3D Game:

## Probuilder,Progrid en Snaps Assets:

## 

De map is gemaakt met pro grids die het versimpeld om assets aan elkaar te plakken. Snaps Assets hebben wij gebruikt voor de muren,grond,dak en de objecten op de grond.

Probuilder en progrid voeg je toe via de package manager.

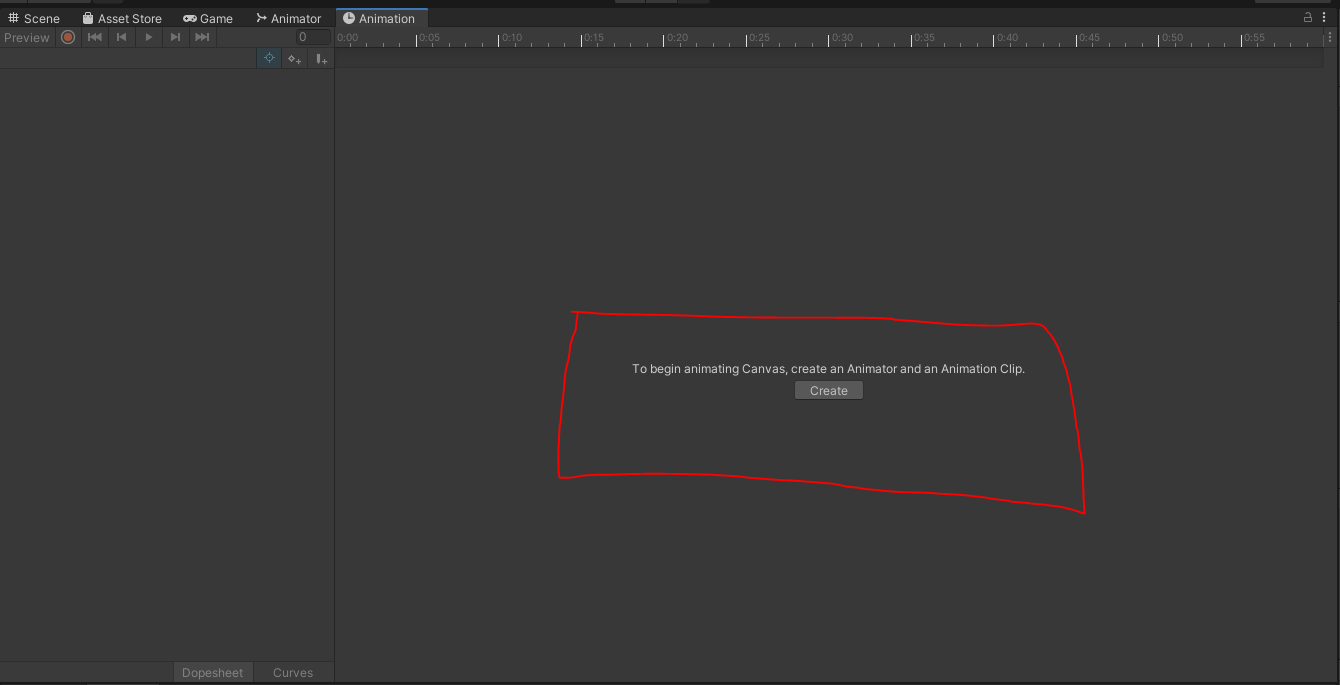
 progrid daarmee kan je objecten perfect met elkaar plaatsen en je krijgt een grid te zien voor afstanden beter te kunnen bepalen.

## 

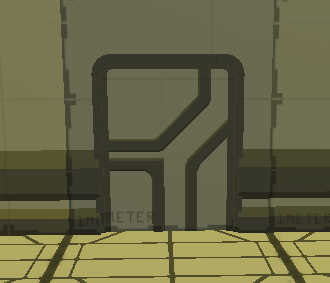
## 

## Deuren:

De deuren op level 1 hebben een zelf gemaakte animatie gekregen.



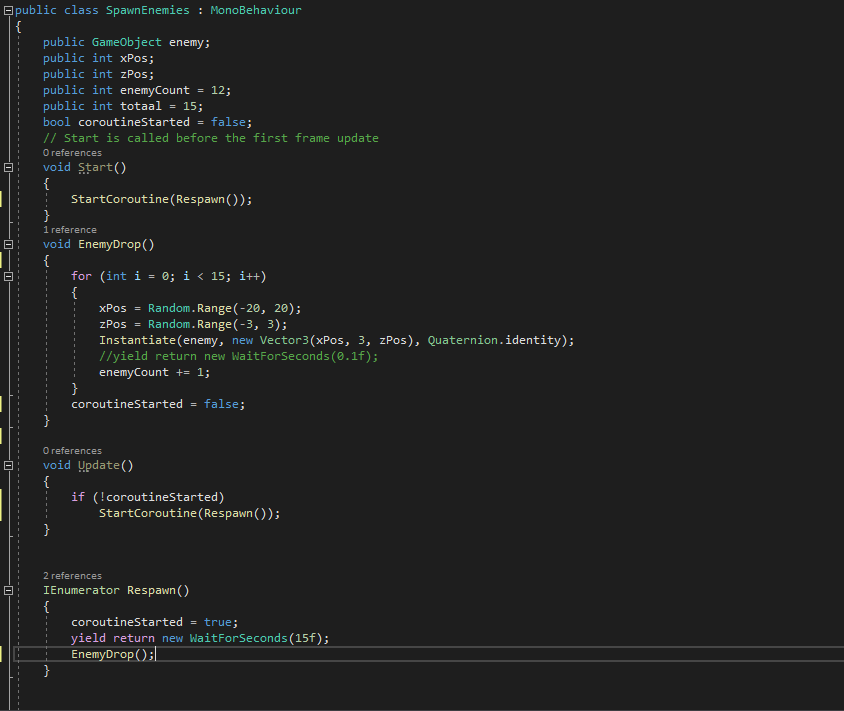
Via de animation kan je een animatie creëren voor een object (in de foto ziet u dat we een animatie kunnen maken voor de canvas). Na het drukken op de knop zal u animation files moeten maken. En in die file via de record button kan u een motion opnemen van een bepaalde tijdsduur. De motion kan gemaakt worden door middel van het object te slepen.



(dit is een deur in level1 die een animatie heeft.)

De code is hetzelfde zoals we bij u video's hebben gezien.

Enemies Laten spawnen:



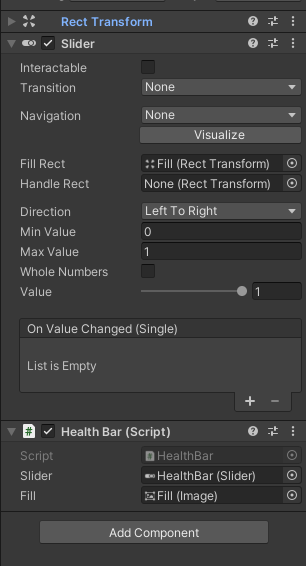
De StartCoroutine() functie werkt samen met de “yield return new WaitForSeconds();” dit zorgt in onze code voor een interval zodat er na een X aantal seconden enemies spawnen.

StartCoroutine is een methode die je nodig hebt om IEnumerator te kunnen oproepen.

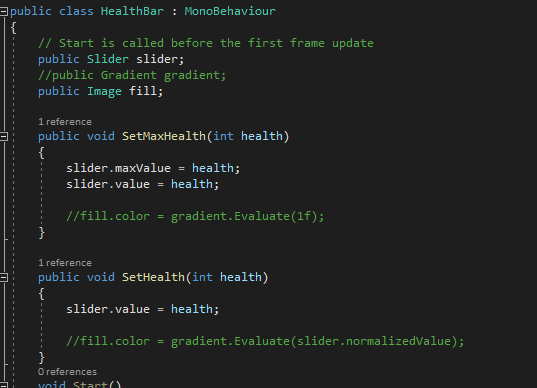
## 

## Healthbar:

Voor de health bar te animeren hebben we een slider gemaakt.



De value van de slider kunnen we makkelijk aanpassen via de Health Bar script:



via de methodes kunnen wij de value aanpassen deze script kunnen wij dan in andere scripts oproepen. Wanneer bijvoorbeeld de held omlaag gaat dat de slider zijn value ook omlaag gaat.

## PlayerMovement:

Deze code bevindt zich in de “PlayerController.cs” script.

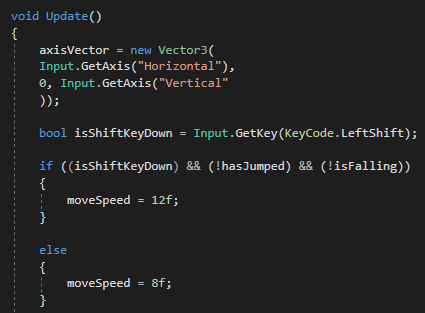
if (rb.velocity.y > 0.1):

Hierin geraken we wanneer we op “Jump” klik (de laatste if voert dit uit). Deze lijn met rb.velocity+= … voegt een betere gravity feel bij jumping of falling.

Variabel hitGround tot en met de if statement met Physics.Raycast:

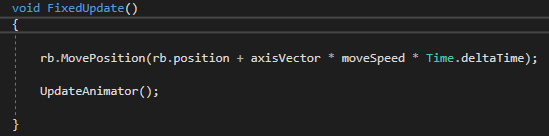
Hier voeren we een Raycast uit tot op de grond om te detecteren of de speler een object onder hem heeft of niet. Als dit waar is kunnen we terug jumpen.

We hebben dit gemaakt met Raycasting omdat het beter werkt met objecten die geen cubes zijn. We hadden ook ondervonden dat we zonder deze Raycasting functie niet konden jumpen terwijl we op MeshColliders zitten.

De laatste functie is zelfsprekend, wordt gebruikt voor jumping en nog logica voor wanneer je niet mag jumpen.

In de FixedUpdate() (wordt na Update opgeroept, basisfunctie van C#) voeren we dan de rb(RigidBody).MovePosition uit en hierin geven we positie van de rb +

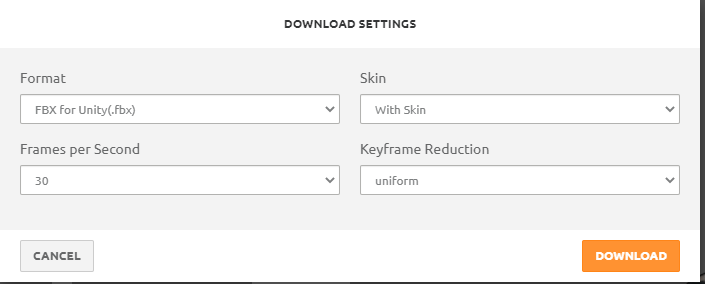
de axisVector (Deze pakken we in de eerste lijn van de update, nodig voor ZQSD) \* moveSpeed \* Time.deltaTime om de movement positie te veranderen.



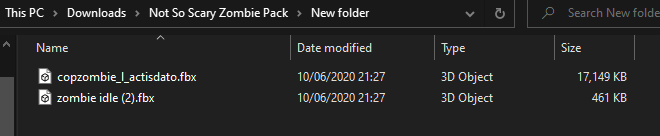
## Player model + Animatie en Enemy model + animatie:

De modellen en animaties hebben we uit Mixamo.com. Hierin kunnen wij de modellen downloaden en importeren in Unity. Hiervoor moeten we wel eerst paar dinges voor aanpassen:

In de “Characters” Tab kunt u kiezen voor welke model u wilt (hier is er een variatie van 100+ 3D character models die compatibel zijn met Unity. Je selecteert een character model en dan kiest u de animatie naar uw keuze. In de animatie settings onder download doet u dit:



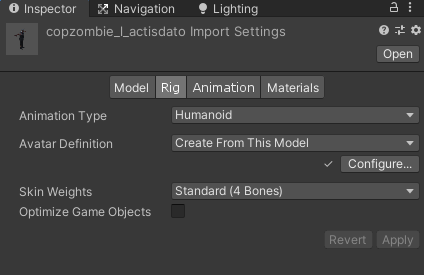
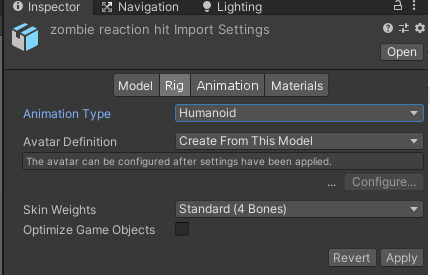
Hiermee krijgt u een Zip bestand met FBX bestanden:



Deze FBX bestanden kan u zo maar slepen in een prefab folder in Unity. Om die te gebruiken (met de juiste textures) moet u wel een paar settings aanpassen:

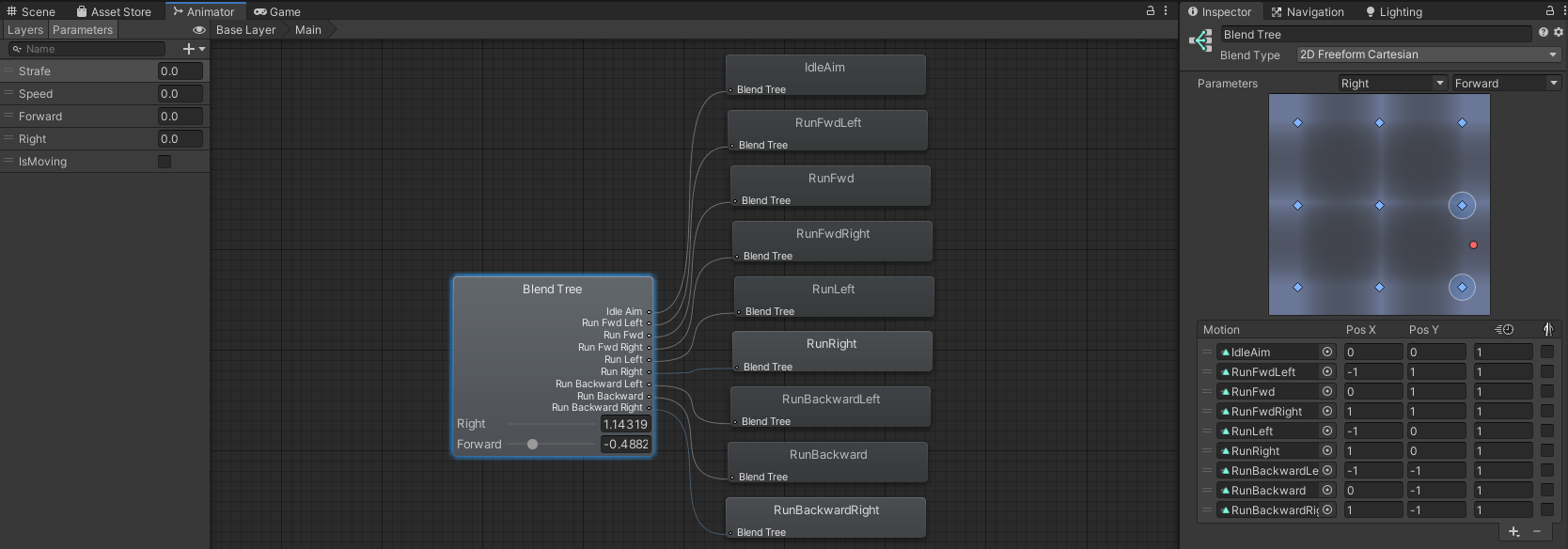
De (Model) Rig Animation Type op Humanoid zetten en Apply klikken

Voor de Animatie doet u het zelfde:



## 

Blendtree (Voor een Top Down Shooter, gebruikt voor de Player)

****

Deze blendtree hebben we met een guide gemaakt:

Enkel de Right, Forward en IsMoving parameters worden gebruikt. De BlendType moet op 2D Freeform Cartesian staan en de Animatie motions (Idle, RunFwdLeft, …) moeten moeten in die volgorde met de juiste Pos X en Pos Y gelegd worden om die juist aan te spreken in code.

Code:

Deze code spreekt de Pos X en Pos Y waardes aan en gebaseerd op waar de speler nu kijkt en doet een mix van de 2 motions waar de waardes het dichts bij op staan. In de animator zet je dan de isMoving op true om te bewegen wanneer de lengte van axisVector langer is dan 0. Zo wordt de boolean parameter van de BlendTree op true gezet om de animatie te beginnen.

De PosX en PosY veranderen de positie van de rode bolletje. De blauwe bolletjes beslissen welke 1 of 2 animaties gespeeld/gemixt moeten worden

## 

## Muren invisible zetten:

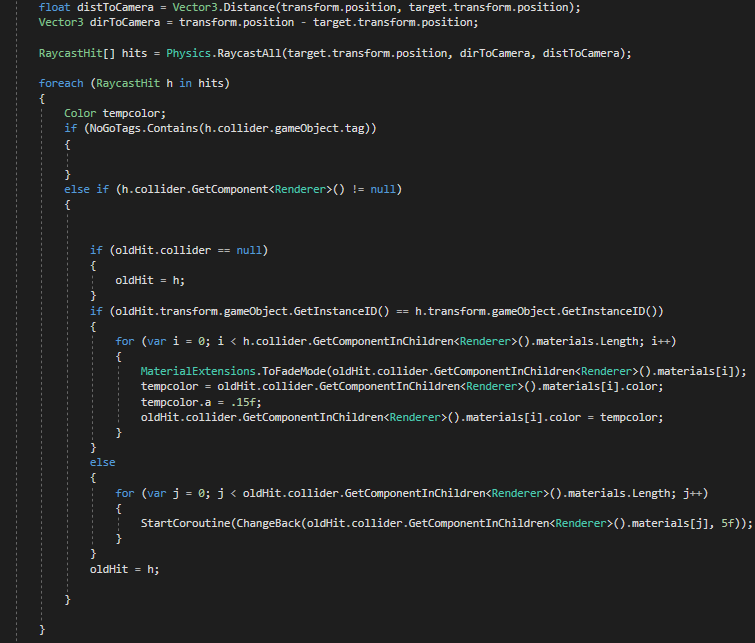
“CameraControl.cs”

Dit is een belangrijke functie van Top down shooters omdat jouw Player niet altijd zichtbaar blijft (door objecten, muren, bruggen…) die tussen de player en de Camera komen. Daarom hebben we een functie hiervoor voorzien om deze muren onzichtbaar te zetten door gebruik te maken van Raycasting. We zetten de objecten onzichtbaar door de rendermode van de Material ervan aan te passen van “Opaque” naar “Fade”.. De code van onze camera zelf gaan we niet uitleggen want die is een vrij simpel camera die de player volgt en in/uitgezoomd kan worden

Code:

We hebben hier een paar Tags die niet onzichtbaar mogen worden, anders crasht de spel. Deze zeten we in een array. Voor de “oldHit” geven we straks meer uitleg over.

De RaycastHit[] hits is een Array van Raycasts die alle objecten bewaart die tussen de speler en de camera komen.

In een foreach hebben we een tempcolor variabel gemaakt die we later terug oproepen (om de kleuren van de material te veranderen). Daar hebben we een if statement die kijkt of de “hits” 1 van de “NoGoTags” heeft geraakt, zo wel mogen we niets uitvoeren.

Als dat niet zo is, voeren wij 3 if statements uit die ongeveer werken als **“Start”, “During” & “End”.**

In de **“Start”** kijken we of de “oldHit” variabel op null staat of niet. Dit is belangrijk wanneer het spel opstart, want dan bestaat er nog geen object waar de raycast door heeft gepasseerd.

In de **“During”** voeren we de meeste logica uit. Hier geraken we erin als er 2 Raycasts door een object is geraakt (old ==huidige) (dit gebeurt vrij snel bij een Update functie, dus over delays maken we geen zorgen). We spreken alle materialen aan die zich bevinden in de gameObject (met een for loop) en hiervan veranderen we de rendermode van Opaque naar Fade. De kleur van deze materialen bewaren we op de tempcolor en veranderen we de alpha (onzichtbaarheid) naar 0.15f. De kleur van de materialen van “oldHit” veranderen we dan naar de tempcolor. Zo geraken ze onzichtbaar.

In de **“End”** (de else-statement) roepen we de materialen van de oldHit variabels terug als de oldHit!=huidige (dit is meestal wanneer de speler van plaats verandert) om de rendermode na 5 seconden terug te veranderen naar Opaque (niet meer onzichtbaar want alpha werkt hier niet op). Helemaal op het einde doen we dan oldHit is gelijk aan de Huidige hit om terug in de “During” te kunnen geraken.

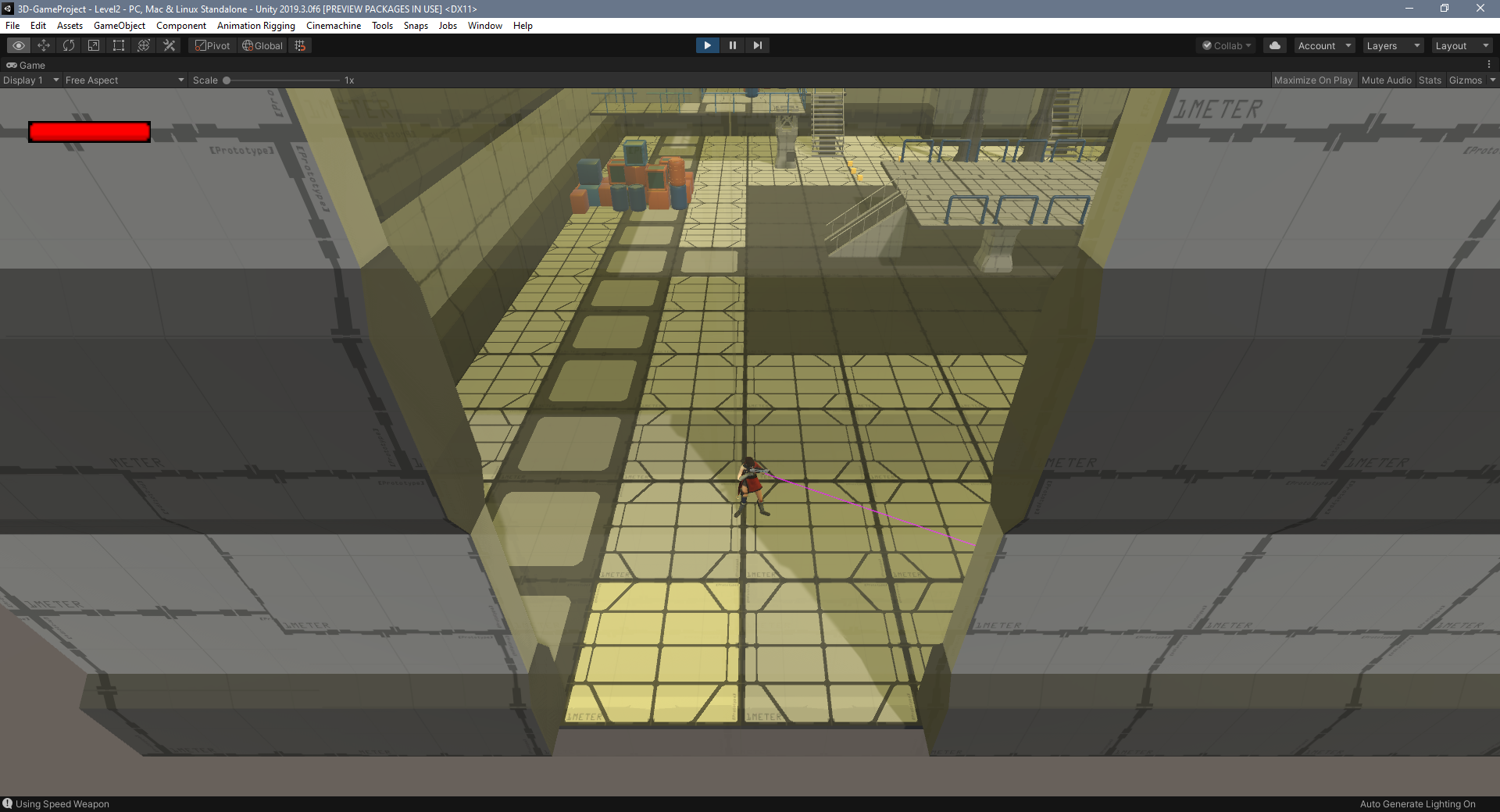
## De code voor MaterialExtensions: Dit is een stuk code die we uit het internet hebben die gebruikt wordt om de render mode van materialen op Opaque of Fade te zetten.

De functie met delay (ChangeBack) die de material terug omzet naar Opaque na 5 seconden.

## 

## 

Dit is het resultaat: (4 muren die onzichtbaar zijn om speler zichtbaar te laten).



## 

## Shooting: Weapon

De eerste if is een check die niet toelaat om te schieten als er een canvas voor de spel komt (zoals inventory).

De if en else statements pakken de snelheid van de wapen die we momenteel aan hebben. Als we geen wapen aan hebben schieten we met een snelheid van 2 bullets/second.

De laatste if statement zorgt voor de shooting, de “shoot” boolean wordt true door middel van een delay om de FireRate (shoot snelheid) in te stellen. Wanneer eerste muisknop ingedrukt gehouden wordt wordt er deze functie opgeroepen + sound effect speelt:

In de Shoot functie spawnen we een bullet met de Instanstiate functie, hier geven we de kogelprefab (die ook een aparte script heeft voor collisie checks). De firePoint is een Transform die we aan de Player hebben gebonden (punt waar kogels uit komen).

De shoot bolean, StartCoroutine + de ShootDelay IEnumerator zorgt voor de FireRate zoals we hadden uitgelegd.

## In de 2de foto hebben we de “FirePoint” aangeduid met de “Move” editor van Unity (rood, groen, blauw). Deze hebben we voor de wapen gezet zodat het zo accuraat mogelijk is. De kogels spawnen hieruit.



## 

## Shooting: Bullets

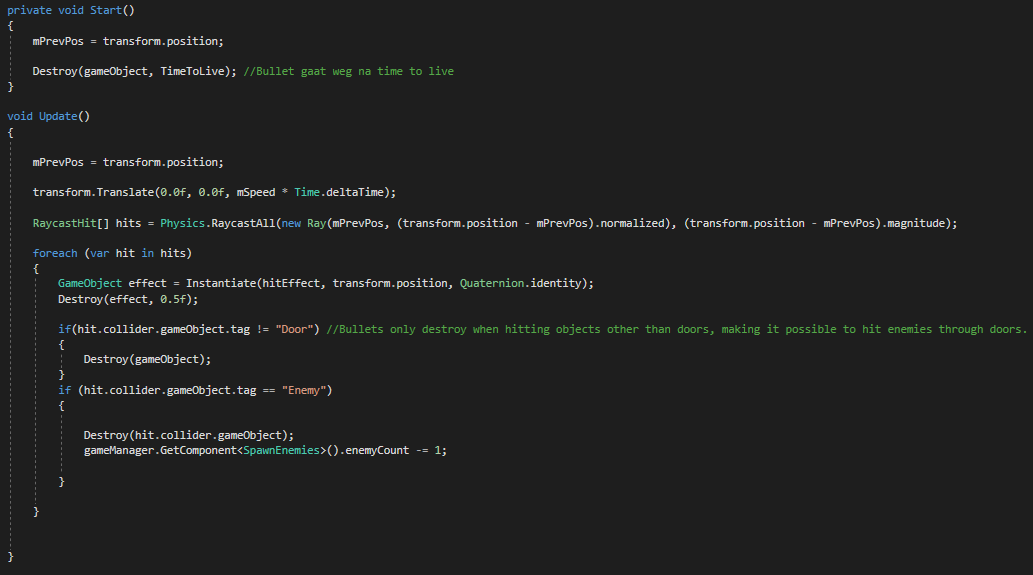
## Bij de bullets werken we niet met de traditionele box/capsule collider. Na onderzoek hebben we ontdekt dat dit kwa performantie voor een kleine downgrade kan zorgen (20 box colliders spawnen per seconde) en ze zijn niet altijd heel accuraat. Daarom hebben we onderzocht en vonden we voor een manier dat met Raycasts werkt. Deze hebben we dan toegepast met de code die we al hadden om te schieten (die toen nog werkte met colliders). Deze tekent intern een lijn tussen het schietpunt en het huidige positie/eindpunt waar dat die met een object raakt en detecteert de objecten. Zo kan het nog accuraat blijven als de bullet bijvoorbeeld heel snel is.

In de start methode zorgen we ervoor dat de bullet gaat verdwijnen na 5 seconden (TimeToLive) als die niets raakt.

In de update zetten we de schietpunt als mPrevPos (previous position) en laten we de kogel bewegen met transform.translate.

Met de raycast doen we wat we hierboven hebben uitgelegd met die lijn tekenen tussen startpunt en huidige positie/eind punt.

In de foreach instantieren we de effect die wordt gespeeld wanneer een kogel iets of iemand raakt endan verwijderen we die effect zodat het niet blijft loopen.

Voor de collisie logica gebruiken we tags om bijvoorbeeld interactie te doen met wat we schieten. Zo kunnen we door de deur schieten (want hij wordt niet “destroyed” wanneer hij een deur raakt). Zo kunnen we ook de enemy aanspreken en de gameObject Destroy-en wanneer die geraakt wordt. Hier wouden we eigenlijk nog health geven aan deze enemy en deze verminderen wanneer hij geschoten werd en zo vermoorden, maar daar zijn we niet geraakt.

## Inventory guides (van Internet):

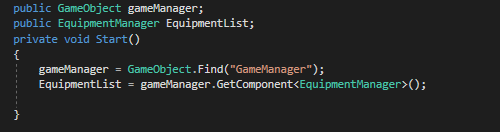
We hebben deze guides gevolgd om een inventory te maken:

<https://youtu.be/w6_fetj9PIw> “INVENTORY UI - Making an RPG in Unity (E05) Door Brackeys

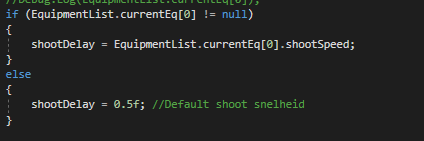
<https://youtu.be/YLhj7SfaxSE> “INVENTORY CODE - Making an RPG in Unity (E06) Door Brackeys”

<https://youtu.be/d9oLS5hy0zU> “EQUIPMENT - Making an RPG in Unity (E07) Door Brackeys”

Deze horen bij de wapens Equippen en Unequippen. De wapen parameters aanspreken zitten hier niet tussen, dat hebben wij zelf ontdekt. Wij doen een referentie aan de equipment manager in onze shooting script



En van hieruit pakken wij de stats van de eerste Equipment slot (de Wapen) en veranderen onze Wapen shoot speed ermee. Als er geen wapen “GeEquipt” is, is de shootspeed hardcoded 2 Bullets per seconde (0.5f);



## **Contribution to the project:**

## Ayoub Aouraghe:

## · Map design lighting…

## · Deuren animatie en interactie

## · Enemy script

## · EnemySpawn Script

## · Level veranderen

· UI healthbar, exit game, informatiepaneel

## Gürsel Gökce:

## · Muren en/of Objecten Transparant maken

## · Player laten bewegen animatie en schieten

## · Enemy animatie

## · Inventory

## · Camera van de player en de player ronddraaien door middel van muis locatie

## · Ui main menu, wapen informatie

## 