    }

    if(!isMoving)
    {
        animations.CrossFade("Idle");
    }
    else
    {
        animations.CrossFade("Walk");
    }

    controller.Move(moveDirection);
}
```



Goed gedaan!

- We hebben een Steve die rondloopt en geanimeerd is, net zoals in Minecraft!
- Sla je scene op door op Ctrl + S te drukken.

We gaan een wereld bouwen!

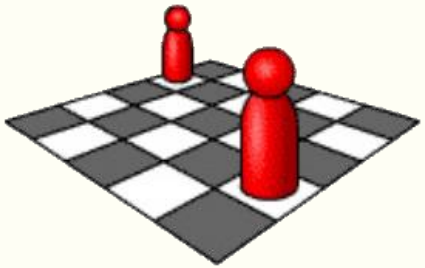
- Nu we Steve kunnen laten rondlopen, willen we hem natuurlijk in een interessante wereld plaatsen om te verkennen.
- Eerst en vooral gaan we bepalen vanuit welk oogpunt we Steve en de wereld rond hem gaan bekijken.
- Minecraft is vooral bekeken vanuit eerste persoon, of beter bekend als **first person**.
- We gaan nu voor dit project eens iets nieuws proberen!
- We kiezen voor een **isometrische projectie**, zoals het voorbeeld hiernaast.
- Deze weergave heeft geen echt perspectief, en bekijkt de wereld schuin van bovenaf.

Isometrische projectie

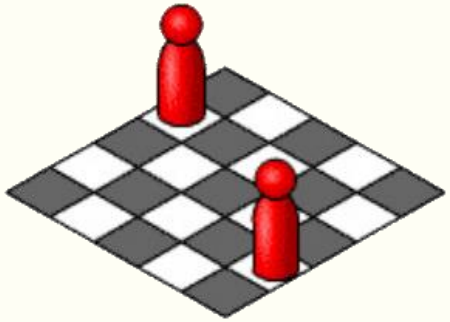
- Om de wereld in isometrische projectie te plaatsen kan je in de scene de camera selecteren en in de **Transform** en **Camera** component ervan de waarden invullen zoals je hiernaast kan zien.
- De camera zal zo schuin naar beneden kijken onder een hoek van 40 graden.
- Kijk nog even na of je overal dezelfde waarden hebt als hiernaast.

Isometrische projectie

- Bij deze weergavevorm willen we geen perspectief, dus moeten we de camera-projectie op **Orthographic** (ortographisch) instellen.
- Dit zorgt ervoor dat er geen echt perspectief is, dus dat voorwerpen niet kleiner worden weergegeven naarmate ze verder van de camera worden geplaatst.



Echt perspectief



Isometrische projectie

Isometrische projectie

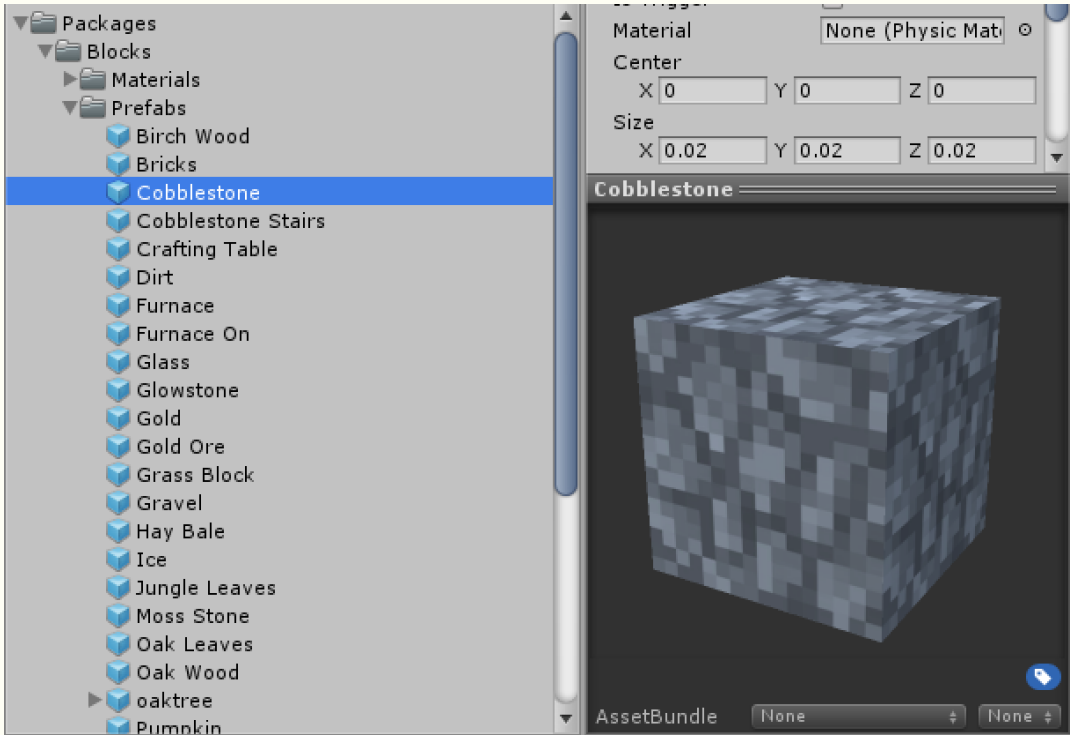
Als we nu Steve op de volgende positie plaatsen:

x: 0
y: 0.5
z: 0

zou je game view er ongeveer zoals hiernaast moeten uitzien.

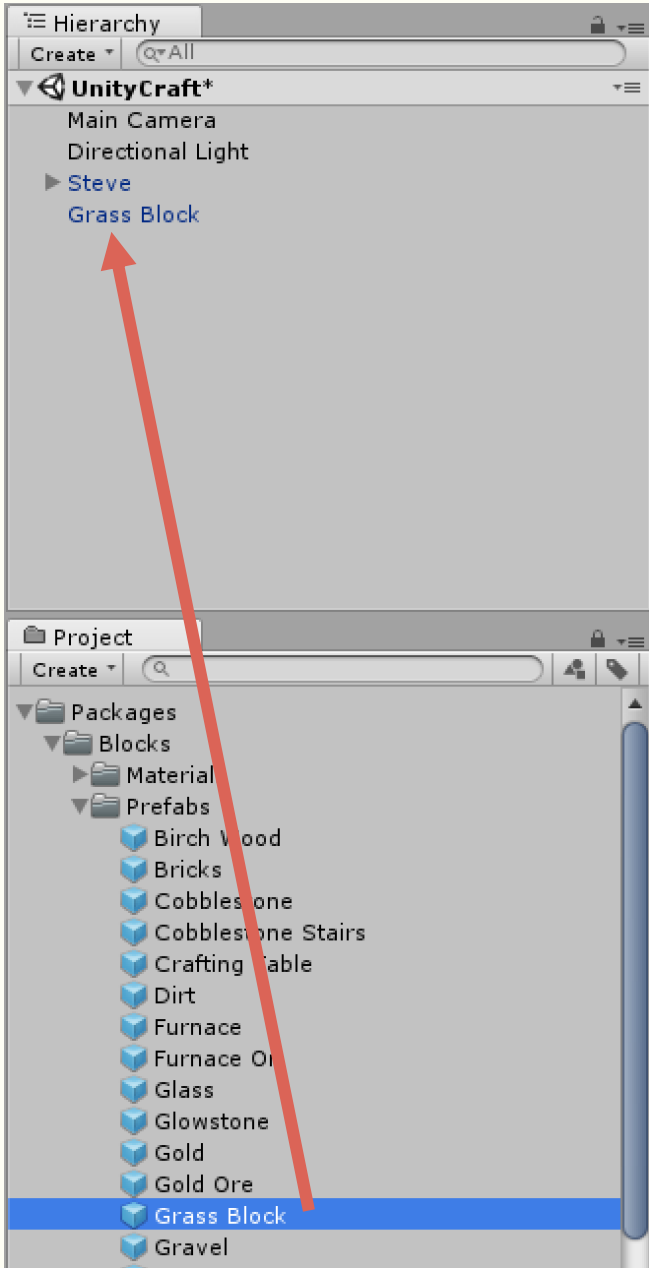
Minecraft blokken importeren

- Je mag nu het **MinecraftBocks** unitypackage importeren.
- Dit package bevatten een 30-tal Minecraft blokken waarmee je een wereld voor Steve zal kunnen bouwen.

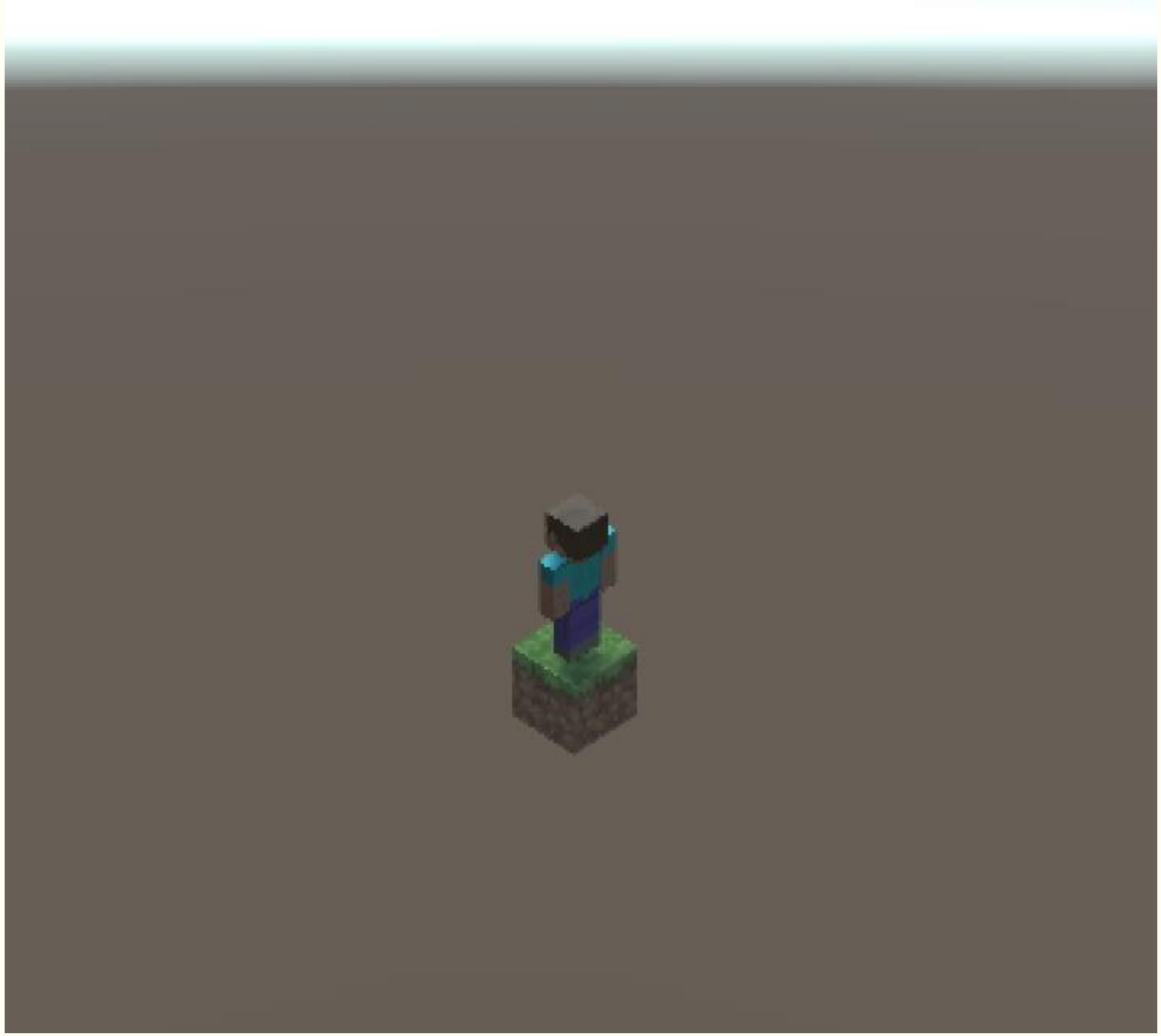


Eerste Minecraft-blok plaatsen!

- We zullen een grasblok plaatsen onder de voeten van Steve!
- Sleep de **Grass Block** prefab van in de project view naar de hierarchy view.
- Plaats de Grass Block op de positie x: 0 y: 0 z: 0



Steve heeft gras onder zijn voeten!



Bouw een Minecraft-wereld

- Je kan nu blokken in de scene plaatsen om een echte Minecraft-wereld te bouwen!
- Nog enkele tips om efficiënter blokken te plaatsen:
  - **vertex snapping**: selecteer een blok in de wereld, houd **V** ingedrukt terwijl je met de muis over een van de hoekpunten van een blok gaat. Blijf V ingedrukt houden, terwijl je ook de linkermuisknop ingedrukt houdt en sleept. Zo laat je de blok **snappen** met het geselecteerde hoekpunt aan andere blokken.

    Je kan dit ook doen met groepen van blokken: selecteer meerdere blokken tegelijk met Ctrl + klik, en volg dezelfde werkwijze als hierboven beschreven.
  - **dupliceren:** je kan een of meerdere blokken selecteren en deze **clonen** door Ctrl + D te drukken.
- Probeer de twee beschreven technieken in te oefenen en te combineren, en je zal veel sneller blokken kunnen plaatsen: bijvoorbeeld 3 blokken selecteren, dupliceren en vervolgens de 3 nieuwe blokken tegelijk vertex snappen aan een andere blok.

Bouw een Minecraft-wereld

- Probeer ook aan de **level design** van de wereld te denken.
  - Zorg ervoor dat de speler altijd goed kan zien waar Steve zich bevindt. Een brug of grot kan wel, maar de speler mag Steve niet “kwijtraken”. Vermijd ook te diepe putten waardoor Steve volledig uit het zicht van de camera zou verdwijnen.
  - Steve mag ook niet vast komen te zitten. Voorzie dus altijd een uitweg. Denk eraan dat we hem later de mogelijkheid gaan geven om te springen, maar enkel 1 blok hoog!
  - Breng variatie in de level, zodat die visueel ook interessant wordt. Speel met verschillende soorten blokken, hoogteverschillen, bruggen, grotten, heuvels, vegetatie, …
  - Steve is 2 blokken hoog. Hou daar dus rekening mee bij het maken van bruggen of grotten.

Bouw een Minecraft-wereld

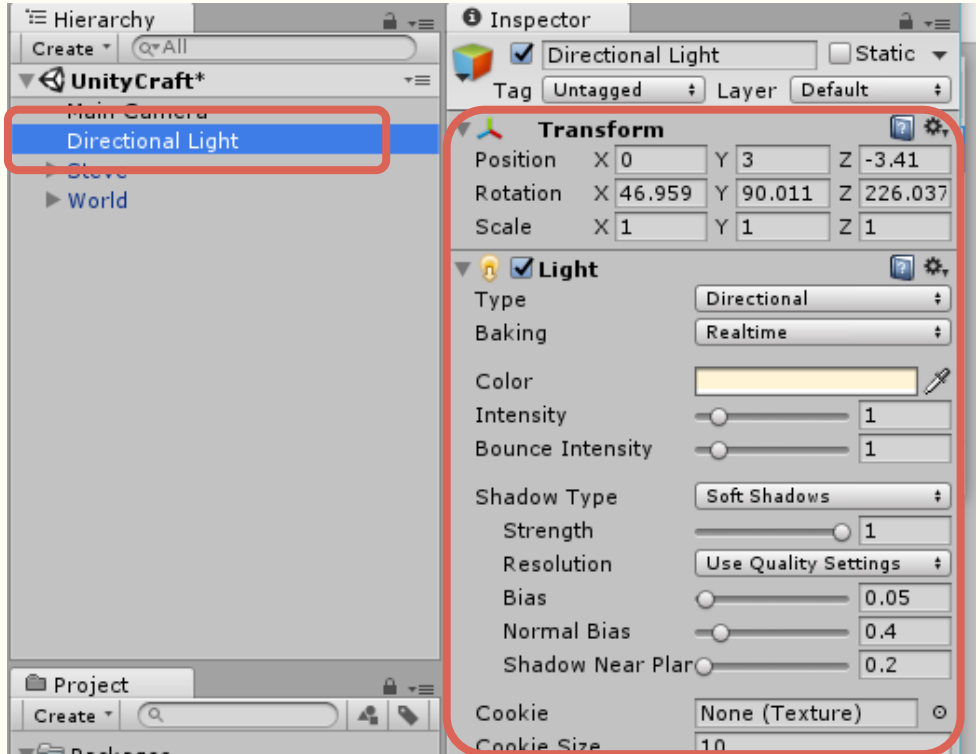
- Een beetje inspiratie!

Opslaan!

- Eens je je wereld hebt gebouwd, mag je niet vergeten om je scene op te slaan.
- Sla je scene op door op Ctrl + S te drukken.

Belichting

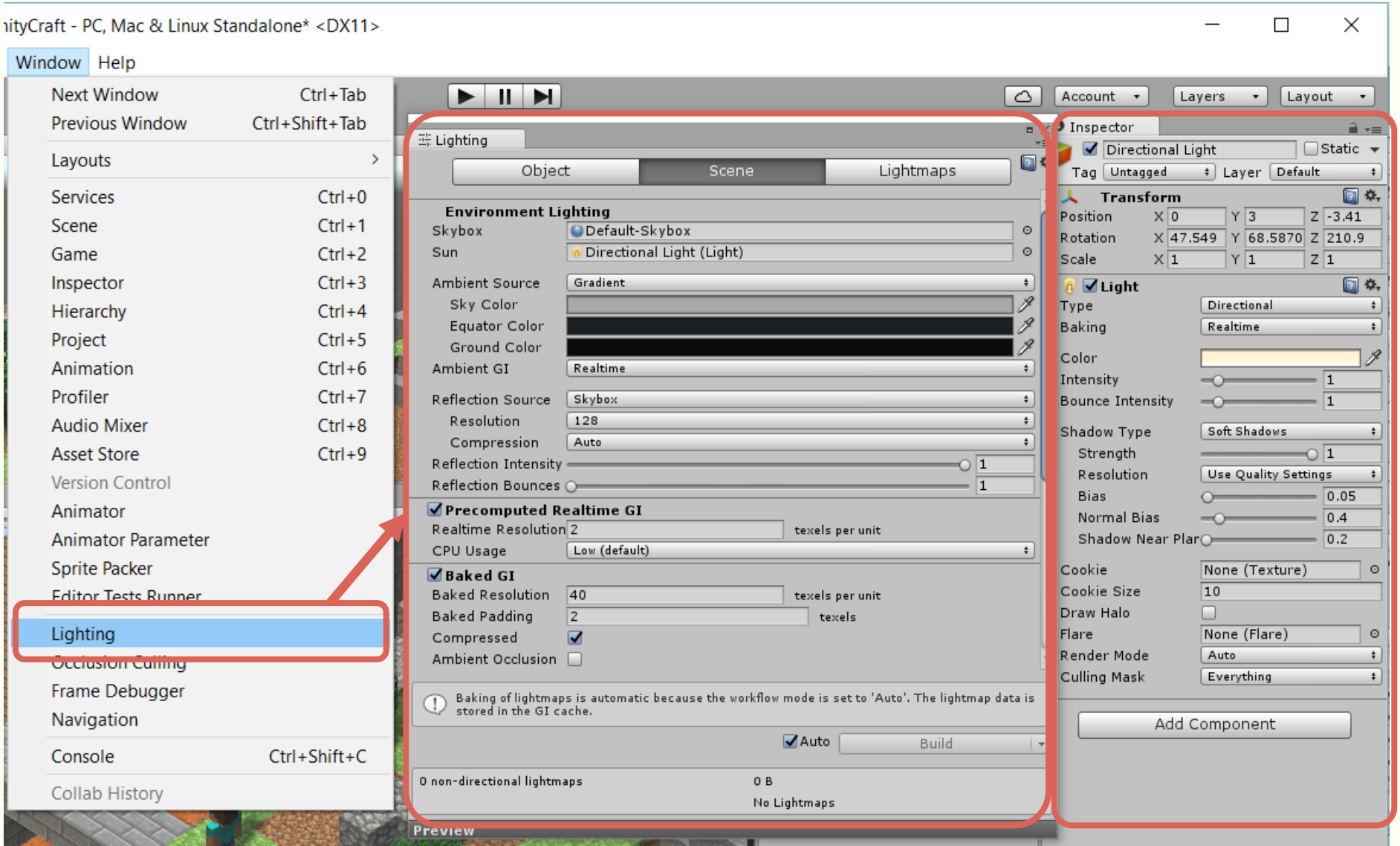
- Eens je een wereld hebt gebouwd, kan het zijn dat je belichting niet optimaal staat ingesteld.
- Speel wat met de instellingen van het Directional Light in de scene tot je een belichting verkrijgt die je zelf goed vindt.



Belichting

- Je kan daarnaast ook nog geavanceerde instellingen wijzigen voor de belichting. Het zou ons te ver leiden om dat allemaal van naderbij te bekijken, maar op de volgende slide vind je een suggestie van instellingen die een goed resultaat opleveren.
- Je vindt op deze slide instellingen terug voor het Directional Light en het Lighting dialoogvenster.
- De geavanceerde instellingen voor belichting vind je onder **Window -> Lighting**.

Belichting


ityCraft - PC, Mac & Linux Standalone* <DX11>
Window Help
Next Window Ctrl+Tab
Previous Window Ctrl+Shift+Tab
Layouts
Services Ctrl+0
Scene Ctrl+1
Game Ctrl+2
Inspector Ctrl+3
Hierarchy Ctrl+4
Project Ctrl+5
Animation Ctrl+6
Profiler Ctrl+7
Audio Mixer Ctrl+8
Asset Store Ctrl+9
Version Control
Animator
Animator Parameter
Sprite Packer
Editor Tests Runner
Lighting
Occlusion Culling
Frame Debugger
Navigation
Console Ctrl+Shift+C
Collab History
Account
Layers
Layout
Lighting
Object
Scene
Lightmaps
Environment Lighting
Skybox Default-Skybox
Sun Directional Light (Light)
Ambient Source Gradient
Sky Color
Equator Color
Ground Color
Ambient GI Realtime
Reflection Source Skybox
Resolution 128
Compression Auto
Reflection Intensity 1
Reflection Bounces 1
Precomputed Realtime GI
Realtime Resolution 2 texels per unit
CPU Usage Low (default)
Baked GI
Baked Resolution 40 texels per unit
Baked Padding 2 texels
Compressed
Ambient Occlusion
Baking of lightmaps is automatic because the workflow mode is set to 'Auto'. The lightmap data is stored in the GI cache.
Auto
Build
0 non-directional lightmaps
0 B
No Lightmaps
Preview
Inspector
Directional Light
Static
Tag Untagged
Layer Default
Transform
Position X 0 Y 3 Z -3.41
Rotation X 47.549 Y 68.5870 Z 210.9
Scale X 1 Y 1 Z 1
Light
Type Directional
Baking Realtime
Color
Intensity 1
Bounce Intensity 1
Shadow Type Soft Shadows
Strength 1
Resolution Use Quality Settings
Bias 0.05
Normal Bias 0.4
Shadow Near Plar 0.2
Cookie None (Texture)
Cookie Size 10
Draw Halo
Flare None (Flare)
Render Mode Auto
Culling Mask Everything
Add Component

Steve laten springen

- Als we Steve nu rondbewegen, merken we nog een probleempje op: hij kan niet op hoger gelegen blokken springen!
- We gaan dus terug in de code van het **Walk**-script moeten duiken om deze mogelijkheid te voorzien.
- Eerst gaan we een aantal variabelen toevoegen die de springkracht en de valsnelheid gaan bepalen.
- We voegen de publieke variabelen **JumpSpeed** en **Gravity** toe en de private variabele **UpSpeed**. Deze zijn allemaal floats.

```
public float WalkSpeed;
public float RotationSpeed;
public float JumpSpeed;
public float Gravity;

private CharacterController controller;
private Animation animations;
private Vector3 moveDirection;
private float UpSpeed;
```

Steve laten springen

- In de **Update** lus gaan we bij de bestaande code het volgende **onderaan** toevoegen:

```
if (controller.isGrounded)
{
    UpSpeed = -Gravity;
    if (Input.GetKeyDown(KeyCode.Space))
    {
        UpSpeed = JumpSpeed;
    }
}
else
{
    UpSpeed -= Gravity;
}

moveDirection.y = UpSpeed * Time.deltaTime;

controller.Move(moveDirection);
```

We kunnen enkel springen als Steve op de grond staat.

Als we op de grond staan, passen we de zwaartekracht toe.

Als we op spatiebalk drukken, wordt onze verticale snelheid gelijk aan de springsnelheid.

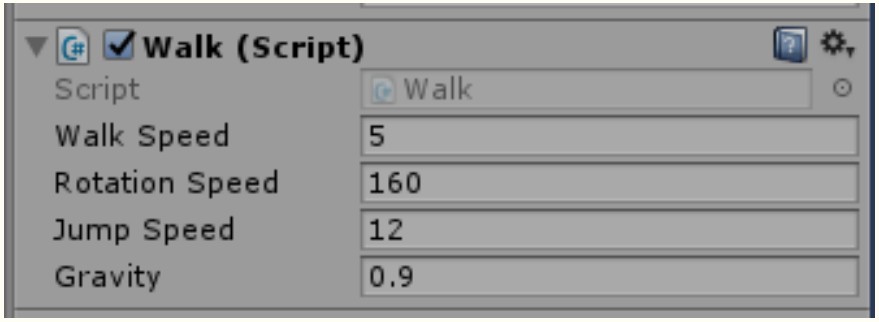
Als we niet op de grond staan, betekent dit dat we aan het springen of vallen zijn. We gaan Steve als het ware naar de grond toe trekken door de snelheid elke keer te blijven verminderen tot Steve terug op de grond komt te staan.

We passen de verticale snelheid toe op onze bewegingsvector.

Dit stukje code heb je al, zorg ervoor dat het als de laatste instructie in de update lus staat.

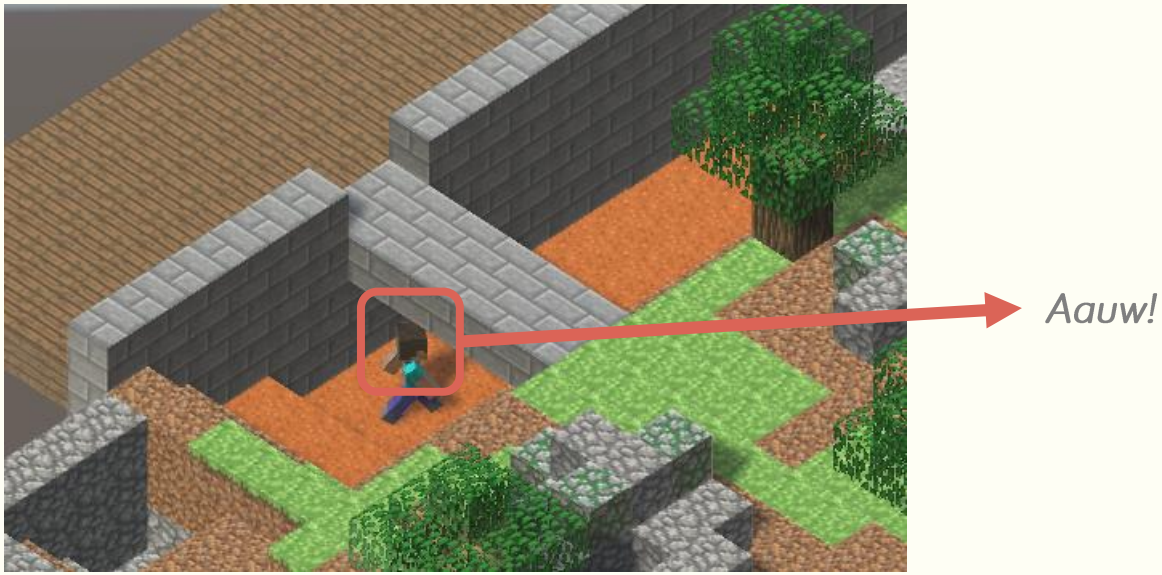
Steve laten springen

- Ten slotte kan je de waarden voor het springen in de inspector wat aanpassen naar je eigen gevoel.
- Wat goed werkt zijn onderstaande waarden.



Steve stoot zijn hoofd!

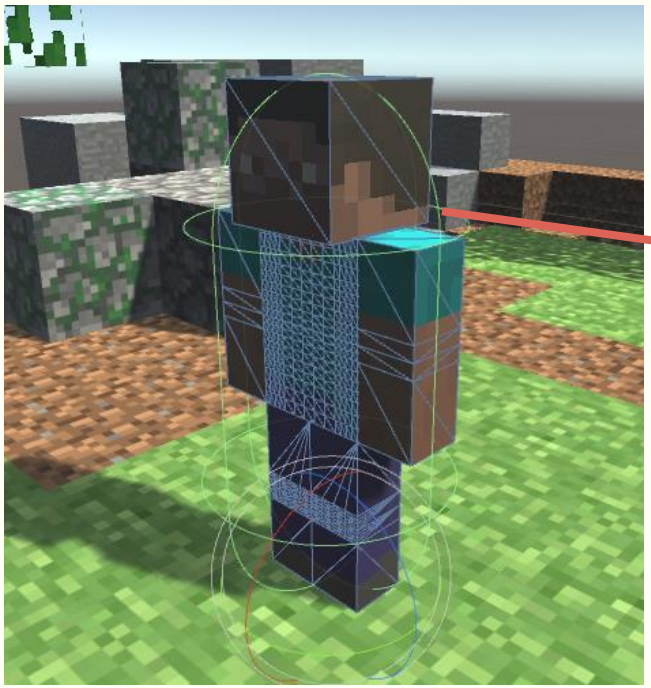
- Het is mogelijk dat Steve niet overal onder of tussen kan als je hem laat rondlopen in je wereld. Dat komt omdat de *Collision detector* van de Character Controller wat te ruim staat ingesteld.

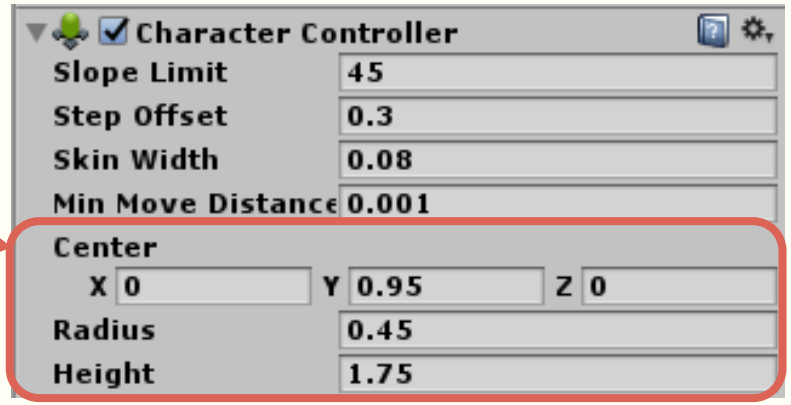


- We gaan de instellingen van de Character Controller aanpassen zodat dit niet meer gebeurt. Op naar de volgende slide!

Steve stoot zijn hoofd!

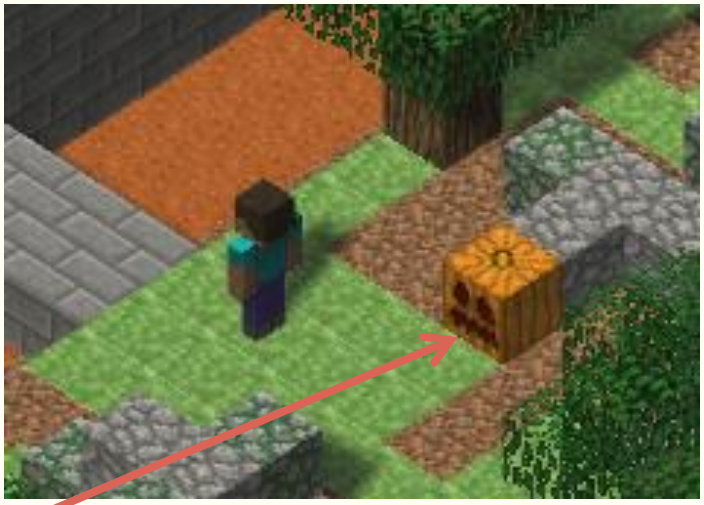
- Het komt erop neer dat we de capsule die de Character Controller rond Steve plaatst een beetje aanpassen.
- Selecteer Steve in de scene, probeer onderstaande waarden uit en kijk of Steve nu tussen een ruimte van 1 blok breed kan en door een ruimte van 2 blokken hoog.





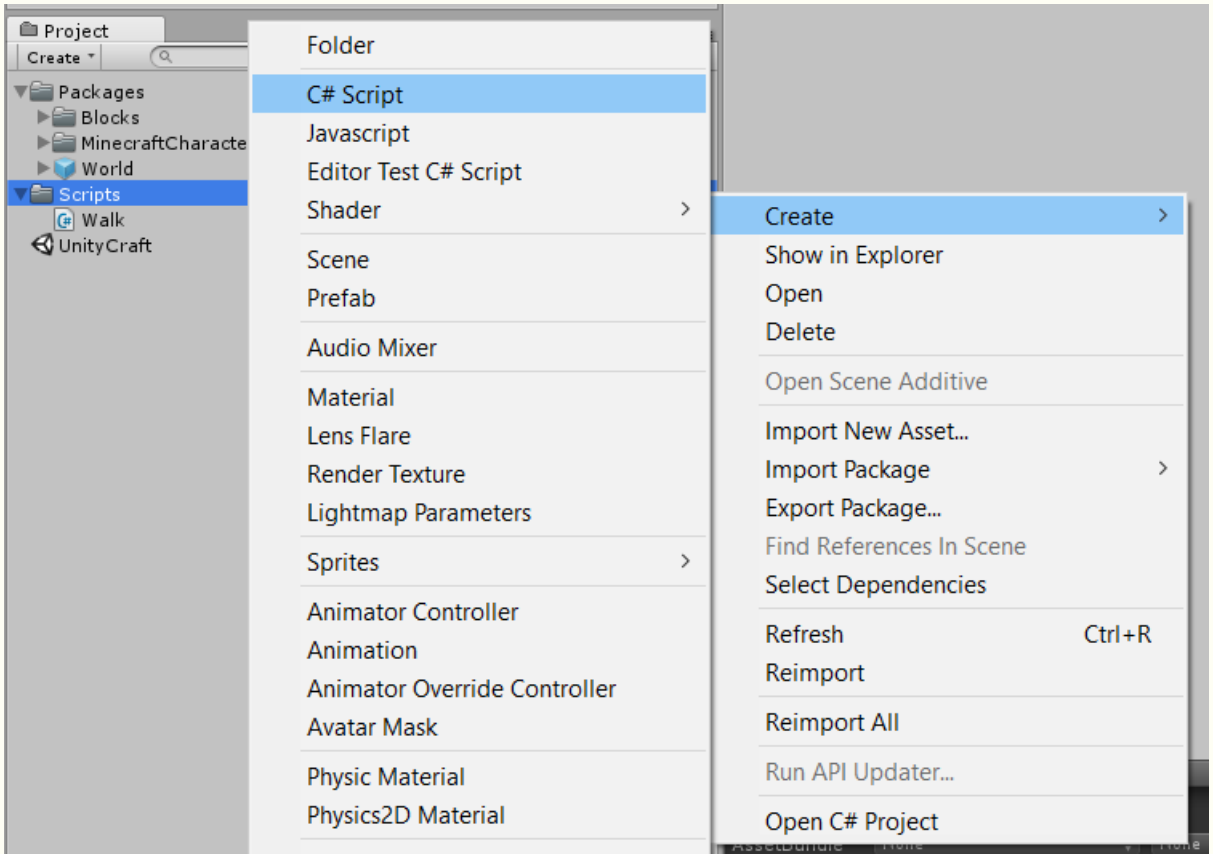
Pompoenen rapen!

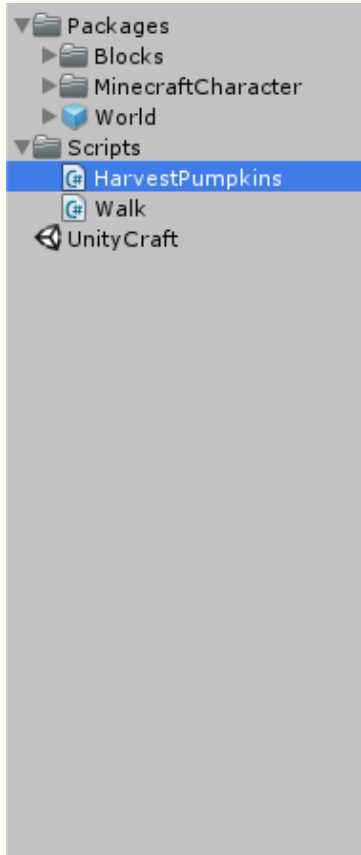
- We gaan een stapje verder en gaan Steve pompoenen laten rapen!
- Zorg er eerst voor dat je al het vorige goed begrijpt, want dit wordt weer een beetje moeilijker, en we gaan ook ietsje sneller vooruit.
- We beginnen eerst met pompoenen in onze wereld te plaatsen. Zoek de **pumpkin prefab** en plaats er enkele in je wereld.



Pompoenen rapen!

- We gaan een nieuw script maken waarin we het pompoenrapen gaan implementeren.
- Maak in het **Project** venster een nieuw script aan onder de map Scripts met de naam **HarvestPumpkins**.





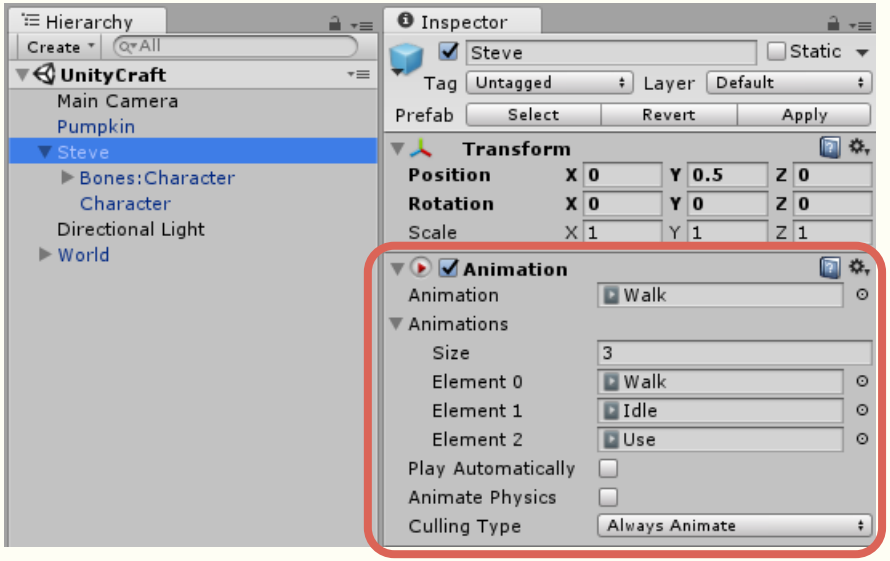
Pompoenen rapen!

- In dit script mag je zoals in het vorige script een referentie voorzien naar de animatie component.

```
public class HarvestPumpkins : MonoBehaviour {

    private Animation animations;

    void Start()
    {
        animations = GetComponent<Animation>();
    }

    void Update ()
    {

    }
}
```

Pompoenen rapen!

- Herinner je je dat Steve een animatie component heeft waarin we 3 animaties konden terugvinden?



- Er is er eentje dat we nog niet gebruikt hebben, namelijk de **Use** animatie.
- De bedoeling is nu dat we de **Use animatie afspelen** als we op de **Ctrl knop** drukken.

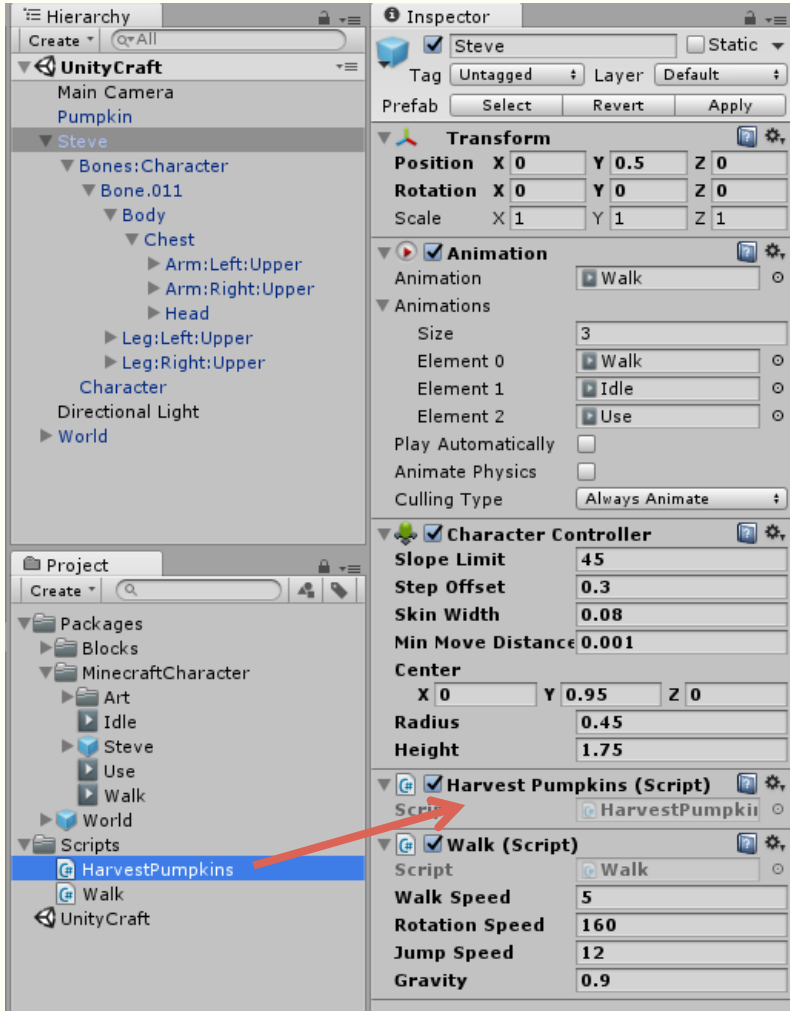
Pompoenen rapen!

- We moeten hiervoor een speciale parameter toevoegen aan de functie om de animatie af te spelen.
- We geven aan de functie mee dat alle andere huidige animaties die worden afgespeeld, worden gestopt: **Playmode.StopAll**
- Dit doen we omdat in het Walk script de Idle animatie continu wordt afgespeeld als Steve stil staat. Dit zorgt ervoor dat de Use animatie steeds overschreven wordt door de Idle. We vragen dan ook aan Unity om alle andere, dus ook de Idle, te stoppen en voorrang te verlenen aan de Use animatie.

```
void Update ()
{
    if (Input.GetKeyDown(KeyCode.LeftControl))
    {
        animations.CrossFade("Use", 0, PlayMode.StopAll);
    }
}
```

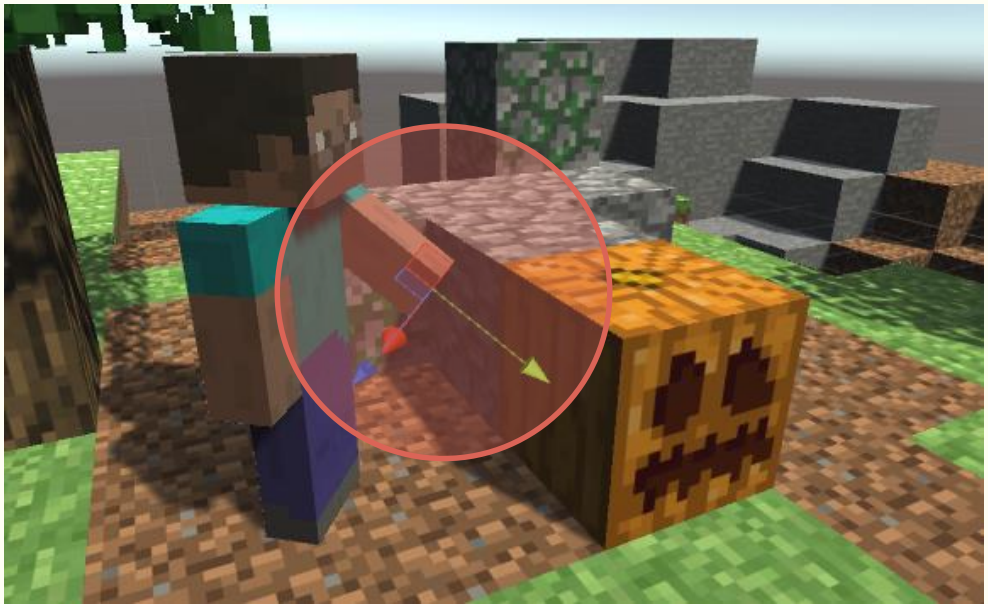
Pompoenen rapen!

- Test of de animatie afspeelt als je op Ctrl drukt.
- Daarvoor ga je eerst het nieuwe **HarvestPumpkins** script aan Steve moeten **toevoegen**.
- Steve zou zijn linkerarm moeten bewegen als je op Ctrl drukt.



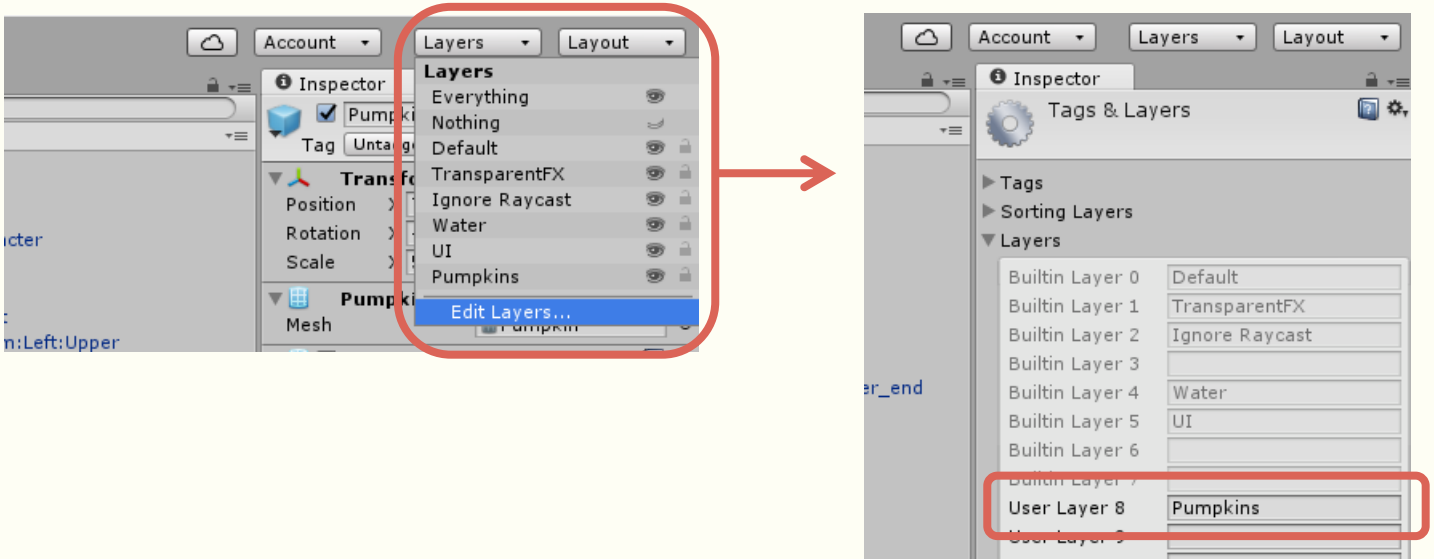
Zoeken naar pompoenen

- Nu het rapen zelf! Hoe gaan we dat precies doen? Wel, de eerste stap is dat we, wanneer we op Ctrl drukken, rondom de linkerhand kijken of er een pompoen voldoende dichtbij ligt die we kunnen oprapen.
- We gaan een nieuwe functie gebruiken: **Physics.OverlapSphere**, waarmee we rondom een bepaald punt kunnen zoeken naar een specifiek soort *GameObject*.



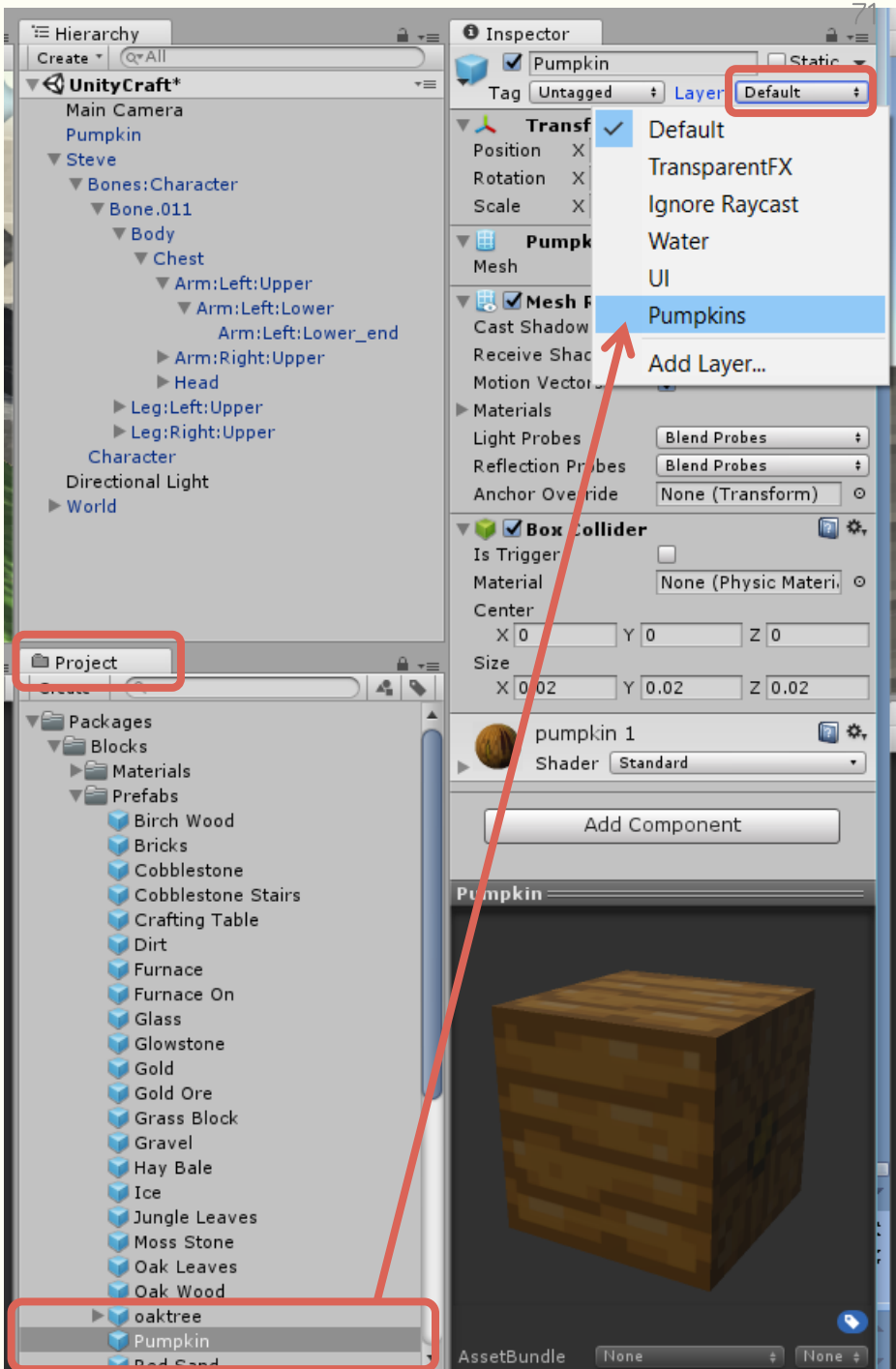
Zoeken naar pompoenen

- We willen enkel zoeken naar pompoenen, en de andere blokken negeren. Om dit te kunnen doen moeten we de pompoenen in een aparte *Layer* stoppen.
- We gaan een nieuwe Layer moeten aanmaken hiervoor. Dit doe je door rechtsboven in Unity **Layers** -> **Edit Layers** te kiezen.
- Op de eerst beschikbare plaats in de Layer manager voeg je dan een nieuwe Layer **Pumpkins** toe.



Zoeken naar pompoenen

- Vervolgens selecteer je de **Pumpkin prefab** in het Project view en verander je de layer hiervan in **Pumpkins**. Als je dit doet, zouden alle pompoenen in je scene ook in de Pumpkins layer moeten terechtkomen. Kijk dit even na!



Zoeken naar pompoenen

- We voegen nu een stukje code toe aan het **HarvestPumpkins** script.
- We hebben eerst een aantal publieke variabelen nodig.
- **LeftHand**: een referentie naar de linkerhand van Steve

  Deze variable is van het type *Transform*. Herinner je je dat Transform een component is die de positie, rotatie en schaal van een GameObject representeert? Wel, we willen de positie van de linkerhand weten, dus vragen we de Transform component van die hand op.
- **PumpkinsLayer**: de Layer waarin de pompoenen zitten Deze variable is van het type *LayerMask*, wat een specifiek type is om Layers aan te duiden in Unity.

```
public class HarvestPumpkins : MonoBehaviour {

    public Transform LeftHand;
    public LayerMask PumpkinsLayer;
```

Zoeken naar pompoenen

- Vervolgens breiden we het script in de Update lus uit met het volgende:

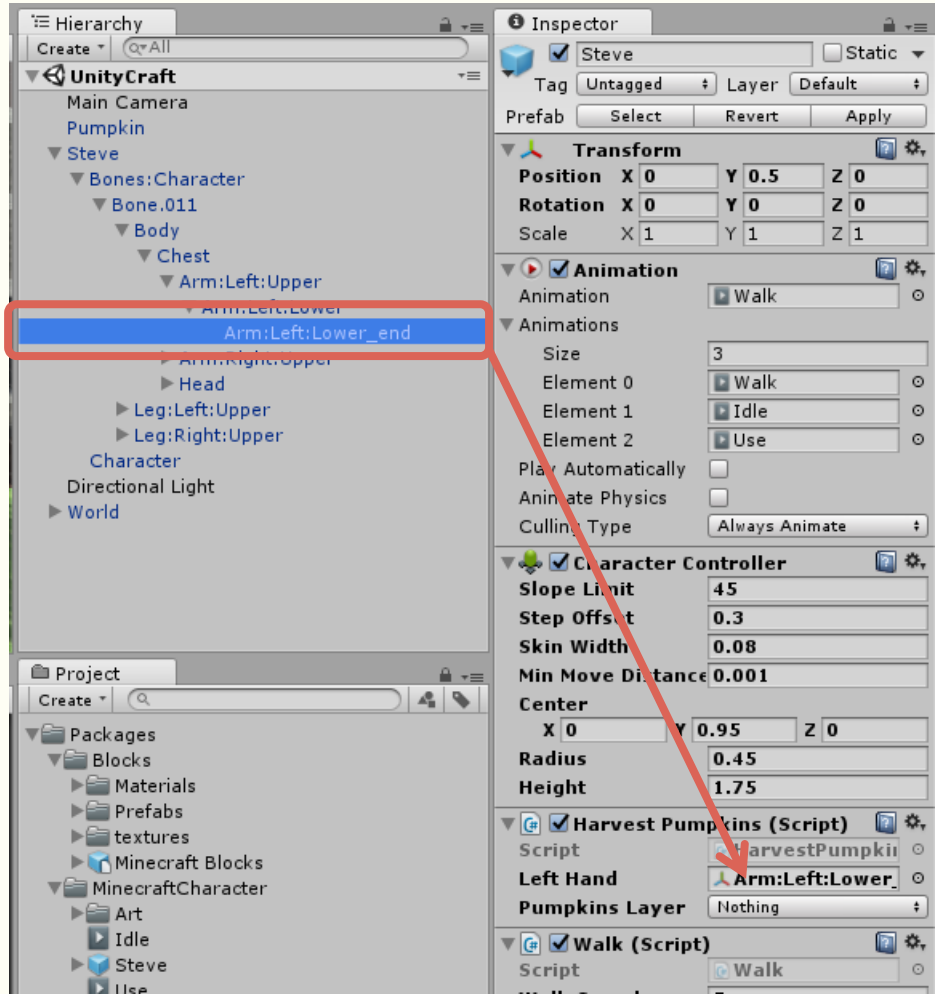
```
void Update ()
{
    if (Input.GetKeyDown(KeyCode.LeftControl))
        {
            animations.CrossFade("Use", 0, PlayMode.StopAll);
            Collider[] pumpkins = Physics.OverlapSphere(
                                                LeftHand.position, 0.75f, PumpkinsLayer);
            if (pumpkins.Length > 0)
            {
                print(pumpkins[0].name);
            }
        }
}
```

- Neem even de tijd om deze code te proberen begrijpen. We gaan hier in een straal van 0.75 rond de *LeftHand* zoeken naar alle GameObjects die zich in de *PumpkinsLayer* bevinden. Als we er vinden (de lengte van de lijst van *pumpkins* is groter dan nul), dan printen we de naam van de eerste in de lijst af in de Console.

Zoeken naar pompoenen

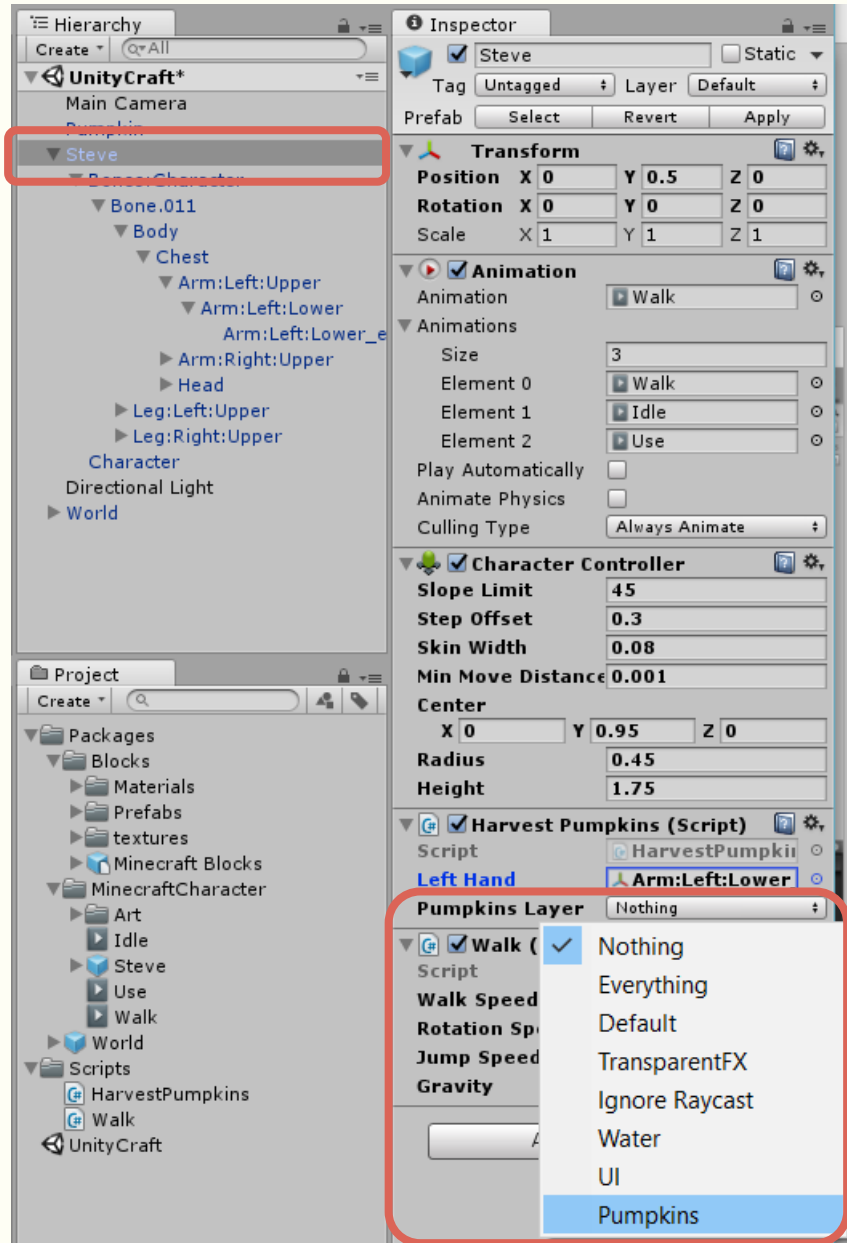
- We gaan nu terug naar de Unity Editor en de publieke variabelen die we hebben toegevoegd aan het HarvestPumpkins script correct invullen.

Eerst moet je op zoek gaan naar het **linkerhand van Steve**. Deze vind je door de hierarchy van Steve te openen in het Hierarchy paneel, en daar op zoek te gaan naar:
Arm:Left:Lower_end
Dit mag je slepen naar het **Left Hand** slot in het HarvestPumpkins script van Steve.



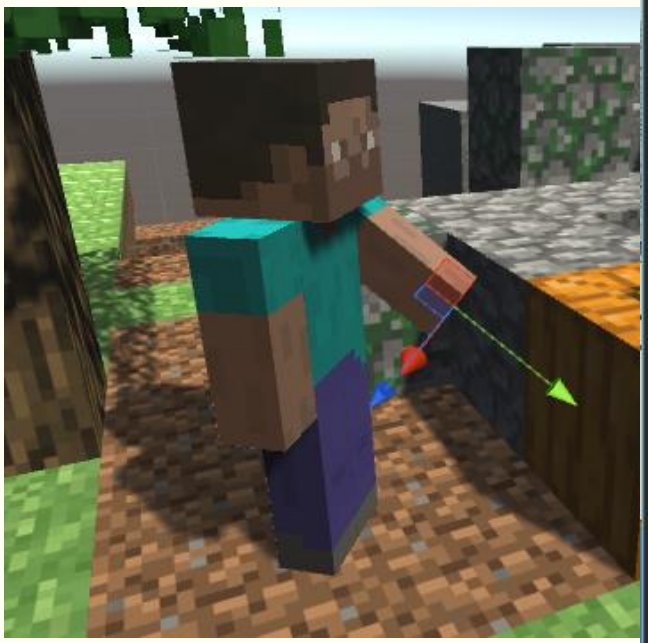
Zoeken naar pompoenen

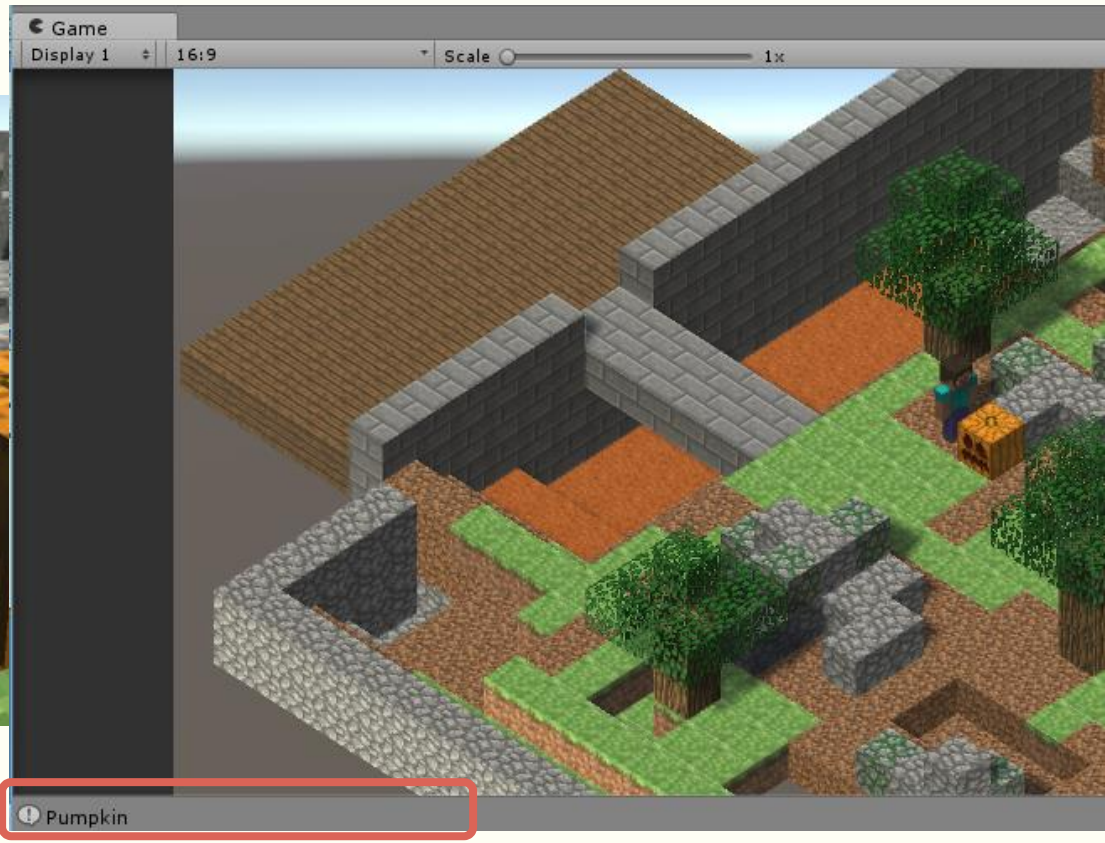
- Vervolgens vullen we de correcte Layer in waar Unity moet zoeken naar pompoenen, namelijk de **Pumpkins** layer.



Testen!

- **Test nu het script** door dichtbij een pompoen te gaan staan en op Ctrl te drukken.
- Kijk of er in de **Console** onderaan het scherm de naam van het GameObject staat dat door je script gevonden wordt, namelijk *Pumpkin*.



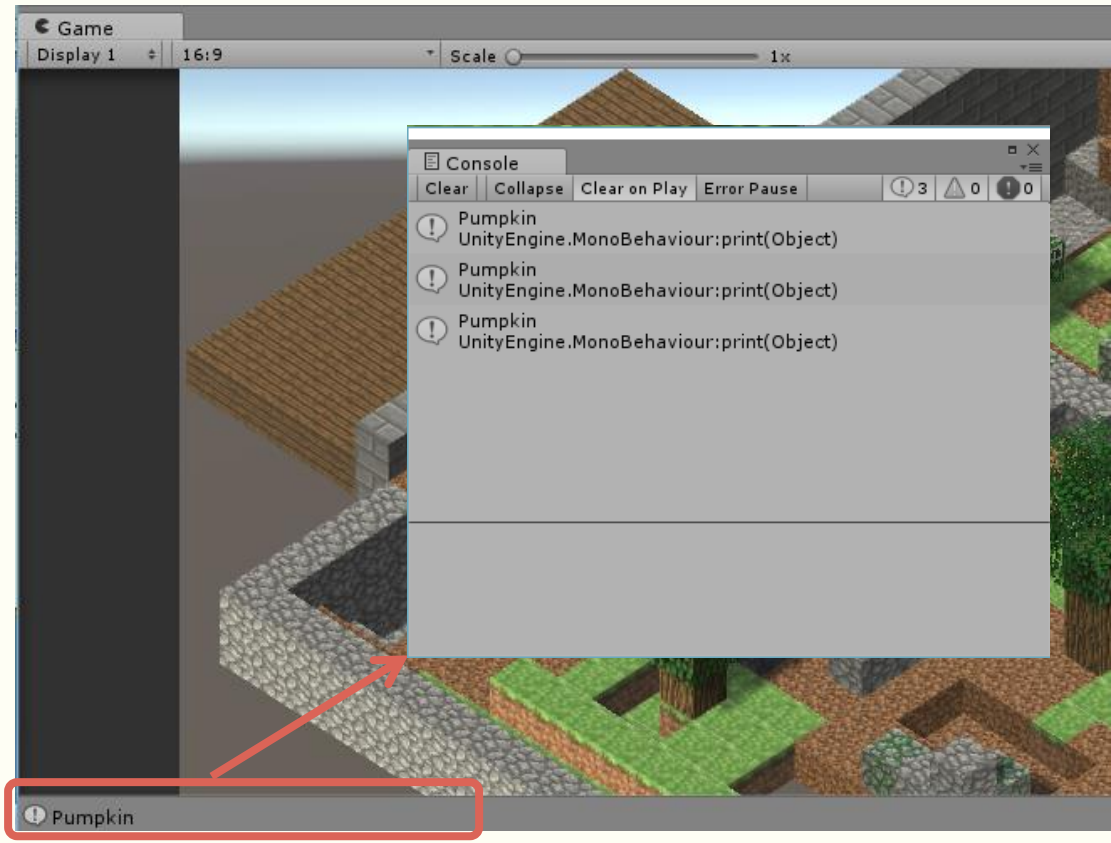


Opslaan!

- Oef, dat was toch niet zo gemakkelijk!
- Sla je scene op door op Ctrl + S te drukken.

Testen!

- Printen naar de **console** is een **snelle en eenvoudige manier om iets te testen** in Unity.
- In dit geval werkt je script als er *Pumpkin* verschijnt in de console **enkel en alleen** als je dicht genoeg bij een pompoen staat en Ctrl drukt
- Door op de console te klikken krijg je een apart venster waarin je een beter overzicht krijgt over de logs in de console

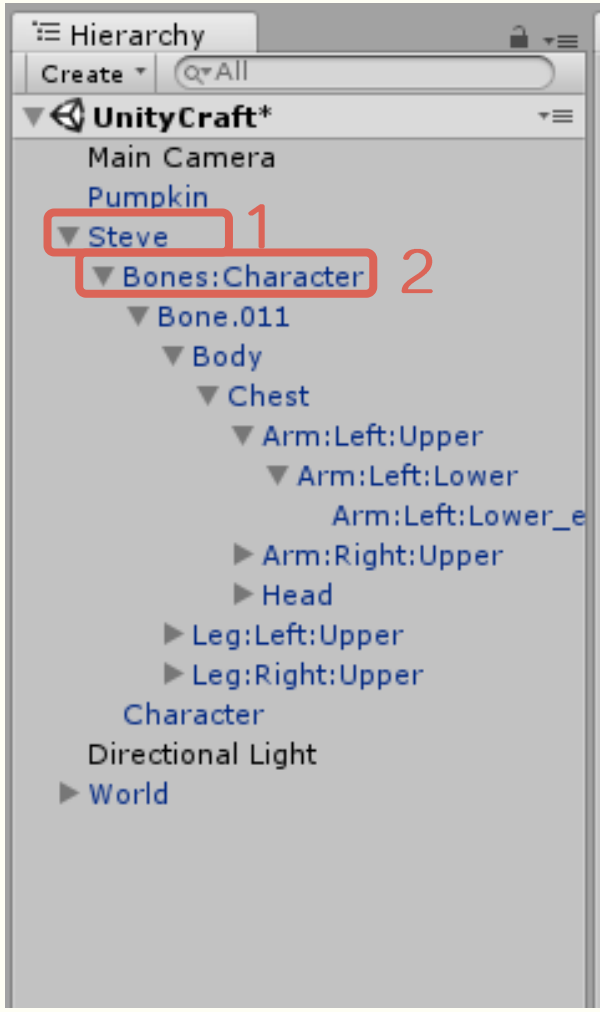


Pompoenen oprapen

- Nu we pompoenen kunnen detecteren, kunnen we ze ook oprapen!
- We gaan **print(pumpkins[0].name);** vervangen door de code die de pompoen vasthangt aan de hand van Steve.
- Om de pompoen aan het hand van Steve vast te hangen moeten we er dus voor zorgen dat de pompoen het hand van Steve volgt. De eenvoudigste manier om dit te doen is de pompoen in de hierarchie onder het linkerhand van Steve te verplaatsen. Dat klinkt moeilijk, maar is het eigenlijk niet. We gaan even dieper in op het concept van ouder en kind in het hierarchie paneel van Unity.

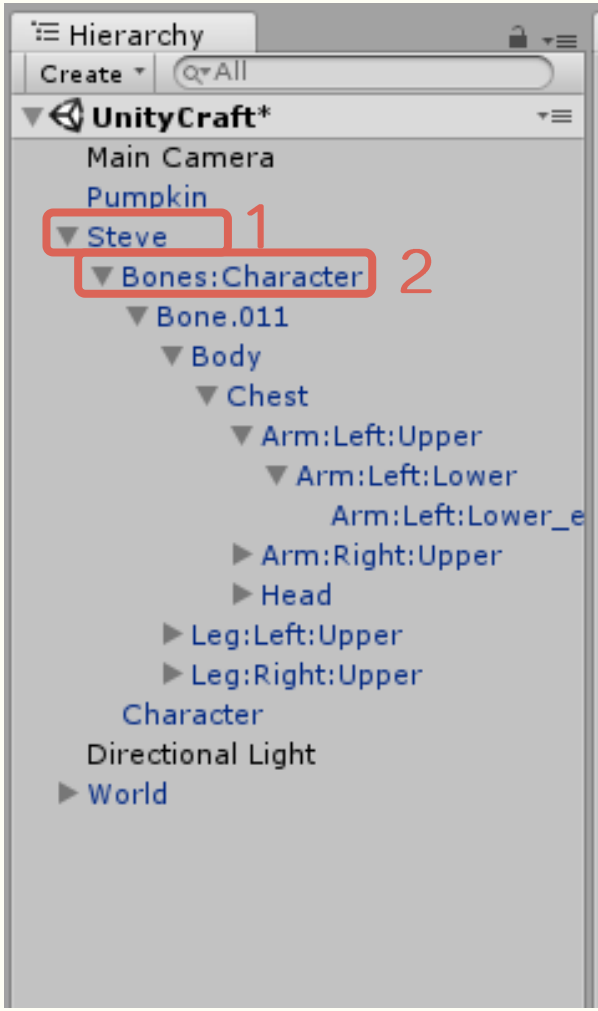
Eerst een woordje over ouders en kinderen!

- Als je het hierarchy paneel bekijkt en je vouwt wat elementen open van Steve, zie je dat hij ook is opgebouwd uit een complexe hierarchie.
- Het element **Steve** (1) zit het hoogste in de hierarchie. Daar direct onder vind je **Bones:Character** (2) terug. Dit noemt men een ***child of kind*** van Steve, of omgekeerd, Steve is een ***parent*** **of ouder** van Bones:Character
- Als een *ouder* beweegt, roteert of verandert van schaal, zullen diens *kinderen* volgen, maar ook de kinderen van die kinderen, enz.



Eerst een woordje over ouders en kinderen!

- Om ervoor te zorgen dat de pompoen het linkerhand van Steve zal volgen moeten we de pompoen dus als een kind maken van Steve's linkerhand, of anders gezegd, Steve's linkerhand wordt een ouder van de pompoen.



Pompoenen oprapen

- We verkleinen de pompoen ook een beetje, om esthetische redenen. Dit is optioneel, je mag het ook zo laten als je dat wil.
- Tenslotte moeten we er ook voor zorgen dat de pompoen die opgeraapt wordt niet meer in rekening gebracht wordt bij het zoeken naar nieuwe pompoenen. We moeten de opgeraapte pompoen dus verwijderen uit de *Pumpkins Layer.*
- Een hele boterham dus! De volgende slide toont de code. Probeer het zo goed mogelijk te begrijpen en test het script door een pompoen te proberen rapen.

Pompoenen oprapen

```
Collider[] pumpkins = Physics.OverlapSphere(LeftHand.position, 0.75f, PumpkinsLayer);
if (pumpkins.Length > 0)
{
    //We verkleinen de pompoen eerst tot de helft van zijn oorspronkelijke grootte.
    Vector3 pumpkinScale = pumpkins[0].transform.localScale;
    pumpkinScale = pumpkinScale * 0.5f;
    pumpkins[0].transform.localScale = pumpkinScale;

    //We maken de linker hand van Steve als ouder van de pompoen.
    //Op deze manier zal de pompoen de hand blijven volgen.
    pumpkins[0].transform.parent = LeftHand;

    //De pompoen moet vervolgens op dezelfde positie als de hand
    //geplaatst worden. We gebruiken 'localPosition' wat de positie
    //is ten opzichte van de ouder/parent, in dit geval de linker hand.
    //Als we (0, 0, 0) kiezen als localPosition, wordt de pompoen dus
    //op de linker hand zelf geplaatst.
    pumpkins[0].transform.localPosition = new Vector3(0, 0, 0);

    //We verwijderen de pompoen uit de layer waarin we zoeken naar pompoenen die
    //we kunnen oprapen, en zetten de layer op 0, wat overeenkomt met de Default layer.
    pumpkins[0].gameObject.layer = 0;
}
```

*Zoals je kan zien werken we hier steeds met **pumpkin[0]**, de eerste pompoen de we vinden. Het kan zijn dat we er meerdere vinden, maar aangezien Steve maar 1 pompoen tegelijk kan dragen nemen we enkel de eerste uit de lijst van gevonden pompoenen.*

Pompoenen oprapen

- Als je een pompoen probeert te rapen zou deze nu zoals onderstaande screenshot aan de hand van Steve moeten hangen.

Pompoenen oprapen

- Als je wat meer uitvoerig test, zal je merken dat wanneer Steve een pompoen heeft opgeraapt ook nog andere pompoenen kan oprapen. Dat is niet de bedoeling, dus moeten we iets voorzien in ons script zodat dit niet kan gebeuren.
- Probeer dit eerst even zelf!

Pompoenen oprapen

- We voegen hiervoor eerst een private variabele **hasPumpkin** toe die bijhoudt of we een pompoen hebben opgeraapt

```
public class HarvestPumpkins : MonoBehaviour {

    public Transform LeftHand;
    public LayerMask PumpkinsLayer;

    private Animation animations;
    private bool hasPumpkin = false;
```

- Deze variable is van het type **bool**, welke twee waarden kan aannemen: **true** of **false**
- We geven deze variable als beginwaarde **false**: we hebben **nog geen pompoen opgeraapt**.
- We zetten deze op **true** van zodra we een **pompoen oprapen**

Pompoenen oprapen

- We zetten de variable op **true** als we een pompoen oprapen. Ook kijken we eerst of we nog geen pompoen hebben opgeraapt voordat we er een trachten te rapen

```
if (Input.GetKeyDown(KeyCode.LeftControl))
{
    animations.CrossFade("Use", 0, PlayMode.StopAll);

    if (!hasPumpkin)
    {
        Collider[] pumpkins = Physics.OverlapSphere(
                                    LeftHand.position, 0.75f, PumpkinsLayer);
        if (pumpkins.Length > 0)
        {
            Vector3 pumpkinScale = pumpkins[0].transform.localScale;
            pumpkinScale = pumpkinScale * 0.5f;
            pumpkins[0].transform.localScale = pumpkinScale;
            pumpkins[0].transform.parent = LeftHand;
            pumpkins[0].transform.localPosition = new Vector3(0, 0, 0);
            pumpkins[0].gameObject.layer = 0;

            hasPumpkin = true;
        }
    }
}
```

! -> betekent 'niet', dus dit lees je als 'als hasPumpkin false is', of met andere woorden: als we nog geen pompoen hebben opgeraapt

Pompoenen gooien!

- Eens we een pompoen hebben opgeraapt willen we die in onze opslagruimte gooien.
- Maak hiervoor eerst een zone in je level waar je de pompoenen kan ingooien. Voorzie voldoende hoge muren, van minimum 2 blokken hoog.

Pompoenen gooien!

- Ga terug naar het **HarvestPumpkins** script en voeg een nieuwe private variabele toe, **pickedUpPumpkin** genaamd.

```
public class HarvestPumpkins : MonoBehaviour {

    public Transform LeftHand;
    public LayerMask PumpkinsLayer;

    private Animation animations;
    private bool hasPumpkin = false;
    private GameObject pickedUpPumpkin;
```

- Deze variabele is van het type **GameObject,** wat dus het algemene type is dat gebruikt wordt voor een Unity objecten. Via het GameObject kan je makkelijk aan al de overige componenten, zoals Transform, Colliders, enz.

Pompoenen gooien!

- We gaan deze variabele gebruiken om een referentie bij te houden naar de opgeraapte pompoen:

```
if (!hasPumpkin)
{
    Collider[] pumpkins = Physics.OverlapSphere(
                                LeftHand.position, 0.75f, PumpkinsLayer);
    if (pumpkins.Length > 0)
    {
        Vector3 pumpkinScale = pumpkins[0].transform.localScale;
        pumpkinScale = pumpkinScale * 0.5f;
        pumpkins[0].transform.localScale = pumpkinScale;
        pumpkins[0].transform.parent = LeftHand;
        pumpkins[0].transform.localPosition = new Vector3(0, 0, 0);
        pumpkins[0].gameObject.layer = 0;

        hasPumpkin = true;
        pickedUpPumpkin = pumpkins[0].gameObject;
    }
}
```

Pompoenen gooien!

- En tenslotte gooien we de pompoen op de grond als we Ctrl drukken!
- Hiervoor moeten we er eerst voor zorgen dat de pompoen een realistisch gedrag vertoont als deze gegooid wordt, zoals botsen op de muren en grond, vallen en rollen, enz.
- De **Rigidbody** component zorgt hiervoor. Als we deze toevoegen aan eender welk GameObject in Unity, gecombineerd met een *Collider* om de botsingen te detecteren, verkrijgen we een realistisch fysisch gedrag van dit object.
- We mogen ook niet vergeten dat de pompoen het hand van Steve niet meer mag volgen als deze wordt weggegooid. Daarvoor moeten we de ouder van de pompoen op *null* zetten, wat betekent dat de pompoen geen ouder meer boven zich heeft en op zichzelf zal bewegen.

Pompoenen gooien!

```
if (!hasPumpkin)
{
    Collider[] pumpkins = Physics.OverlapSphere(
                                LeftHand.position, 0.75f, PumpkinsLayer);
    if (pumpkins.Length > 0)
    {
        Vector3 pumpkinScale = pumpkins[0].transform.localScale;
        pumpkinScale = pumpkinScale * 0.5f;
        pumpkins[0].transform.localScale = pumpkinScale;
        pumpkins[0].transform.parent = LeftHand;
        pumpkins[0].transform.localPosition = new Vector3(0, 0, 0);
        pumpkins[0].gameObject.layer = 0;

        hasPumpkin = true;
        pickedUpPumpkin = pumpkins[0].gameObject;
    }
}
else   Als we een pompoen in de hand hebben, gooien we die
{
    pickedUpPumpkin.transform.parent = null; Linkerhand als ouder verwijderen
    pickedUpPumpkin.AddComponent<Rigidbody>(); Rigidbody component toevoegen aan de pompoen
    hasPumpkin = false; We hebben geen pompoen meer vast, dus zetten hasPumpkin op false
}
```

Is dat gooien?!

- Als je het script nu test zal je zien dat Steve niet echt pompoenen gooit, maar ze gewoon laat vallen.
- Dat is normaal, weet je waarom?

Is dat gooien?!

- We geven de Rigidbody geen beginsnelheid! Als je een Rigidbody aanmaakt heeft die als startsnelheid 0.
- De enige versnelling die de Rigidbody krijgt is die veroorzaakt door de zwaartekracht: neerwaarts dus.
- We moeten de Rigidbody van de pompoen dus een extra startsnelheid meegeven in de richting waar Steve kijkt.
- Deze richting kan je opvragen via de transform component, meerbepaald door **transform.forward**
- Dit geeft ons een eenheidsvector, wat betekent een vector met grootte 1. We willen natuurlijk verder gooien, dus moeten we deze vector ook nog vermenigvuldigen met een getal.
- Laten we hiervoor een publieke variabele aanmaken:

```
public class HarvestPumpkins : MonoBehaviour
{

    public Transform LeftHand;
    public LayerMask PumpkinsLayer;
    public float ThrowSpeed;
```

Dát is gooien!

- Het uiteindelijke script wordt dan:

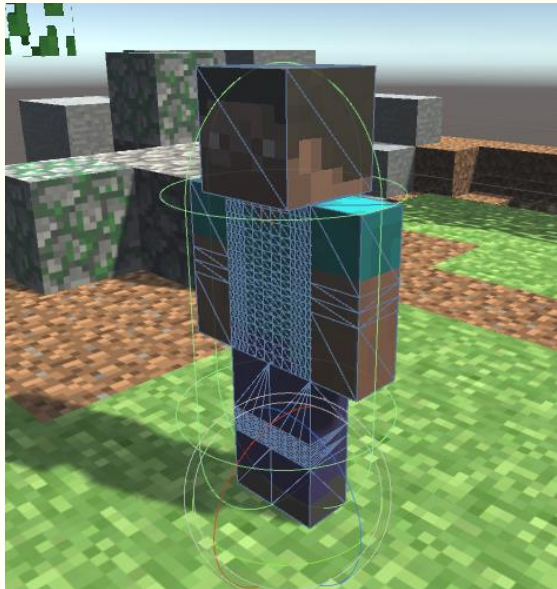
```
if (!hasPumpkin)
{
    Collider[] pumpkins = Physics.OverlapSphere(
                                            LeftHand.position, 0.75f, PumpkinsLayer);
    if (pumpkins.Length > 0)
    {
        Vector3 pumpkinScale = pumpkins[0].transform.localScale;
        pumpkinScale = pumpkinScale * 0.5f;
        pumpkins[0].transform.localScale = pumpkinScale;
        pumpkins[0].transform.parent = LeftHand;
        pumpkins[0].transform.localPosition = new Vector3(0, 0, 0);
        pumpkins[0].gameObject.layer = 0;

        hasPumpkin = true;
        pickedUpPumpkin = pumpkins[0].gameObject;
    }
}
else
{
    pickedUpPumpkin.transform.parent = null;
    Rigidbody rb = pickedUpPumpkin.AddComponent<Rigidbody>();
    rb.velocity = transform.forward * ThrowSpeed;
    hasPumpkin = false;
}
```

Volgorde is hier belangrijk! Anders wordt de snelheid van de linkerhand ook in rekening gebracht en kan dit onverwachte resultaten opleveren.

- Vergeet niet om **ThrowSpeed** ook een waarde te geven in de Inspector!

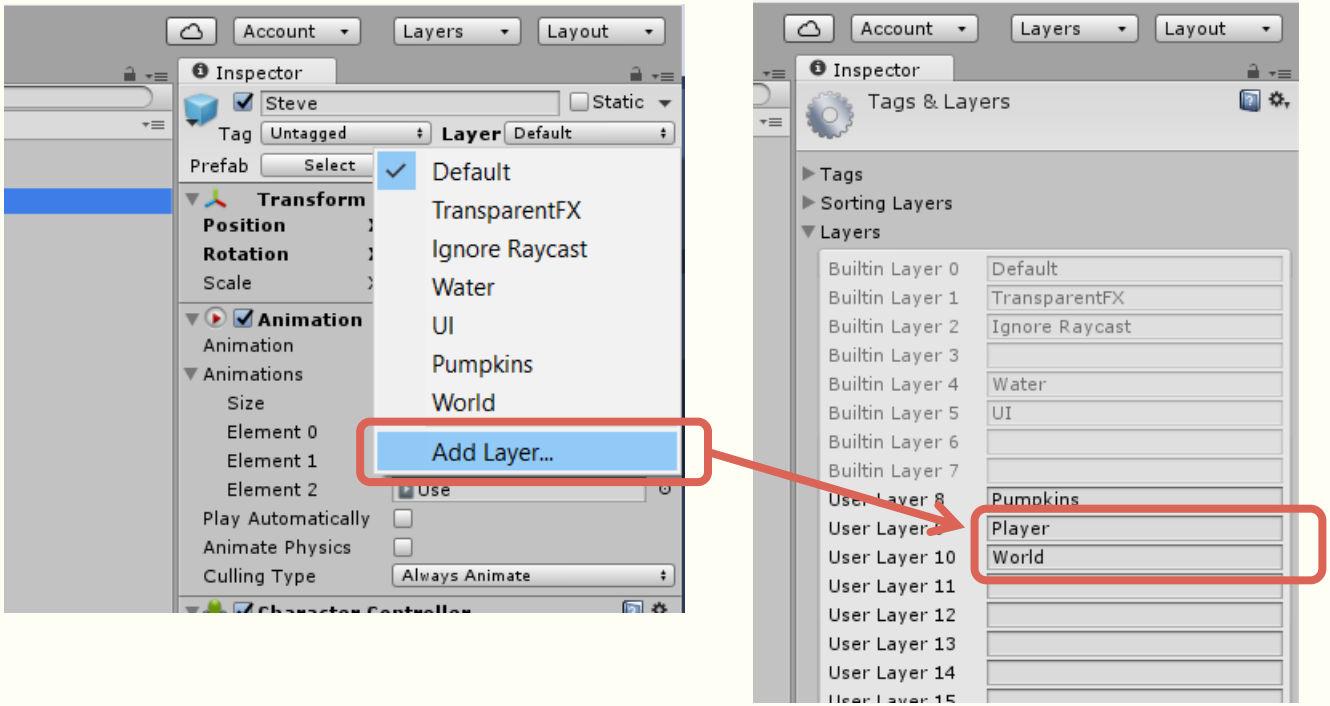
Wat botst met wat?

- Steve heeft een *Collider*, ingebouwd in de *Character Controller* om botsingen te detecteren. De capsule, weet je nog?
- Het probleem is nu dat als je een pompoen weggooit, deze eerst met deze capsule kan botsen alvorens deze vertrekt volgens de snelheid dat je hebt toegewezen in je code.
- De pompoen vertrekt immers vanaf de hand van Steve, die zich in deze *Capsule Collider* bevindt.
- Dit zorgt ervoor dat de pompoen zijn richting kan afwijken bij het gooien.
- We moeten er dus voor zorgen dat de pompoen niet botst met de capsule als deze wordt weggegooid.
- Hiervoor moeten we een aantal kleine wijzigingen maken in de scene.



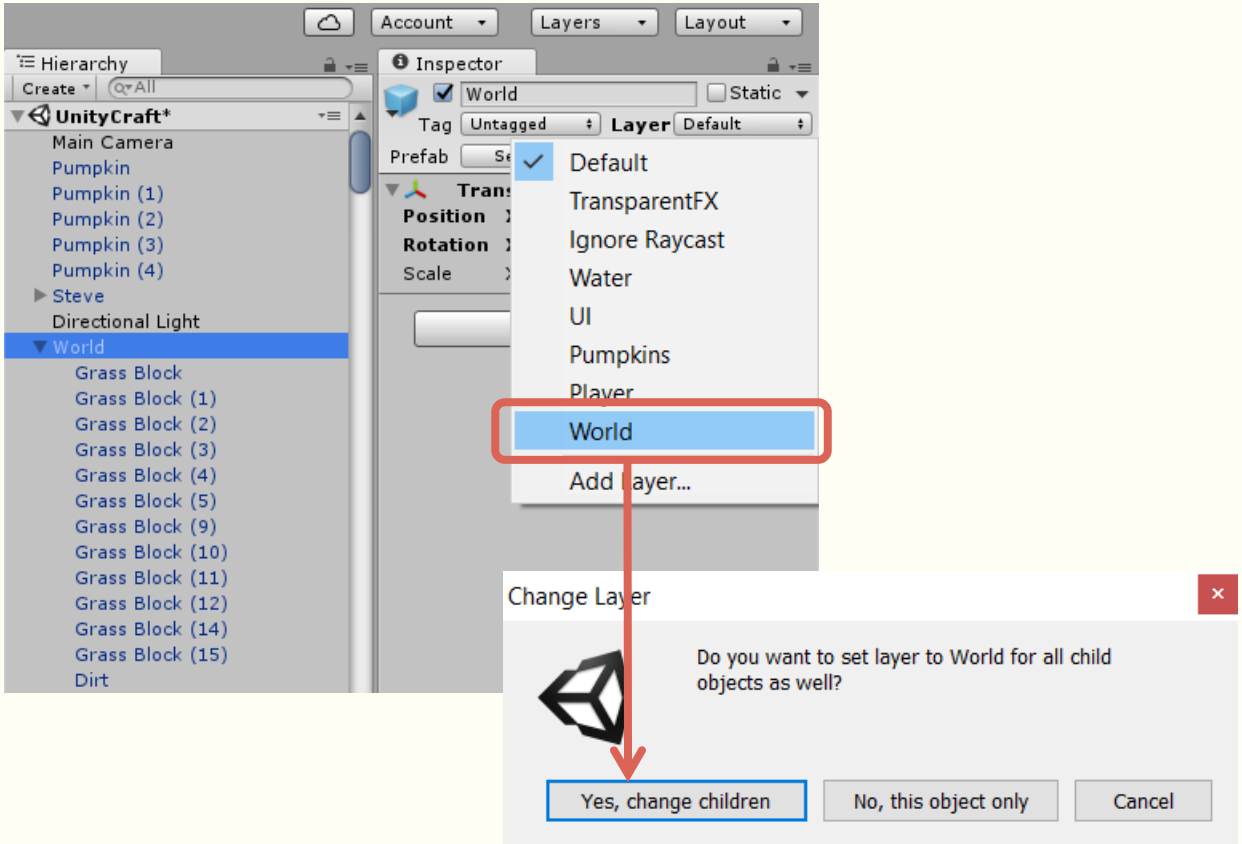
Wat botst met wat?

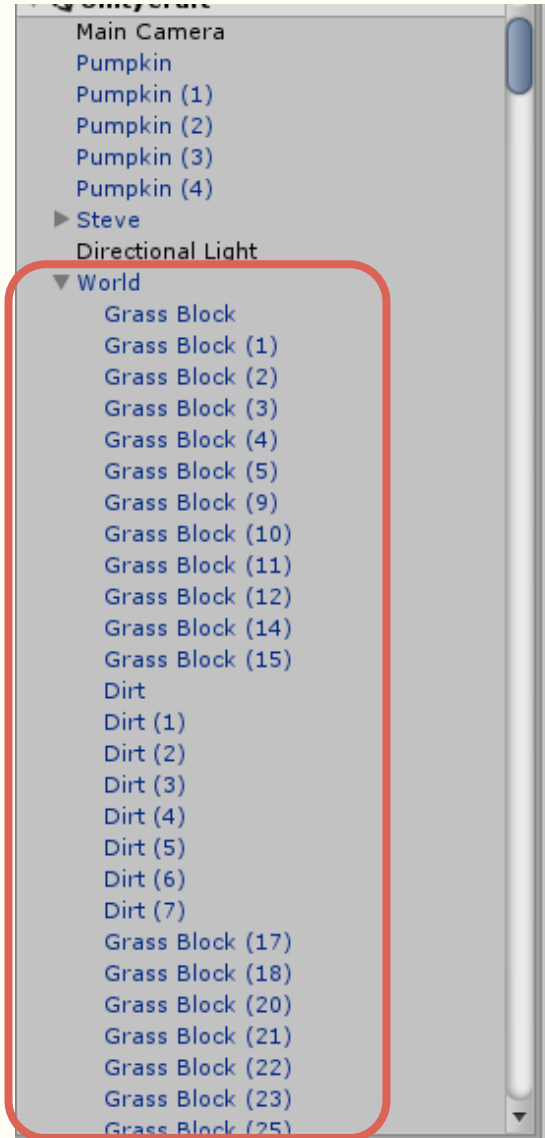
- Maak twee nieuwe *Layers* aan, **Player** en **World** genaamd.



Wat botst met wat?

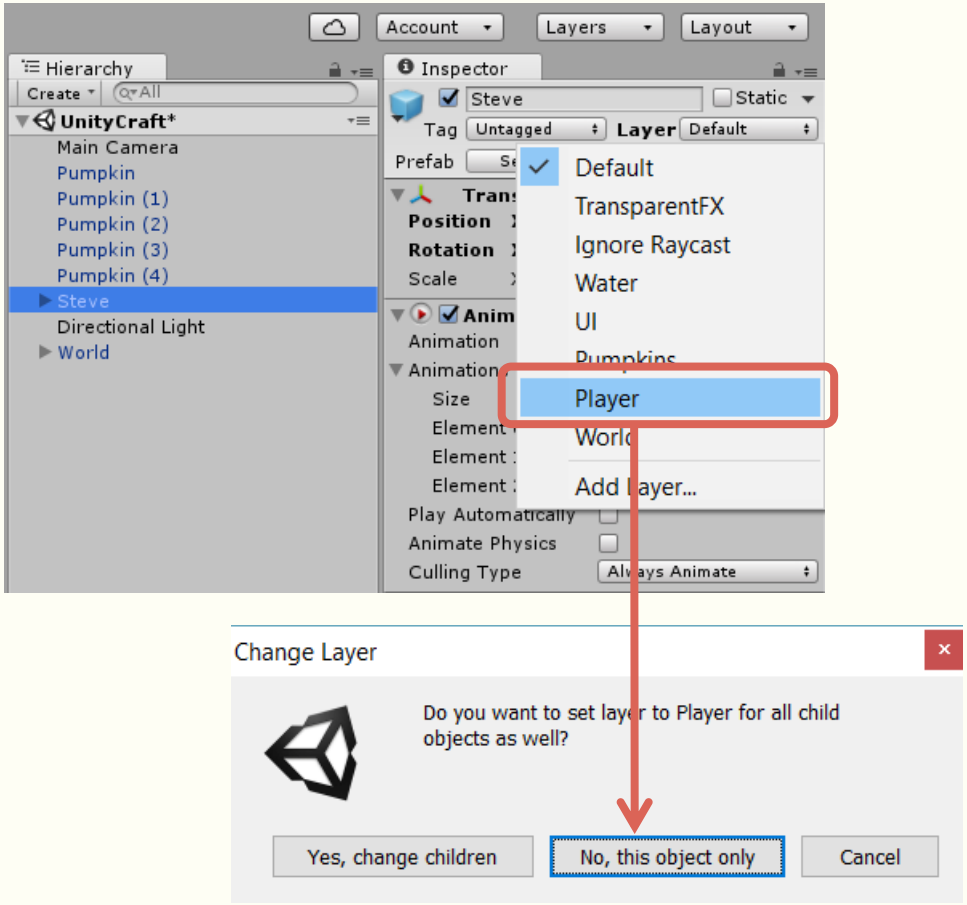
- Zorg ervoor dat alle blokkenvan je wereld, **behalve de pompoenen en Steve**, onder één *ouder* zitten
- Je kan de ouder bvb **World** noemen
- Stop **World en al zijn kinderen** in de **World layer**





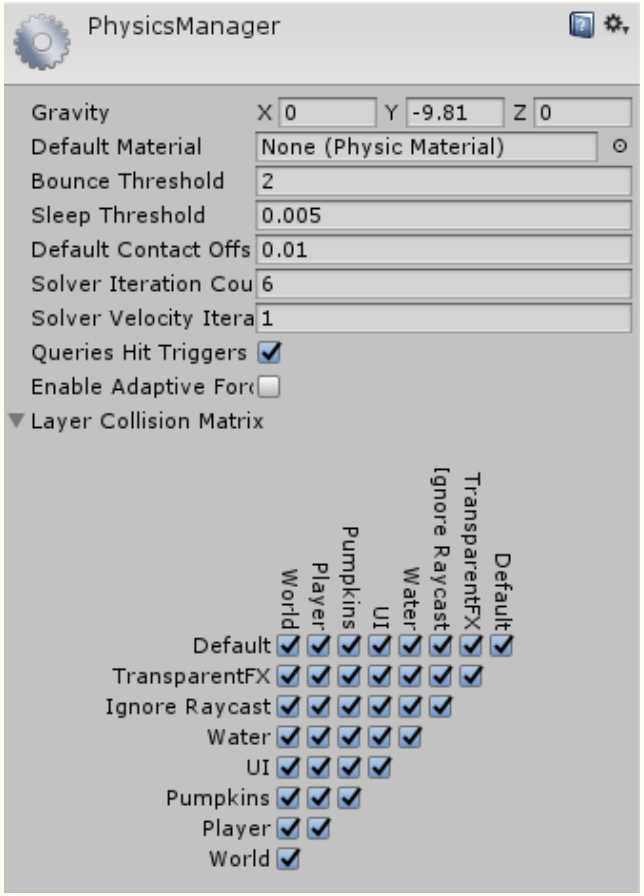
Wat botst met wat?

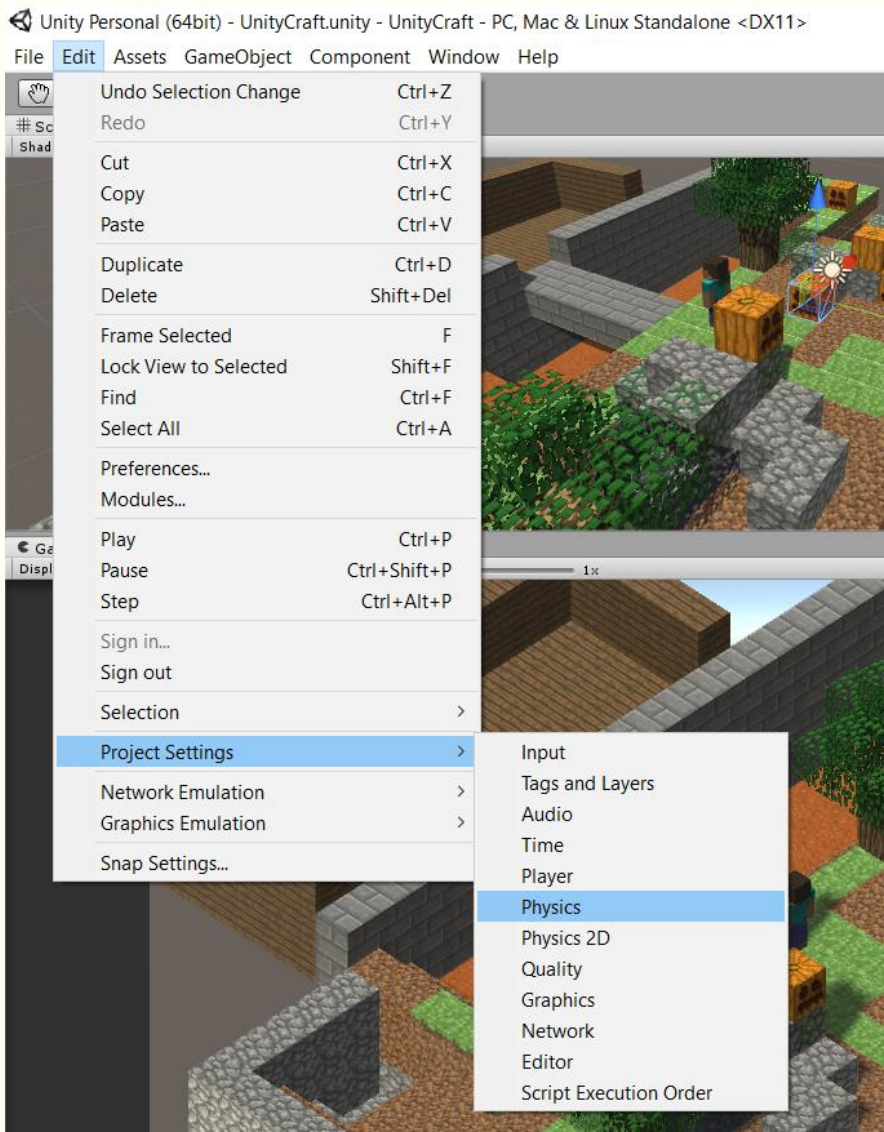
- Stop **Steve** in de **Player layer**, maar niet zijn kinderen!
  We zijn trouwens enkel geinteresseerd in *Colliders*. Bij Steve zit er enkel een collider op het Steve object zelf, ingebouwd in de *Character Controller*.



Wat botst met wat?

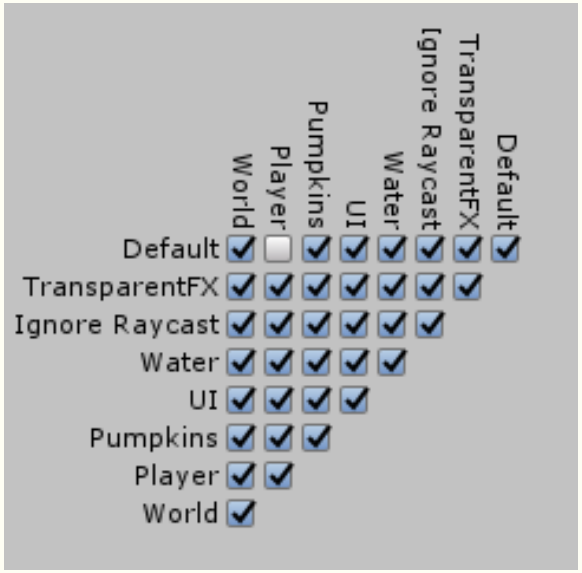
- Ga nu naar **Edit -> Project Settings -> Physics**
- Dat ziet er ingewikkeld uit, niet?





Wat botst met wat?

- Eigenlijk valt het goed mee. Het is de matrix die ons interesseert, en die weergeeft wat met wat kan botsen, uitgedrukt in *Layers*
- Herinner je je dat we de layer van een weggeworpen pompoen op 0 zetten? 0 staat voor de *Default layer*, dat is hier wel belangrijk.
- Nu, wat botst met wat?
- Player botst met World, anders zou Steve doorheen de blokken kunnen lopen
- Player botst met Pumpkins, anders zou Steve doorheen de pompoenen kunnen lopen
- Default botst met World, anders zouden weggegooide pompoenen door de grond vallen



- Default mag niet botsen met Player, zodat pompoenen niet botsen met Steve op het punt dat ze worden weggegooid
  → **dit vakje moeten we dus uitvinken**
- Het enige kleine nadeel is dat Steve zal kunnen lopen doorheen weggegooide pompoenen die op de grond liggen.

Opslaan!

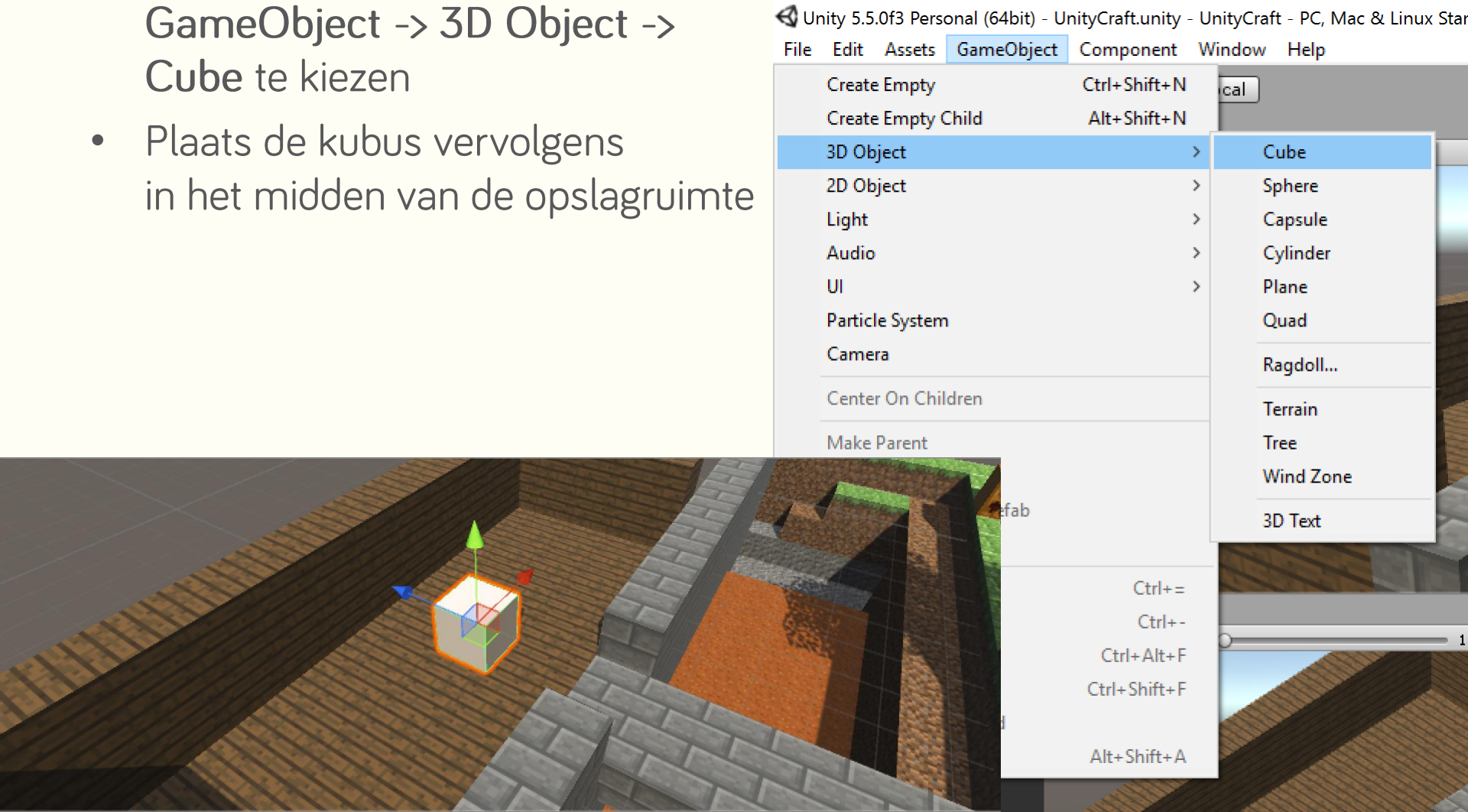
- Na al dat harde werk kan je even uitblazen!
- Sla je scene op door op Ctrl + S te drukken

Pompoenen tellen

- Om van dit projectje een echt game te maken gaan we een doel toevoegen: de pompoenen rapen in een zo kort mogelijke tijd.
- Daarvoor gaan we eerst tellen hoeveel pompoenen we in de opslagruimte gegooid hebben. Daarna gaan we een timer toevoegen om de tijd te meten.
- Om de pompoenen die in de opslagruimte gegooid worden te tellen moeten we ze kunnen detecteren van zodra ze in de opslagruimte terecht komen.
- We kunnen dit doen aan de hand van een *Trigger*. Dit is een zone die je in de scene kan plaatsen en die laat weten wanneer er een bepaald voorwerp de zone binnenkomt.

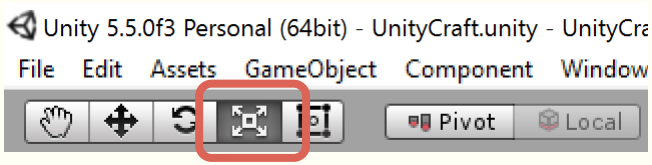
Pompoenen tellen

- Als basis voor onze trigger gaan we een **kubus** gebruiken
- Voeg een kubus in de scene toe door **GameObject -> 3D Object -> Cube** te kiezen
- Plaats de kubus vervolgens in het midden van de opslagruimte

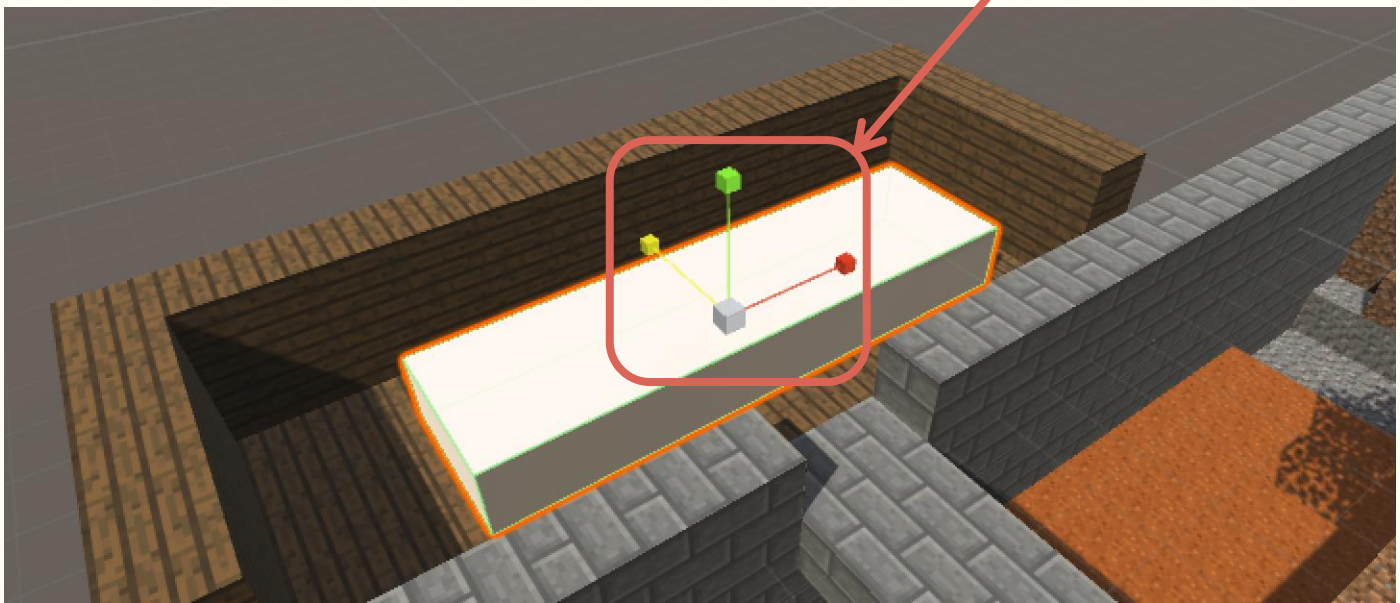


Pompoenen tellen

- Zorg ervoor dat onze kubus de hele opslagruimte omvat. Daarvoor kan je de kubus **schalen**
- Activeer hiervoor de **Scale** mode linksbovenaan

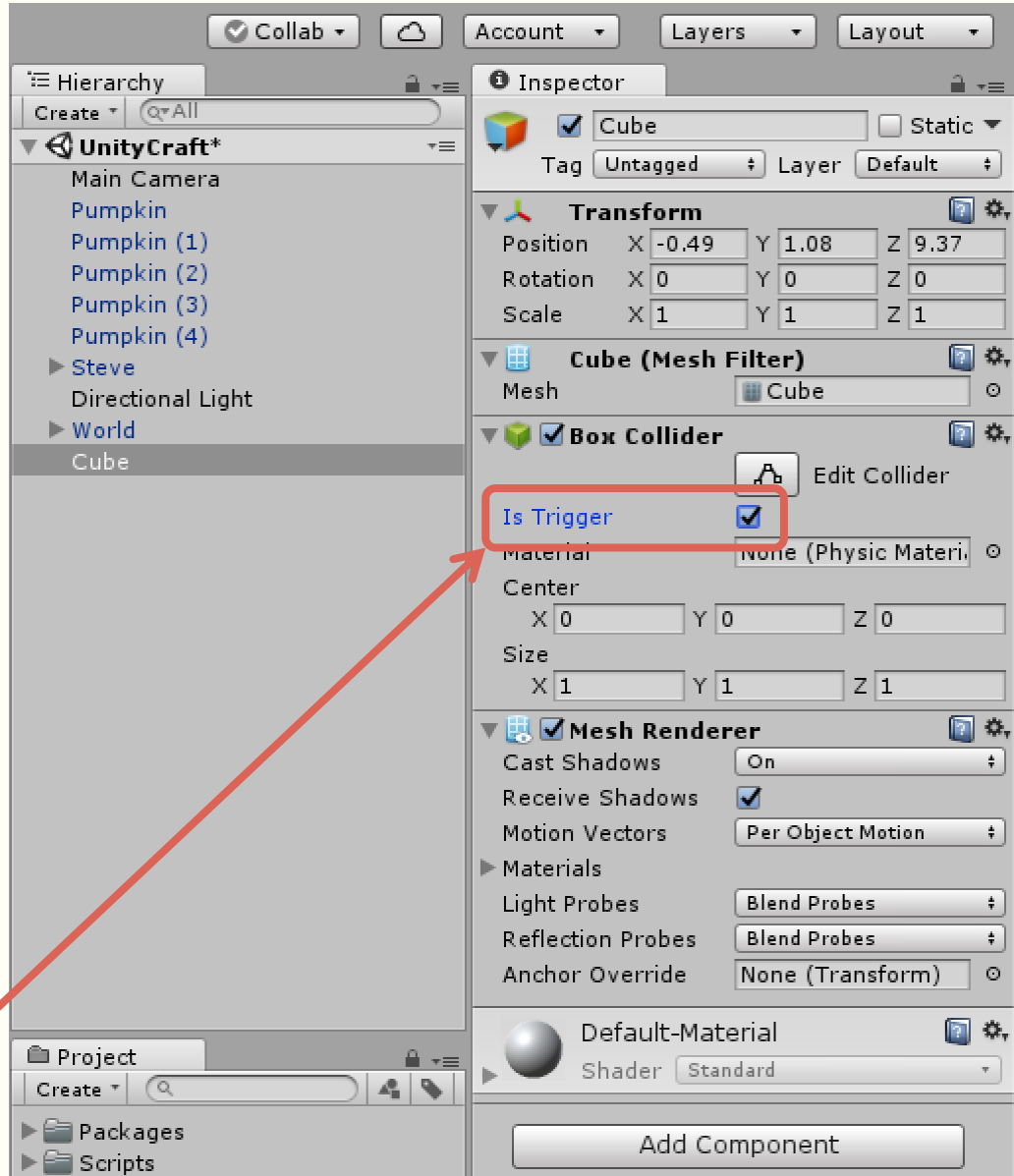


- Zodra deze mode is geactiveerd verandert de *Gizmo* van de geselecteerde kubus in de schaal-modus en kan je de kubus vergroten zodat die in de opslagruimte past



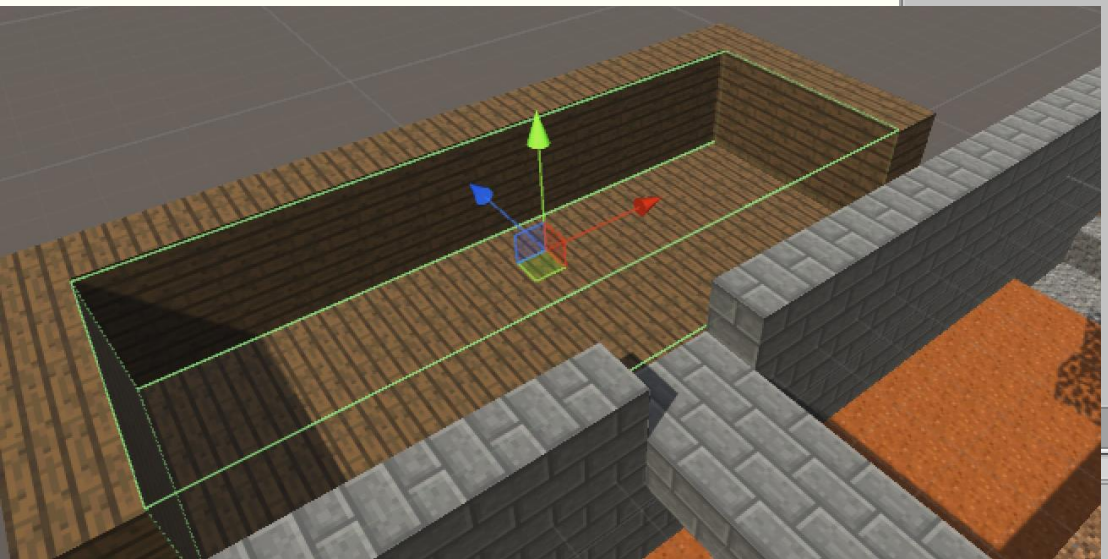
Pompoenen tellen

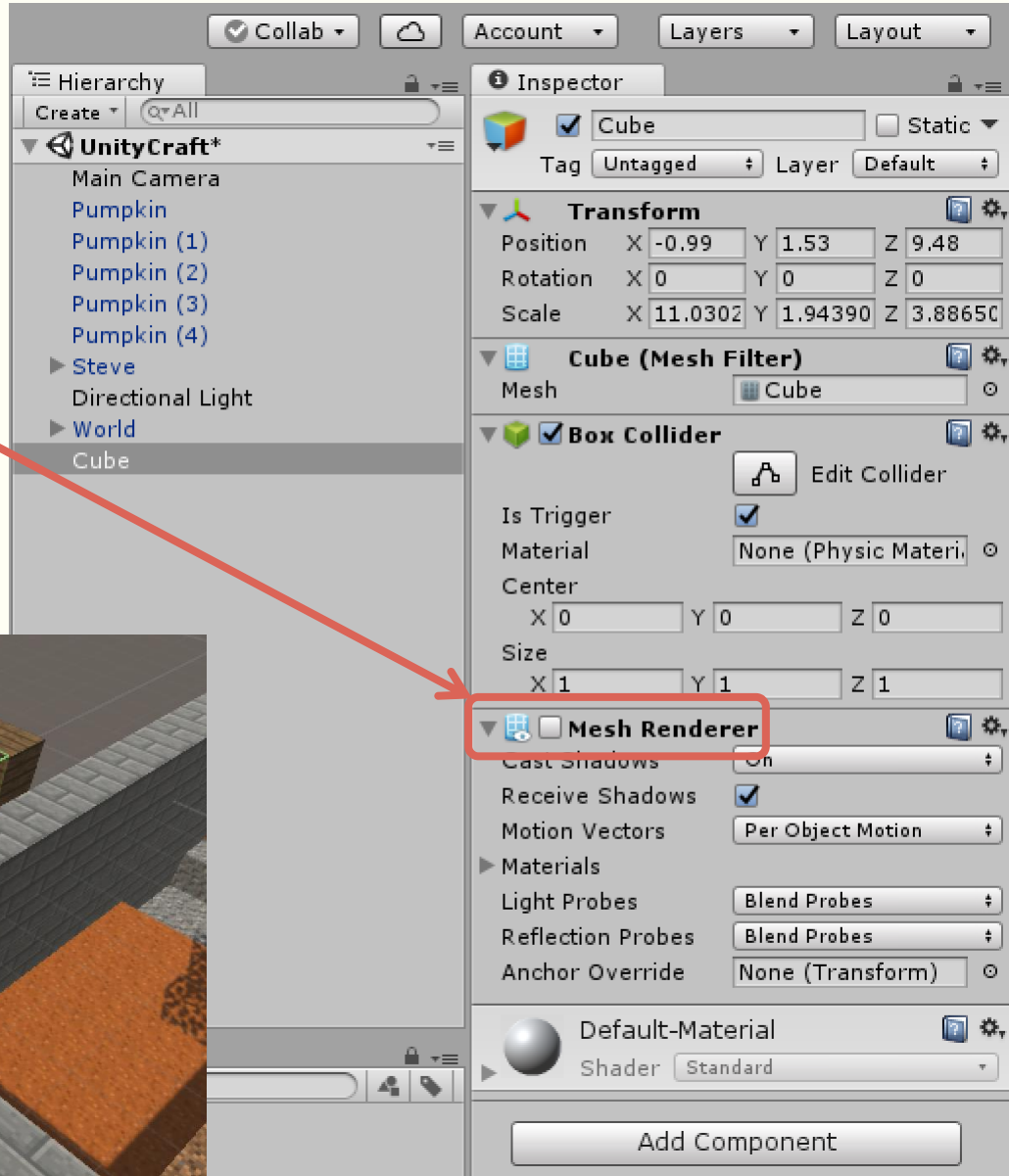
- Als je de kubus selecteert in het hierarchy paneel zal je zien dat deze standaard een **Box Collider** en een **Mesh Renderer** bevat.
- De Box Collider hebben we nodig, aangezien we andere GameObjects, meerbepaald pompoenen, willen detecteren. **Maar:** we willen niet dat de pompoenen met de kubus botsen, maar dat ze wel gedetecteerd worden als ze binnen de kubus komen. Daarvoor moeten we er dus **een Trigger van maken**.



Pompoenen tellen

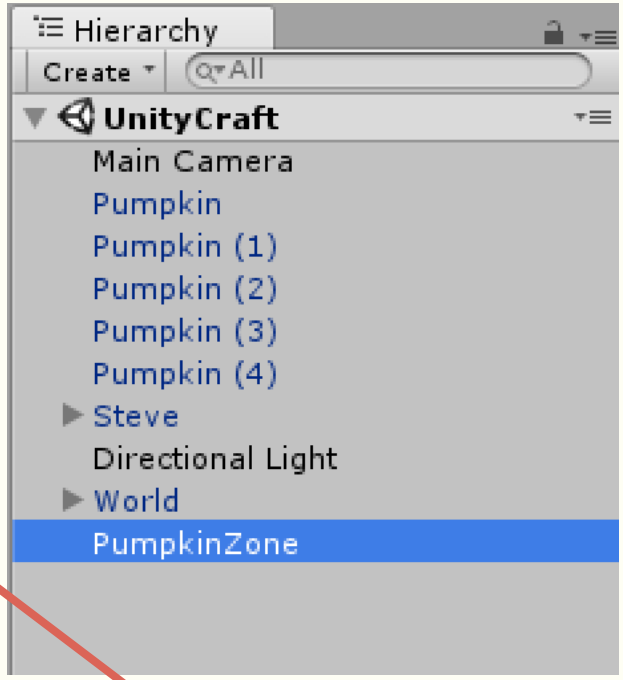
- De trigger is een onzichtbare zone die pompoenen detecteert.
- Om de kubus onzichtbaar te maken kan je de Mesh Renderer uitzetten

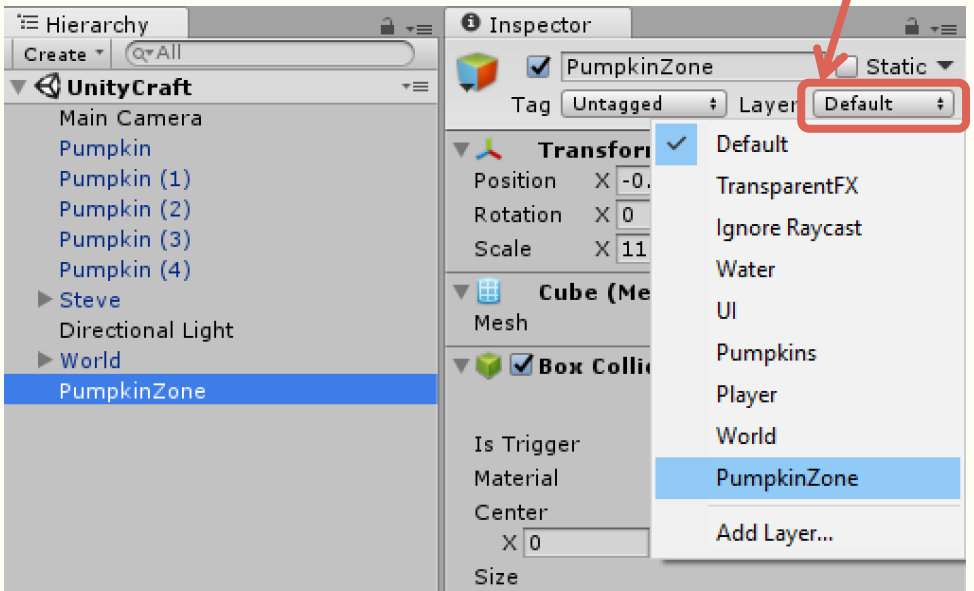


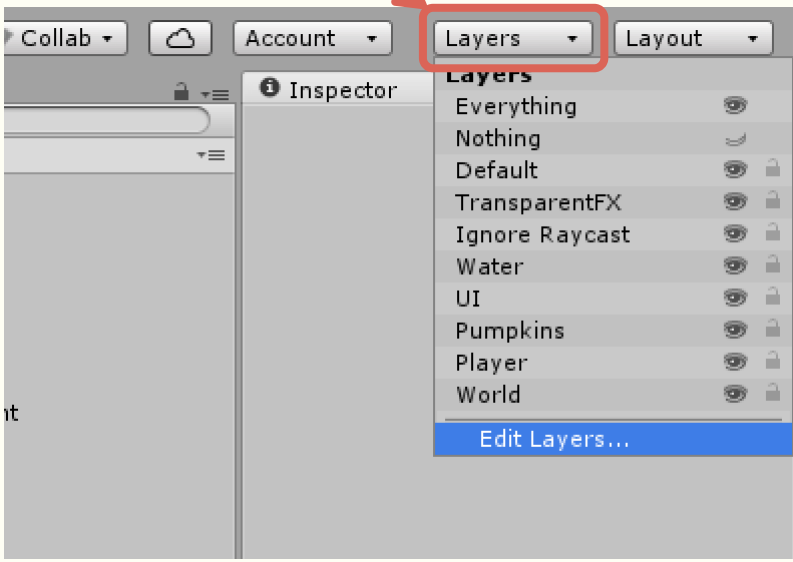


Pompoenen tellen

- We geven onze trigger een goeie naam, bvb *PumpkinZone*
- We voegen ook een nieuwe *Layer* toe waarin we de trigger gaan onderbrengen
- Noem de nieuwe layer *PumpkinZone* en voeg de trigger in deze layer toe

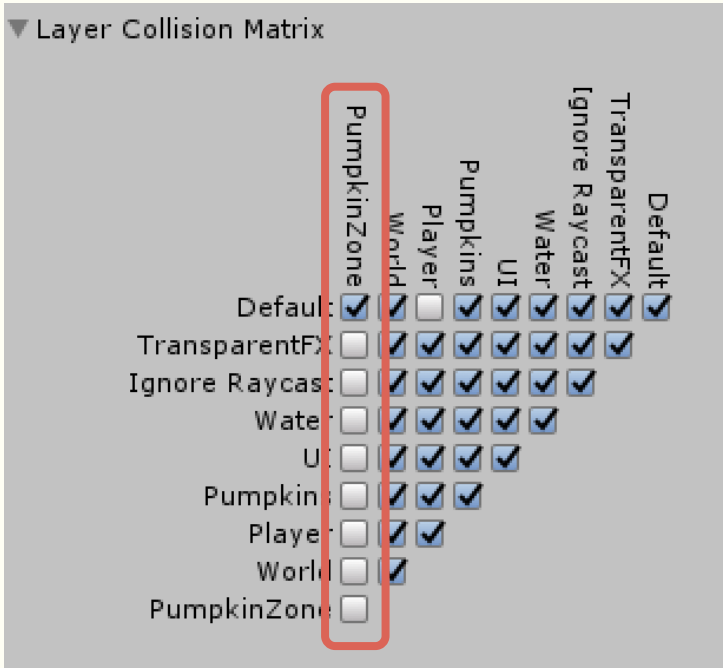






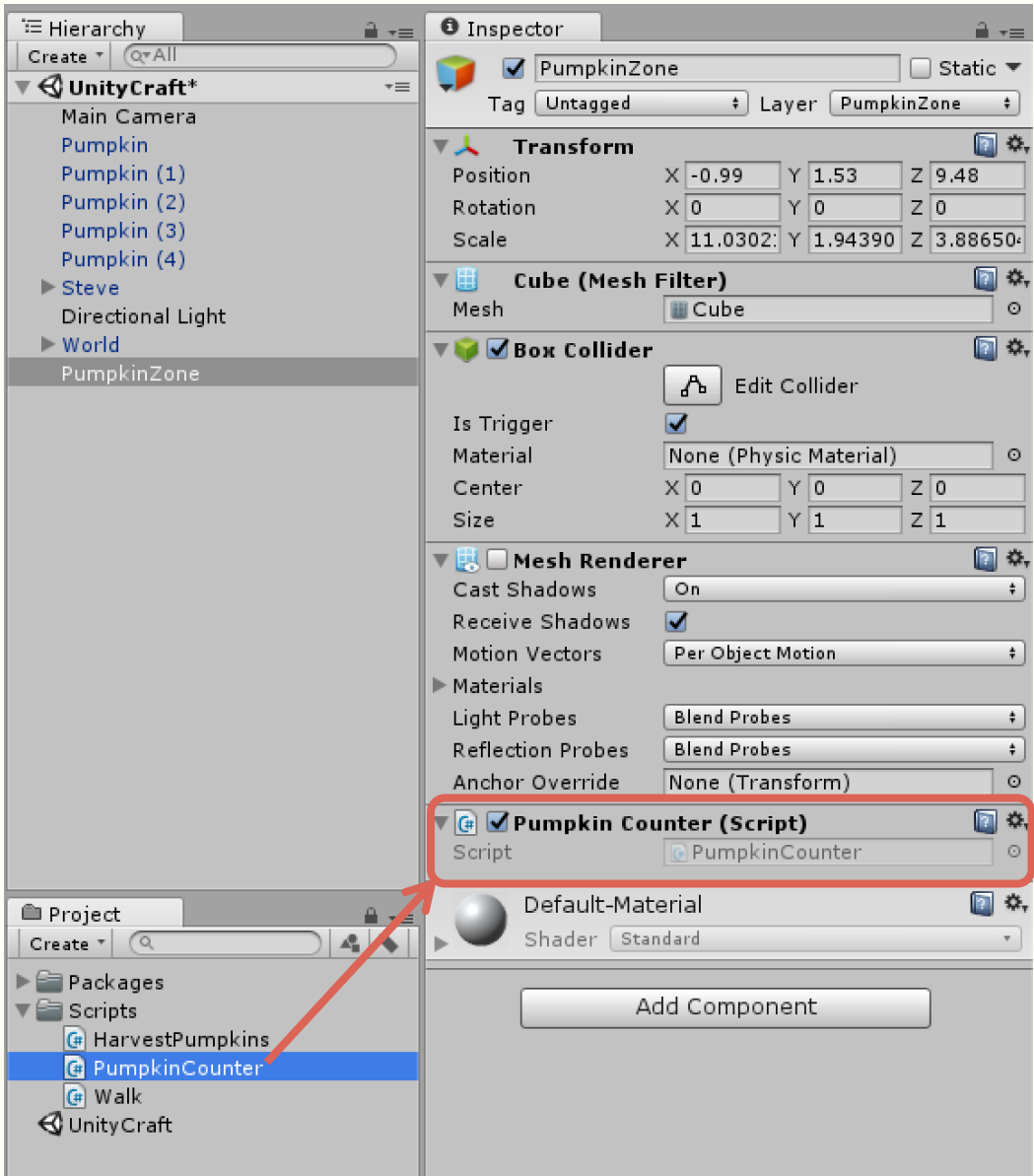
Pompoenen tellen

- De trigger zal enkel weggegooide pompoenen detecteren.
- Herinner je dat we de weggegooide pompoenen toevoegen aan de *Default* layer?
- We gaan de layer waarin de trigger zich bevindt dus zo instellen dat die enkel gameobjects uit de *Default* layer detecteert.
- Daarvoor gaan we naar *Edit -> Projects Settings -> Physics* en stellen we de *Layer Collision Matrix* zo in dat *PumpkinZone* met *Default* kan interageren, maar met geen enkele layer



Pompoenen tellen

- De volgende stap is het schrijven van een script waarin we een teller gaan bijhouden die aanduidt hoeveel pompoenen we al in de opslagruimte hebben gegooid
- Maak een nieuw script met de naam *PumpkinCounter*, en voeg dit script toe aan de trigger
- Dubbelklik op het script om het te openen



Pompoenen tellen

- Om de pompoenen te kunnen tellen moeten we een melding krijgen elke keer een pompoen in de trigger terecht komt
- In Unity kunnen we hiervoor een speciale methode gebruiken: *OnTriggerEnter*. Deze methode wordt automatisch opgeroepen telke male er iets in de trigger terecht komt.
- Natuurlijk werkt dit enkel als het script is toegevoegd aan een trigger.
- We kunnen deze methode alvast eens testen:

```
public class PumpkinCounter : MonoBehaviour
{
    private void OnTriggerEnter(Collider other)
    {
        print("Pumpkin detected");
    }
}
```

- Dit script zal ervoor zorgen dat er *"Pumpkin detected"* in de console wordt geprint telkens een pompoen in de opslagruimte wordt gegooid. Als dit niet zou werken, kan je de vorige slides nog eens overlopen.

Pompoenen tellen

- Als de trigger naar behoren werkt, kunnen we vervolgens een teller toevoegen die het aantal pompoenen bijhoudt die in de opslagruimte zijn gegooid.
- Probeer dit eerst even zelf, voordat je naar de volgende slide kijkt.

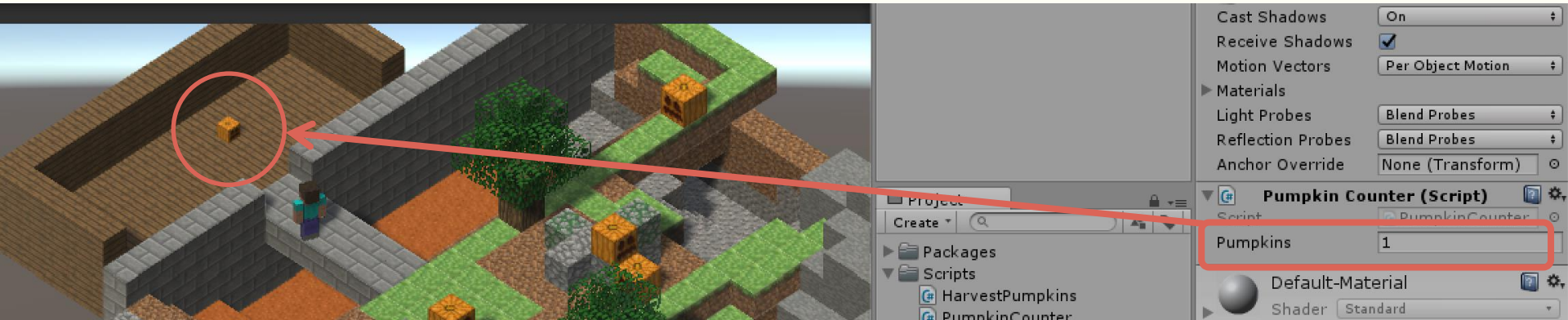
Pompoenen tellen

- Onderstaand stukje code voorziet een teller die bijhoudt hoeveel pompoenen er in de opslagruimte zijn gegooid.

```
public class PumpkinCounter : MonoBehaviour
{
    public int Pumpkins = 0;

    private void OnTriggerEnter(Collider other)
    {
        Pumpkins++;
    }
}
```

Dit is de verkorte notatie voor: `Pumpkins = Pumpkins + 1`

- Je kan het nieuw stukje code testen door een pompoen in de opslagruimte te gooien en te kijken of de teller op 1 komt te staan.

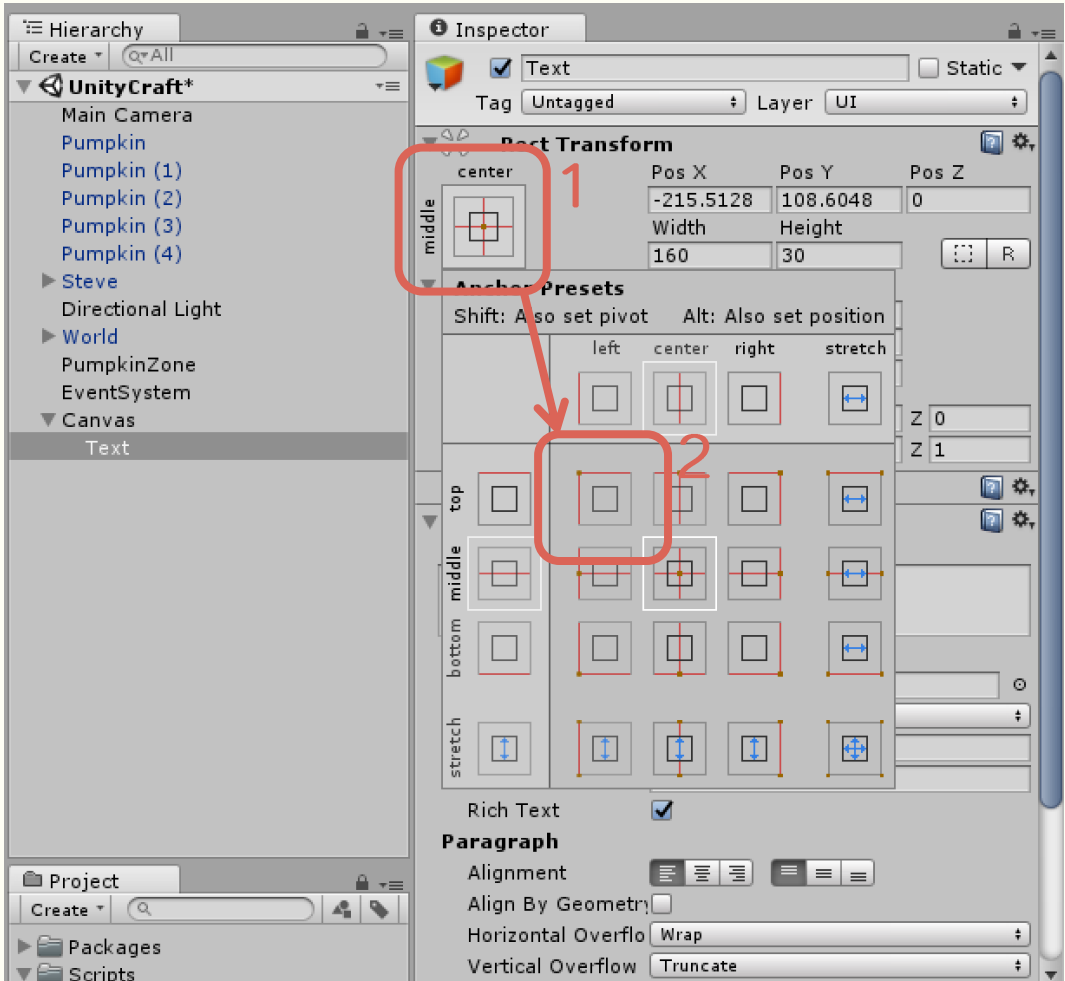


Pompoenen tellen

- De teller kunnen we tonen op het scherm door gebruik te maken van het UI systeem van Unity. UI staat voor User Interface, wat de verzamelnaam is voor alle vormen van 2D interactie: knoppen, tekstvakken, keuzemenu's, figuren,...
- We voegen een tekstveld toe dat de waarde van de teller zal weergeven. Kies *GameObject->UI->Text*
- Het tekstveld gaan we nu zodanig instellen dat het steeds de linkerbovenhoek van het scherm volgt, ongeacht de resolutie van het uiteindelijke game

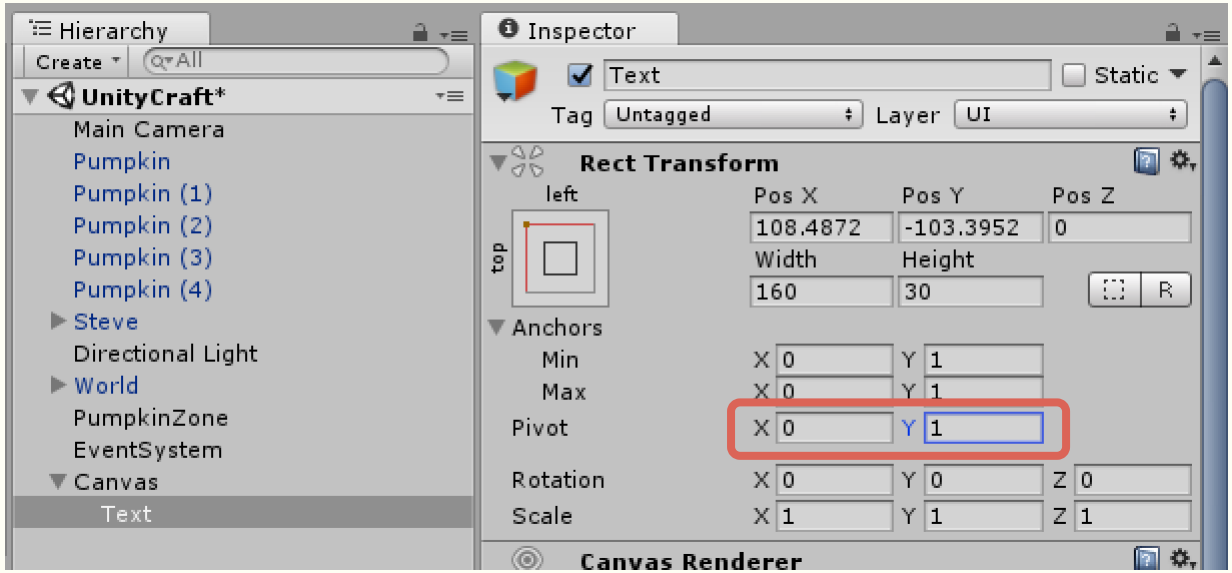
Pompoenen tellen

- Ten eerste moet het *anker* van het tekstveld zodanig worden ingesteld dat het zich vasthecht aan de linker bovenhoek van het UI canvas. Dit kan je doen door de *Anchor Preset* (1) aan te klikken en vervolgens deze te veranderen naar linksboven (2).

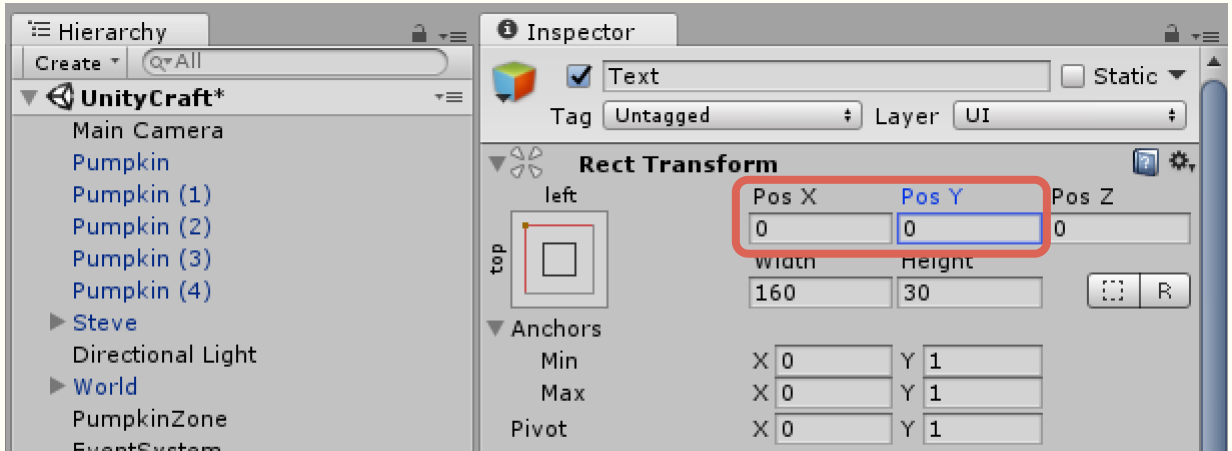


Pompoenen tellen

- Vervolgens dient de *Pivot* van het tekstvak op (0, 1) gezet worden.



- Tenslotte zetten we *PosX* en *PosY* op (0, 0).

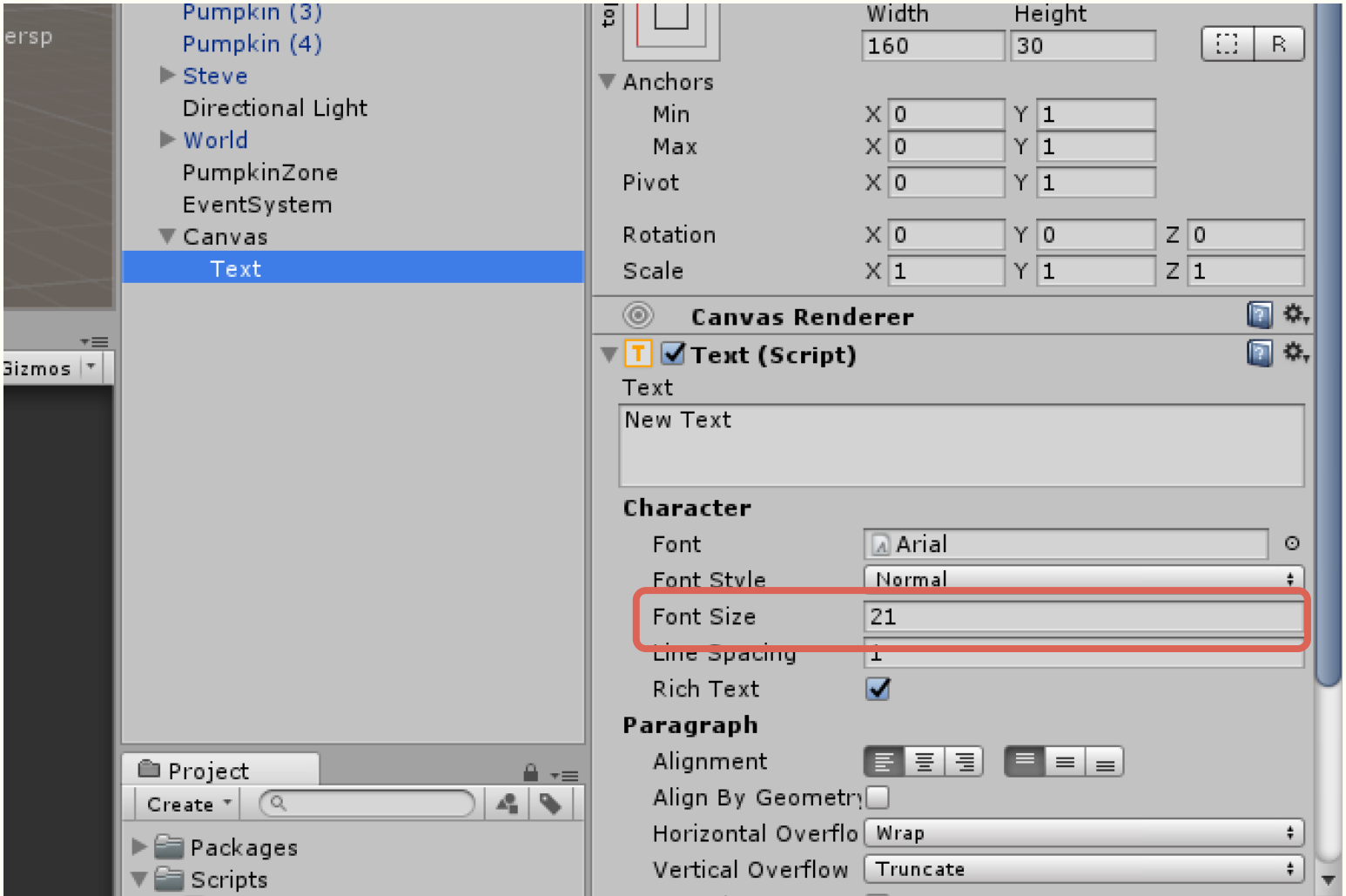


Pompoenen tellen

- Al deze instellingen zouden er moeten voor zorgen dat het tekstvak linksbovenaan verschijnt, ongeacht de resolutie van het *game view*

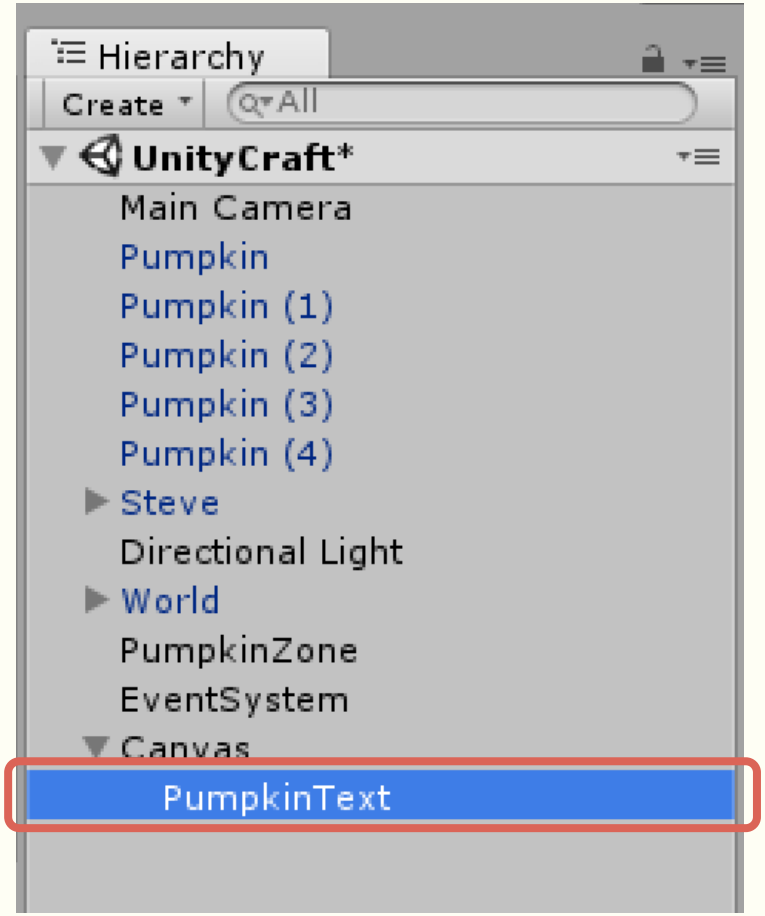
Pompoenen tellen

- Je kan de tekst ook eventueel wat groter maken, door de *Font Size* parameter aan te passen



Pompoenen tellen

- Hernoem het tekstveld zodat het een meer betekenisvolle naam krijgt, bijvoorbeeld *PumpkinText*



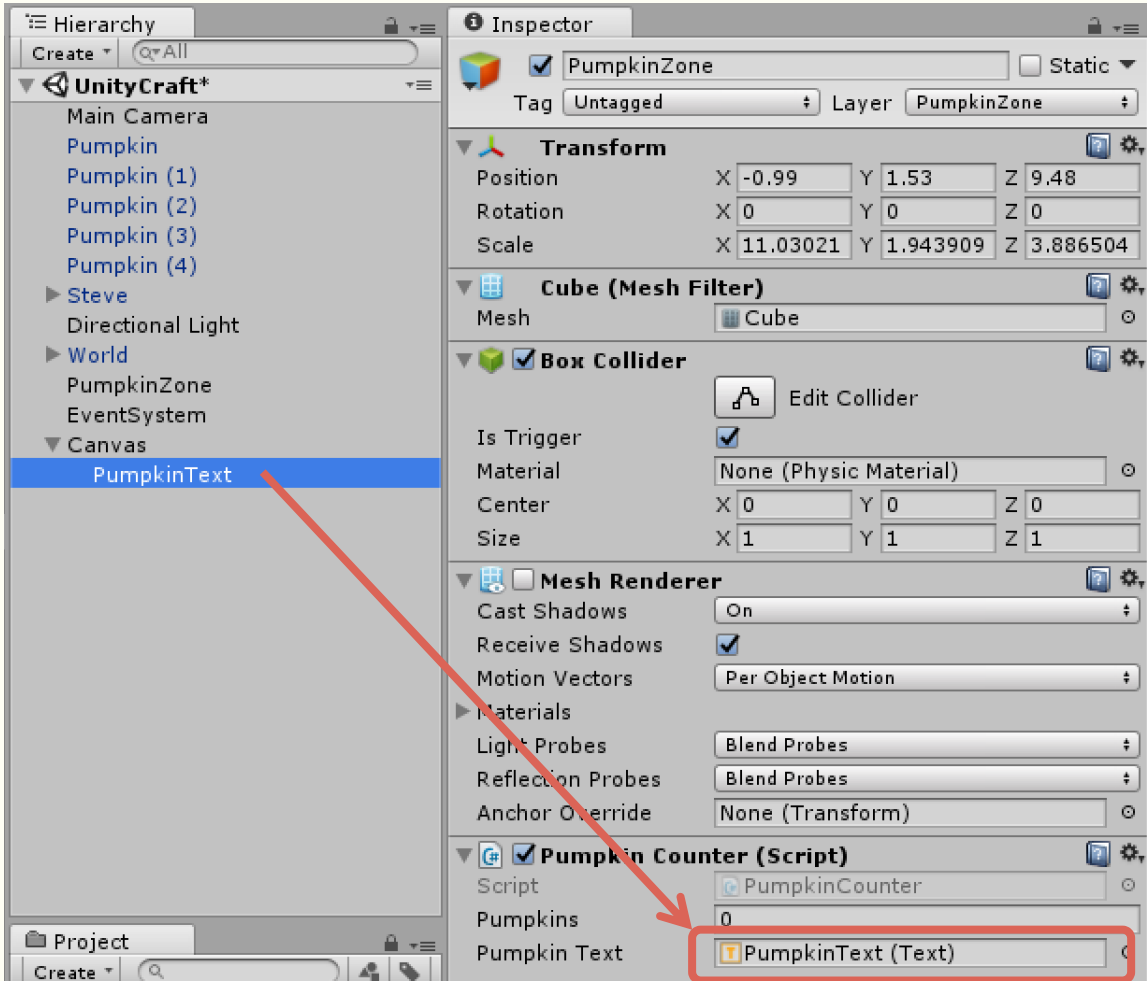
Pompoenen tellen

- We keren nu terug naar het *PumpkinCounter* script en voegen een publieke variable toe die we gaan laten refereren naar het tekstveld dat we zojuist hebben gemaakt.
- Om UI elementen van Unity in code te kunnen aanspreken, moet je de *namespace* UnityEngine.UI toevoegen aan de code.

```
using UnityEngine;
using UnityEngine.UI;

public class PumpkinCounter : MonoBehaviour
{
    public int Pumpkins = 0;
    public Text PumpkinText;

    private void OnTriggerEnter(Collider other)
    {
        Pumpkins++;
    }
}
```

Pompoenen tellen

- De publieke variabele die we net hebben aangemaakt kunnen we vervolgens laten refereren naar het tekstveld door het *PumpkinText* gameobject te slepen naar het slot van de publieke variabele.



Pompoenen tellen

- We kunnen het *PumpkinCounter* script nu uitbreiden zodat het aantal verzamelde pompoenen wordt weergegeven in het tekstveld
- Probeer je het eerst zelf eens? Je kan de tekst van het tekstveld aanpassen in de code via PumpkinText.**text**

Pompoenen tellen

- Onderstaand stukje code toont hoe je het tekstveld kan gebruiken om het aantal pompoenen weer te geven. Let hierbij op het gebruik van het '+' teken, waarmee je woorden en variabelen aan elkaar kan plakken tot één stukje tekst.

```
public class PumpkinCounter : MonoBehaviour
{
    public int Pumpkins = 0;
    public Text PumpkinText;

    private void Start()
    {
        PumpkinText.text = "Pumpkins: 0";
    }

    private void OnTriggerEnter(Collider other)
    {
        Pumpkins++;
        PumpkinText.text = "Pumpkins: " + Pumpkins;
    }
}
```

Als het game start zetten we de tekst op "Pumpkins: 0", aangezien we op dat moment nog geen pompoenen hebbe verzameld

We gebruiken het '+' teken om vóór het aantal pompoenen de tekst "Pumpkins: " te plakken

Als een pompoen in de opslagruimte wordt gegooid, veranderen we de tekst zodat het huidige aantal verzamelde pompoenen wordt weergegeven.

Pompoenen tellen

- Als we nu een pompoen gooien in de opslagruimte verschijnt er in de linker bovenhoek hoeveel pompoenen we al hebben verzameld.

Opslaan!

- Dat was een heel werk! Even pauze nemen…
- Sla je scene op door op Ctrl + S te drukken

Tijd meten

- Om van het projectje een echt game te maken moeten we nog zorgen voor een score
- In dit geval zal de score de tijd zijn waarin de speler al de pompoenen kan verzamelen
- We gaan dus een timer (tijdsklok) moeten toevoegen zodat we de tijd kunnen meten
- We zullen de timer toevoegen aan het script dat de pompoenen telt, zo kunnen we de timer makkelijk stoppen als alle pompoenen verzameld zijn
- Open het *PumpkinCounter* script
- We voegen een publieke variabele *Timer* toe van het type *float* (decimaal getal) en voegen ook een *Update* functie toe waarin we *Timer* vermeerderen met *Time.deltaTime* (de tijd in seconden tussen de huidige en de vorige Update)
- Op de volgende slide vind je het geupdate script

Tijd meten

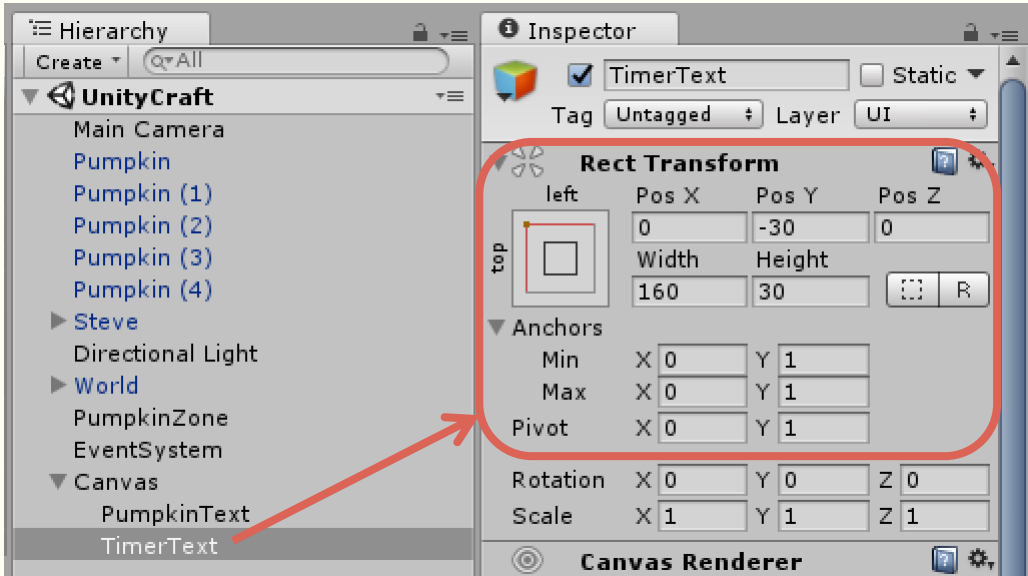
```
public class PumpkinCounter : MonoBehaviour
{
    public int Pumpkins = 0;
    public float Timer = 0;

    public Text PumpkinText;

    private void Start()
    {
        PumpkinText.text = "Pumpkins: 0";
    }

    private void Update()
    {
        Timer += Time.deltaTime;
    }

    private void OnTriggerEnter(Collider other)
    {
        Pumpkins++;
        PumpkinText.text = "Pumpkins: " + Pumpkins;
    }
}
```

Tijd meten

- De timer moet nu nog op het scherm getoond worden. We hebben dit al gedaan voor de pompoenteller, dus dezelfde werkwijze kan gebruikt worden voor de timer.
- Kijk even terug hoe we de pompoenteller hebben weergegeven in de linkerbovenhoek van het scherm en probeer dan zelf eens om de timer te laten verschijnen op het scherm.

Tijd meten

- Om de timer weer te geven op het scherm hebben we een tekstvak nodig.
- Voeg dit toe door *GameObject -> UI -> Text* te kiezen.
  - Een alternatieve en snellere manier is door het reeds aangemaakte tekstvak *PumpkinText* te selecteren en op **Ctrl + d** (d van *Duplicate*, wat verdubbelen betekent) te drukken.
- Geef het tekstvak de naam *TimerText* en pas de onderstaande waarden toe voor positie, pivot en anchor.

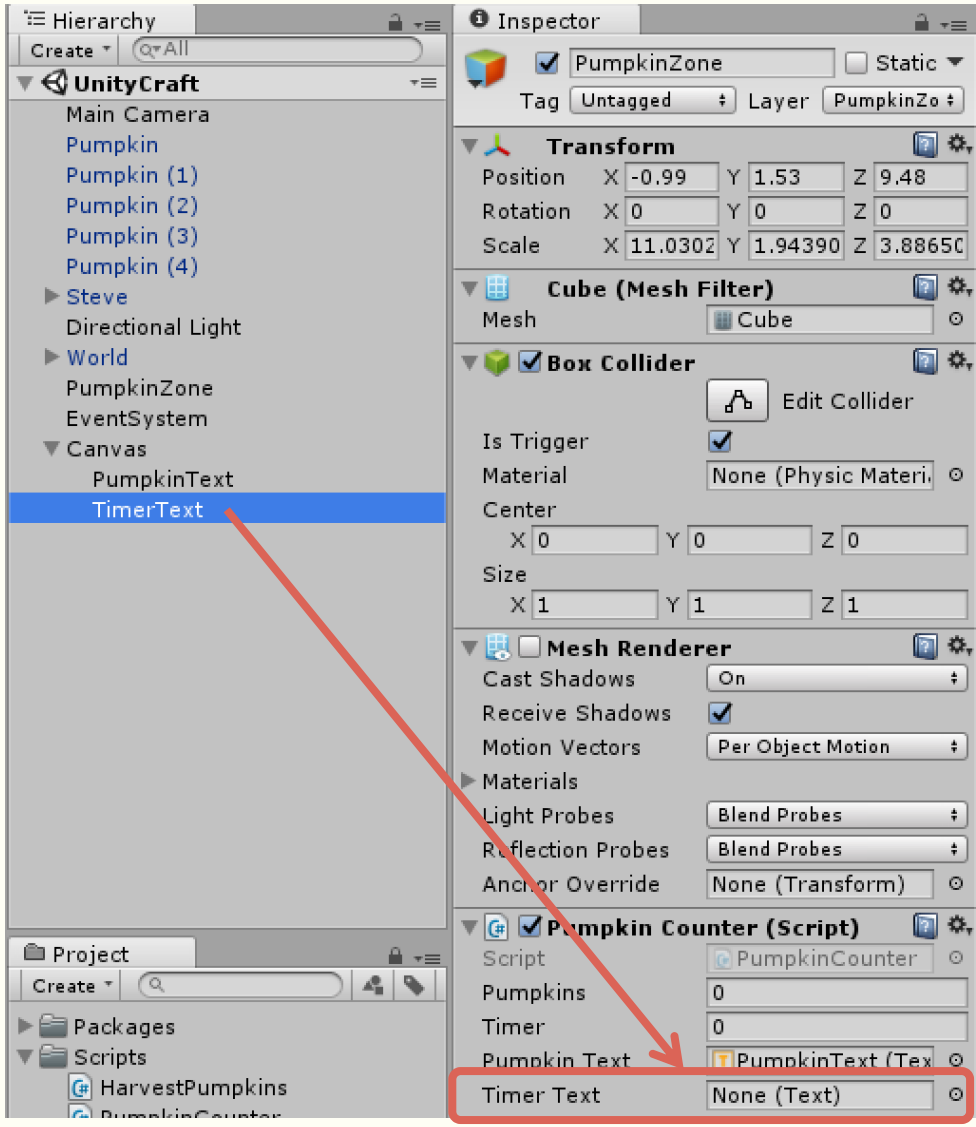


Tijd meten

- Maak een publieke variabele aan in het *PumpkinCounter* script met de naam *TimerText*

```
public class PumpkinCounter : MonoBehaviour
{
    public int Pumpkins = 0;
    public float Timer = 0;

    public Text PumpkinText;
    public Text TimerText;
```

- Vervolgens kan je het net aangemaakte tekstvak linken aan deze publieke variabele



Tijd meten

- Nu dat we in het *PumpkinCounter* script een referentie gemaakt hebben naar het tekstveld kunnen we er vervolgens de tijd in invullen.

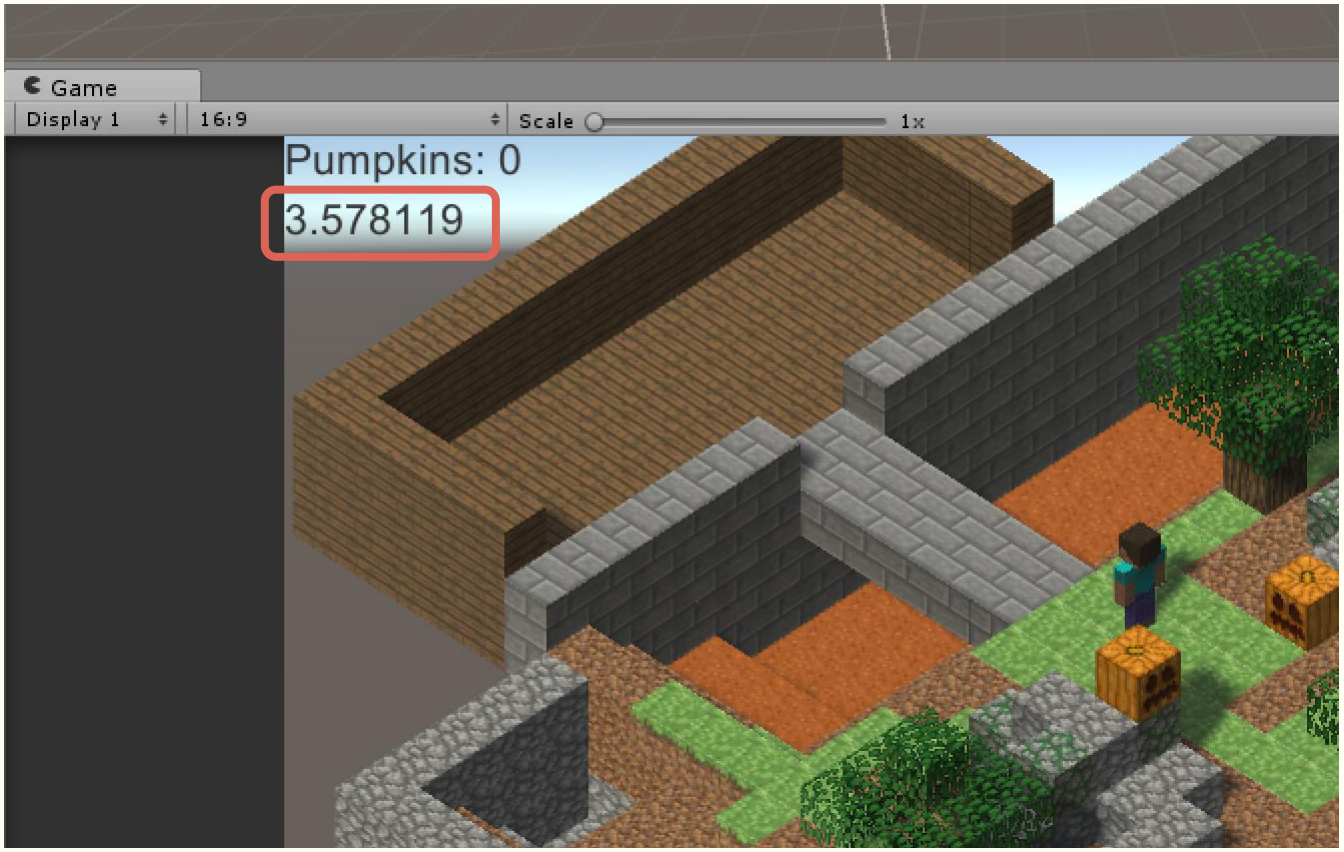
```
private void Update()
{
    Timer += Time.deltaTime;
    TimerText.text = Timer.ToString();
}
```

- We voegen *.ToString( )* toe achter Timer om de waarde van Timer om te zetten in een woord (in programmeertaal wordt een woord of een stuk tekst aangeduidt met *string*).
- Als we dit niet doen krijgen we een compileerfout:

- Dit komt omdat *TimerText.text* een woord of stukje tekst verwacht als waarde (*string*), terwijl Timer een decimaal getal is (*float*)

Tijd meten

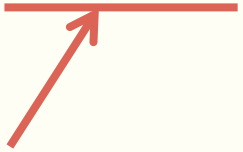
- Als we het game nu starten zien we de timer optellen in de linkerbovenhoek



Tijd meten

- We gaan de timer nog een beetje 'netjes' maken. Zo hoeven we niet zoveel cijfers na de komma te tonen, en gaan we nog een 's' van *seconden* achteraan toevoegen om de tijdseenheid aan te duiden.
- Beide aanpassingen (cijfers na de komma en de 's' achteraan) kunnen we toepassingen door iets toe te voegen aan de *ToString( )* methode.

```
private void Update()
{
    Timer += Time.deltaTime;
    TimerText.text = Timer.ToString("0.00s");
}
```



- De *ToString* methode laat toe om het resulterende stukje tekst te *formatteren* met een speciale notatie. '0' staat voor een willekeurig cijfer. We definiëren hier dus een getal met 2 cijfers na de komma en een 's' op het einde.

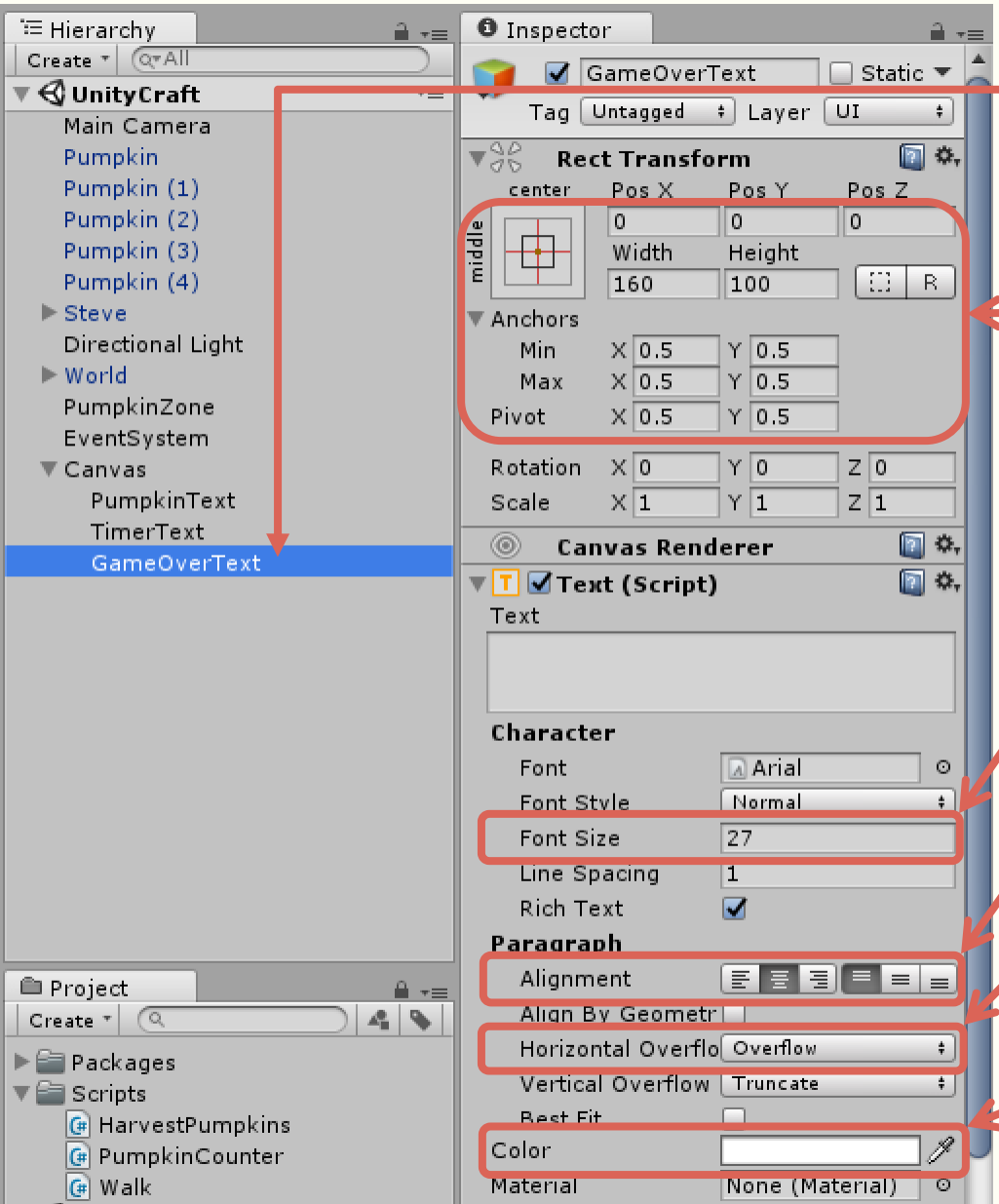
Tijd meten

- Dat ziet er al veel beter uit!

Game einde

- Er resteert ons nog één taak om het game te voltooien: een einde.
- Het spelt stopt als de speler alle pompoenen in de opslagruimte heeft gegooid.
- Op dat moment stopt de timer, en wordt op het scherm een bericht getoond dat het spel gedaan is met daarbij de waarde van de timer.
- Je kan dit als oefening zelf proberen maken, waarna je de oplossing in de volgende slides terugvindt.

Game einde



We voegen een nieuw tekstveld toe met de naam *GameOverText*.

We plaatsen het tekstvak in het midden van het scherm en maken het hoog genoeg.

Het lettertype maken we groter.

De tekst wordt gecentreerd op het scherm.

We stellen het tekstvak zo in dat de tekst buiten het tekstvak mag gaan.

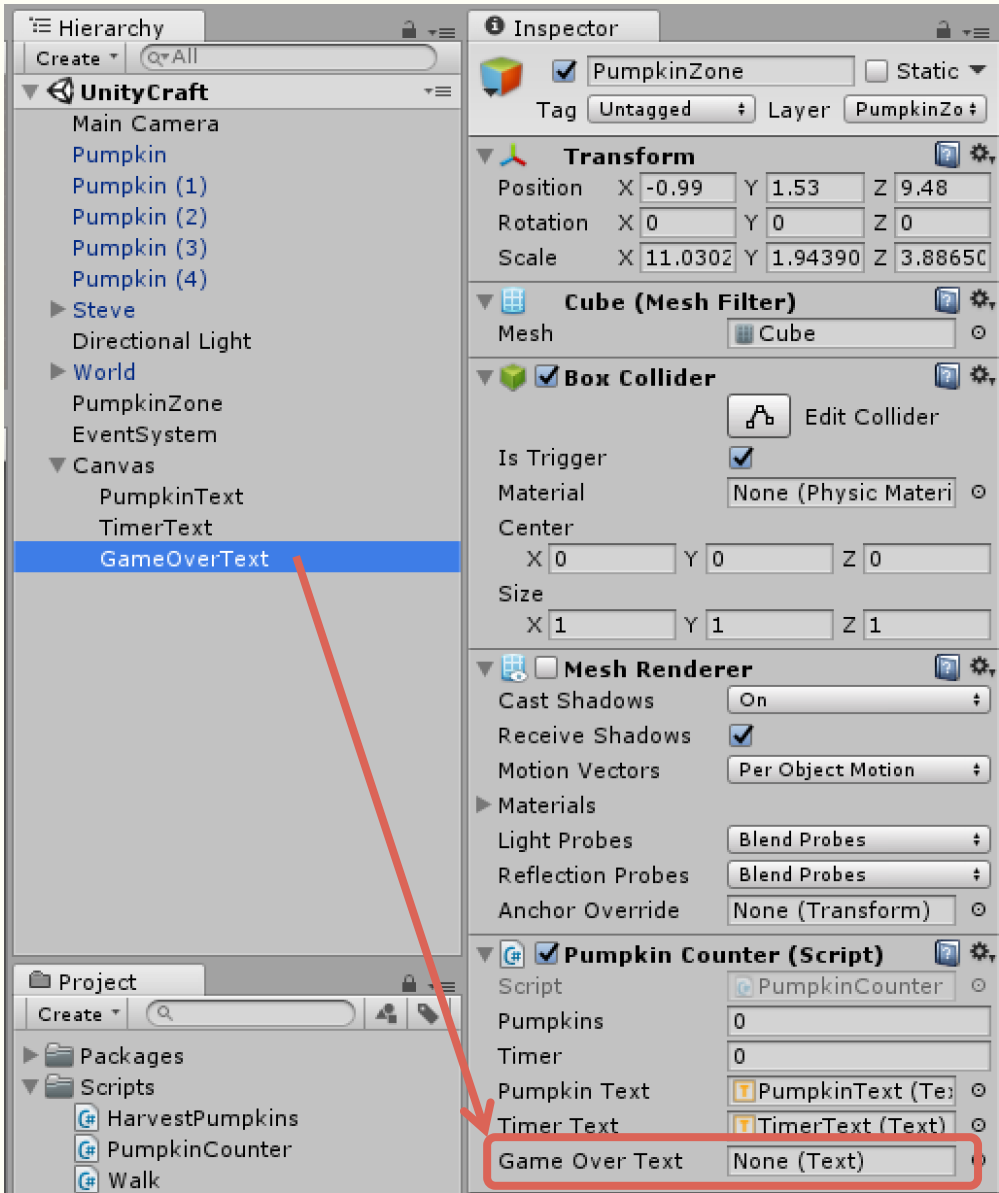
En de kleur van de tekst maken we wit.

Game einde

- In het *PumpkinCounter* script maken we een publieke variabele *GameOverText* aan die refereert naar het *GameOverText* tekstveld.

```
public class PumpkinCounter : MonoBehaviour
{
    public int Pumpkins = 0;
    public float Timer = 0;

    public Text PumpkinText;
    public Text TimerText;
    public Text GameOverText;
```

- In de inspector kunnen we vervolgens de verbinding maken tussen *GameOverText* en de publieke variabele.



Game einde

- Als je het *GameOverText* tekstveld selecteert kan je in de inspector de tekst invullen die op het scherm zal worden weergegeven.

Game einde

- De boodschap die we net op het scherm hebben laten verschijnen willen we natuurlijk enkel tonen als het game afgelopen is. Als we het game starten zetten we het *GameOverText* tekstveld dan ook uit.
- Dit doen we in het *PumpkinCounter* script, waar we reeds een referentie hebben gemaakt naar *GameOverText.*

```
private void Start()
{
    PumpkinText.text = "Pumpkins: 0";
    GameOverText.enabled = false;
}
```

Game einde

- Als de speler alle pompoenen heeft verzameld is het game afgelopen. De timer moet dan stoppen en de eindboodschap moet op het scherm getoond worden.
- In de *OnTriggerEnter* functie van het *PumpkinCounter* script kunnen we deze functionaliteit toevoegen.
- In het onderstaande stukje code moet de speler 5 pompoenen verzamelen om het spel te beëindigen.

```
private void OnTriggerEnter(Collider other)
{
    Pumpkins++;
    PumpkinText.text = "Pumpkins: " + Pumpkins;

    if (Pumpkins == 5)
    {
        enabled = false;
        GameOverText.enabled = true;
    }
}
```

"enabled = false;" zorgt ervoor dat de Update methode van het GameObject waaraan dit script is toegekend niet meer wordt opgeroepen.
Hierdoor zal de code die de timer vermeerdert niet meer uitgevoerd worden.

Game einde

- Als je alle pompoenen hebt geraapt, zou de eindboodschap van je game er ongeveer zoals hieronder moeten uitzien.
- De eindscore is de tijd in de linkerbovenhoek.

Proficiat!

- We zijn er geraakt! Je hebt nu heel wat technieken kunnen oefenen die gebruikt worden bij het maken van games.
- Je hebt vast en zeker ook gemerkt waarom het belangrijk is om wiskunde onder de knie te hebben als je games wil maken.
- Maar vooral, games maken is iets dat iedereen kan leren, en hoeft helemaal niet moeilijk te zijn.

Licenties

- Dit werk valt onder een Creative Commons Naamsvermelding-NietCommercieel-GelijkDelen 4.0 Internationaal-licentie.

- Volgende assets zijn gebruikt in deze tutorial:
  - Minecraft Steve en Minecraft blocks, gemaakt door Boxscape Studios: https://sites.google.com/view/boxscape-studios/downloads
  - Floppy afbeelding: https://commons.wikimedia.org/wiki/File:Media-floppy.svg
  - Bariol lettertype: http://atipofoundry.com/fonts/bariol
- Deze tutorial is geïnspireerd op Minecraft: https://account.mojang.com/terms
- En tenslotte een dankwoordje aan de mensen die de Unity game engine ontwikkelen: