Programmeren van een computergame

Unity

IT factory

Wout Mergaerts

Academiejaar 2017-2018

Campus Geel, Kleinhoefstraat 4, BE-2440 Geel

LID VAN ASSOCIATIE KU LEUVEN

THOMAS MORE

INHOUDSTAFEL

INHOUDSTAFEL **3**

1 UNITY **5**

1.1 Wat is Unity? **5**
1.2 Enkele games die gemaakt zijn met Unity **6**
1.3 Hoe installeer je Unity? **6**
1.4 Enkele handige bronnen voor het leren van Unity **7**
1.4.1 Unity zelf 7
1.4.2 YouTube 7
1.4.3 Gewoon Google 8
1.4.4 Ik 8

2 PROJECT KLAARMAKEN **9**

2.1 Voorbereiding **9**
2.1.1 De speler 9
2.1.1.1 Het model 9
2.1.1.2 De animaties 9
2.1.1.3 De character controller 9
2.1.1.4 Het Player Script 10
2.1.1.5 De camera 13
2.1.1.6 De UI 14
2.1.2 Dogoo 15
2.1.2.1 Enemy AI Script 15
2.1.3 Rupee 18
2.1.3.1 De box collider 18
2.1.3.2 Het Rupee Script 19
2.1.4 Dim Crystal 19
2.1.4.1 Geluidsbron 19
2.1.4.2 Dim Crystal Script 20
2.1.5 Safezone Shield 20
2.1.5.1 Het Damage Trigger Script 20
2.1.6 GameInfo 21
2.1.7 Loading Screen 21
2.1.7.1 Load Scene Script 22
2.2 Prefabs maken **22**

3 JUMPPAD **23**

3.1 Prefabs **23**
3.1.1 Jumppad 23
3.1.2 Jumppad platform 23
3.2 Script **24**
3.3 Uittesten **24**

4 AUTOMATISCH LEVEL GENEREREN **25**

4.1 Platformen genereren **25**
4.1.1 Script 25
4.1.2 Script in de game zetten 26
4.2 Meer soorten platformen **27**
4.2.1 Script 27
4.2.2 Script testen 28
4.3 Nog meer soorten platformen **29**
4.3.1 Script 29
4.3.2 Script testen 32
4.4 Willekeurige posities **32**
4.4.1 Script 32
4.4.2 Script testen 34

5 MENU'S 35

5.1 Hoofdmenu 35
5.1.1 Het uitzicht 35
5.1.1.1 Achtergrond 35
5.1.1.2 Titel 36
5.1.1.3 Knoppen 36
5.1.2 Script 38
5.1.3 Script toevoegen aan het menu 39
5.1.4 GameInfo 40
5.1.4.1 GameInfo 40
5.1.4.2 LevelManager 40
5.2 Pauzemenu 41
5.2.1 Het uitzicht 41
5.2.2 Script 41

6 BESLUIT 43

1 UNITY

1.1 Wat is Unity?

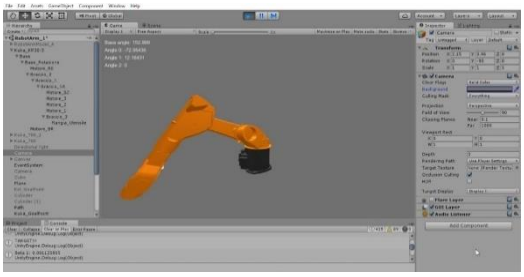
Unity is een cross-platform game-engine dat ontwikkeld is door het Deens-Amerikaanse bedrijf Unity Technologies.

- *Game-engine:* Een programma dat helpt bij het ontwikkelen van computergames. Deze maakt verschillende onderdelen van het maken van computergames veel gemakkelijker. Enkele voorbeelden hiervan zijn bijvoorbeeld zwaartekracht simuleren en renderen (dingen op het scherm laten zien).

- *Cross-platform:* Dit programma kan gebruikt worden op of kan games maken voor meerdere soorten apparaten (platformen). Bij Unity gaat dit over pc, Mac, consoles, Android, Iphones, websites en de WII U.

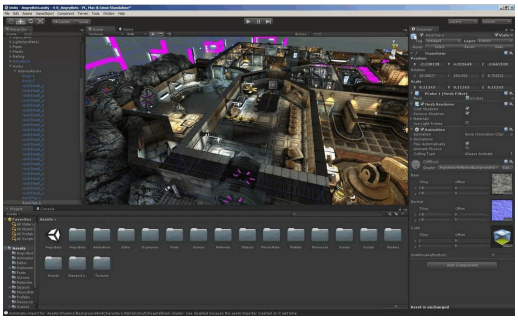
Unity bestaat op dit moment uit 3 Versies. Unity Personal, Unity Plus en Unity Pro.

- *Personal:* Zoals de naam al zegt, dit is vooral voor persoonlijk gebruik. Als bedrijf kan je deze versie ook gebruiken zolang je games onder de €100 000 euro per jaar verdienen. Deze is de enige versie van Unity dat gratis is. Tenzij je heel serieus met Unity bezig bent en professionele games wil uitbrengen, zou ik altijd voor deze versie kiezen.



Afbeelding 1 - Unity Personal

- *Plus:* Deze versie kost €25 per maand, maar komt met enkele voordelen. Zo kan je eerst en vooral tot €200 000 met je games verdienen. Vervolgens krijg je toegang tot verschillende exclusieve dingen zoals bijvoorbeeld cloud opslag, contact met Unity experts, … . Maar het allerbelangrijkste, je krijgt toegang tot het Unity Dark Theme (je kan dit ook op een stiekeme manier in Unity Personal doen, indien je dit wil weten kan je mij een mailtje sturen)! Deze versie kan nuttig zijn voor hobbyisten die ook games willen uitbrengen.



Afbeelding 2 - Unity Dark Theme (Plus & Pro)

- *Pro:* Deze versie kost €125 per maand en is enkel bedoeld voor professionele bedrijven. Met deze versie krijg je nog extra toegang tot verschillende toepassingen en daarbovenop nog €800 in Unity Asset Store items. Op deze versie staat ook geen limiet op hoeveel je kan verdienen met je games.

Voor een volledige vergelijking kan je hier klikken.

In Unity kan je ook in verschillende talen programmeren. Zo kan ja zowel in C# (wat we in deze cursus gaan doen) als in UnityScript programmeren. UnityScript is een taal die heel hard lijkt op JavaScript, maar het heeft ook enkele eigenschappen van andere talen.

1.2 Enkele games die gemaakt zijn met Unity

Afbeelding 3 - Hearthstone

Afbeelding 4 - Firewatch

Afbeelding 5 - Rust

Afbeelding 6 - Hollow Knight

Afbeelding 7 - Cuphead

Afbeelding 8 - Monument Valley 2

Meer games die gemaakt zijn met Unity kan je hier vinden.

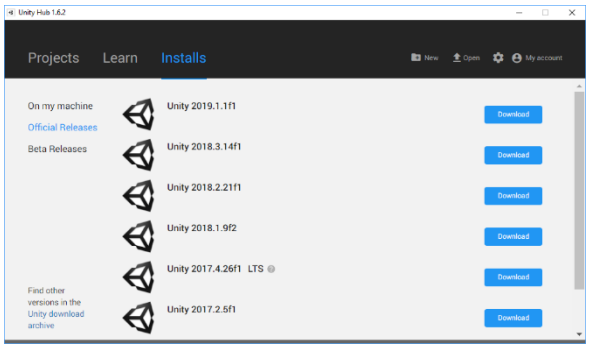
1.3 Hoe installeer je Unity?

Stap 1: Surf naar de Unity website. Indien je wilt, kan je daar eerst rondkijken naar wat je allemaal zou kunnen doen met Unity.

Stap 2: Ga naar het "Products" gedeelte. Hier kan je kiezen welke versie van Unity je kan downloaden. Omdat jij en ik waarschijnlijk nog geen games gaan uitbrengen, kiezen we hier voor Personal. Daarna klik je op "Try Personal" en accepteer je de voorwaarden om vervolgens op de download knop te drukken.

Stap 3: Zoek naar je gedownloade bestand (op dit moment noemt dit UnityHubSetup.exe. Open dit bestand en volg de verschillende stappen die gevraagd worden.

Stap 4: Nadat Unity Hub is geïnstalleerd kan je dit openen. Na het openen wordt gevraagd of je Unity wil installeren. Indien dit niet gebeurde, kan je ook naar "Installs" gaan en daar een Unity versie kiezen.



Afbeelding 9 - Installs

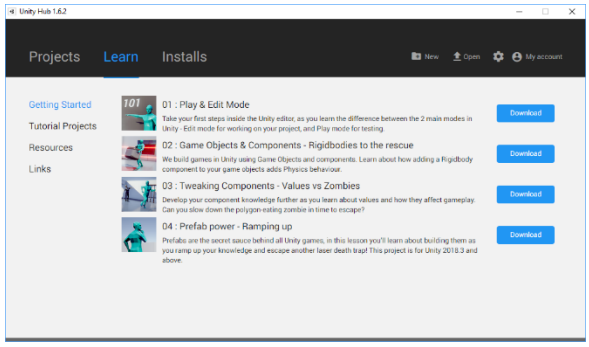
Stap 4.5: Indien nodig wordt er ook gevraagd om Visual Studio te installeren. Indien je gaat werken in C# (zoals in deze cursus) is dit een zeer handig programma om te helpen met programmeren. Zelf gebruik ik dit ook voor zowel Unity als voor school.

Stap 5: Nadat al het nodig geïnstalleerd is, kan je je eerste project aanmaken of eventueel verder werken aan al bestaande projecten.

1.4 Enkele handige bronnen voor het leren van Unity

1.4.1 Unity zelf

Unity heeft zelf zeer veel handige lessen en hulpmiddelen voor het aanleren van verschillende aspecten van Unity. Zo is er eerst al op hun website een volledig deel gericht daarop. Hier kan je alles vinden van online lessen tot certificaten die je kan halen tot workshops die je kan volgen en nog veel meer. Ook vind je hier de documentatie met beschrijvingen van alle onderdelen van Unity. Ten tweede zijn er in de Unity Hub app ook enkele projecten die je kan downloaden in de "Learn" tab. Deze zijn praktische lessen om de basis van Unity onder de knie te krijgen. Ze zijn ook meestal begeleid door een YouTube afspeellijst met alle nodige uitleg waar je mee de les kan volgen.



Afbeelding 10 - Learn

1.4.2 YouTube

Zolang je een beetje Engels kan, vind je op YouTube duizenden en duizenden filmpjes met lessen voor het aanleren van verschillende Unity onderdelen. Enkele van mijn favorieten zijn:

- Brackeys: Hij geeft zeer enthousiast les over zowel basisonderdelen als meer gevorderde lessen. Alles wordt zeer duidelijk uitgelegd in korte filmpjes. Af en toe zijn is er ook wat afwisseling met grappige filmpjes of vlogs.

- Blackthornprod: Hij is nog maar 18 jaar maar hij maakt al sinds enkele jaren games in Unity. Ook hij maakt enthousiaste lessen over verschillende basis en moeilijkere onderdelen in Unity. Deze worden allemaal in een leuke cartoon stijl gemaakt. Hij heeft ook een leuke Discord server waar iedereen mekaar helpt en waar hij zelf ook actief meehelpt en praat met iedereen.

- Thomas Brush: Een iets professionelere lesgever. Zijn filmpjes duren iets langer maar dit komt ook omdat hij vaak livestreams doet. Hij geeft ook tips over verschillende andere aspecten die bij het ontwikkelen van games komen.

- Dani: Hij maakt geen lessen, maar in plaats daarvan maakt hij elke week een vlog over het maken van zijn game Off The Sticks. Met deze filmpjes kan je zien hoe het ontwikkelen met games kan gecombineerd worden met andere dingen zoals bijvoorbeeld school.

1.4.3 Gewoon Google

Natuurlijk na je al het bovenstaande ook gewoon op Google vinden. Daarbovenop zijn er nog gigantisch veel lessen, gratis of betalend, beschikbaar op duizenden sites op het internet. Daarbovenop kan je ook op verschillende forums vragen stellen indien je echt niet vind wat je zoekt.

1.4.4 Ik

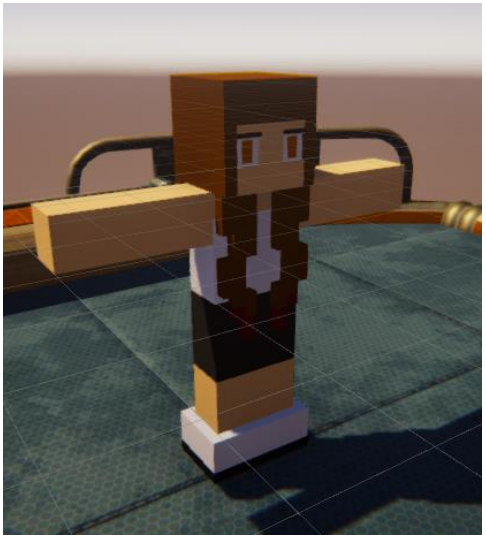
Ik ben geen expert in Unity zoals alle vorige opties, maar indien je vragen hebt over Unity of over de games die je maakt of wil maken mag je mij altijd een mailtje sturen. Ik zal dan zo snel als ik kan en zo goed als ik kan proberen te antwoorden.

2 PROJECT KLAARMAKEN

2.1 Voorbereiding

In de rest van deze cursus ga ik er van uit dat je het originele project (Hyperblock) kent. Indien dit niet, of niet helemaal, het geval is ga ik hier nog eerst eens even over alle belangrijke onderdelen. Indien enkele details niet duidelijk zijn, kan je mij altijd een mailtje sturen of een van de hulpmiddelen uit 1.4 gebruiken.

2.1.1 De speler



Afbeelding 11 - Zoey

De speler is gemaakt, geanimeerd en gemodelleerd door Diaz en werd Zoey genoemd. Dit is het belangrijkste, en waarschijnlijk ook meest uitgebreide, onderdeel van de game. Zoey bestaat uit enkele belangrijke onderdelen:

- Het model
- De animaties
- De character controller
- Het Player Script
- De camera
- De UI

2.1.1.1 Het model

Elk object in Unity heeft een model nodig. Dit model zorgt voor het uitzicht van deze objecten. Zo heeft de vloer onder Zoey ook een model, net als de platformen in de rest van de game. Deze worden door Unity zichtbaar gemaakt door een Renderer.

2.1.1.2 De animaties

Zonder animaties zou een game maar saai zijn. Zoey heeft bijvoorbeeld een wandel- en spring animatie. De Dogoo's (zichtbaar in de achtergrond van afbeelding 12) hebben ook een animatie elke keer als ze springen.

Afbeelding 12 – Wandelen

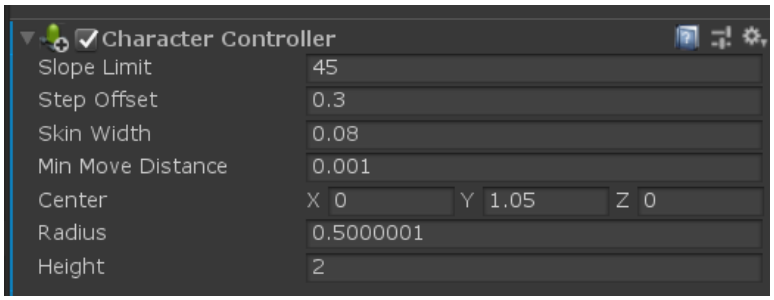
Afbeelding 13 – Springen/vallen

2.1.1.3 De character controller

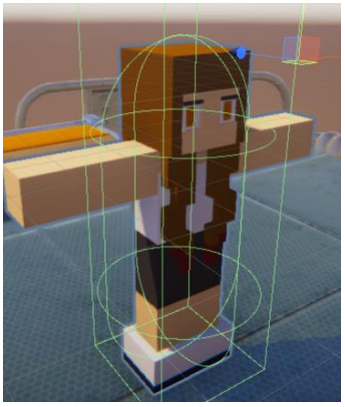
Een character controller is een hulpmiddel om de bewegingen van je speler, hier bijvoorbeeld Zoey, makkelijker te maken in Unity. Hiermee kan je Zoey makkelijk laten rond bewegen zonder al te veel code. Vooraleerst dit gebeurt moet je wel enkele dingen configureren:

- *Slope limit:* De limiet van de helling waar je kan oplopen, uitgedrukt in graden.
- *Step offset:* De stapafstand, uitgedrukt in meters.
- *Skin Width:* De dikte van de "huid". Deze wordt gebruikt om te kijken of er contact is met andere objecten.

- *Min Move Distance:* De minimale afstand die er moet gedaan worden indien men beweegt.
- *Center:* Het middelpunt van de speler.
- *Radius:* De straal van de cirkel waarin de speler zich bevind. Deze wordt ook gebruikt voor het zoeken naar contactpunten.
- *Height:* De hoogte van de speler.



Afbeelding 14 - Character Controller



Afbeelding 15 – Radius en Height zorgen voor een koker rond Zoey

2.1.1.4 Het Player Script

Het Player Script zorgt voor de bewegingen van Zoey. Dit script is gemaakt in C#, net zoals alle andere scripts waarover we het gaan hebben in de rest van de cursus.

Zoals bijna elk C# programma, begint ook dit script met enkele variabelen. Indien deze niet in het script zelf gedefinieerd worden, worden deze in Unity zelf manueel gedefinieerd (zoals te zien is in afbeelding 14).

- *PlayerCC:* De character controller van de speler. Zie 2.1.1.3.
- *CharAM:* De animator van de speler. Zie 2.1.1.2.
- *DEBUGMODE:* Een boolean (kan waar of niet waar zijn) om te kijken of de game in debug mode is.
- *CameraRoot:* De plaats en richting van de camera die de speler volgt.
- *MovementSpeed:* De bewegingssnelheid waarmee de speler kan rondlopen.
- *JumpForce:* De kracht waarmee de speler kan springen.
- *Fallingscene:* Om te kijken of de speler in de valscène is of niet.
- *YSpeed:* De snelheid van de speler in de Y-richting (naar boven of onder).

```
CharacterController PlayerCC;
Animator CharAM;

public bool DEBUGMODE = false;

public Transform CameraRoot;

public float MovementSpeed = 2, JumpForce = 3;
public bool FallingScene = false;

private float YSpeed;
```

Afbeelding 16 - Variabelen

Vervolgens is er de Awake-functie. Deze functie is ingebouwd in Unity en wordt enkel opgeroepen in het begin als de game begint en het script voor de eerste keer opgeroepen wordt. In deze functie worden, net zoals hier, meestal enkele hoofdzaken gedefinieerd die gebruikt worden in de rest van het script. Zo worden hier de character controller en animator opgehaald uit het spelerobject, worden enkele variabelen in de GameInfo klasse (meer hierover later in de cursus) gedefinieerd en kijkt het script na of Zoey op de grond staat.

```
private void Awake()
{
    PlayerCC = GetComponent<CharacterController>();
    CharAM = GetComponent<Animator>();

    GameInfo.dmgCoolDown = 0;
    GameInfo.health = 10 * GameInfo.lvl;

    if (!FallingScene)
    {
        PlayerCC.Move(new Vector3(0, Physics.gravity.y, 0));
        Parenting();
        CharAM.SetBool("IsGrounded", true);
    }
}
```

Afbeelding 17 - Awake

Net zoals een film bestaat een game uit verschillende frames die achter mekaar worden afgespeeld. De Update functie is, net als de Awake functie, een ingebouwde functie van Unity die bij elke frame wordt opgeroepen. Hierbij worden 3 condities gesteld:

```
private void Update()
{
    if (!FallingScene && !GameInfo.isInCutScene)
    {
        Movement();
        Parenting();
        VarUpdate();
    }
    else if (FallingScene && !GameInfo.isInCutScene)
    {
    }

    if(transform.position.y <= -15f || GameInfo.health == 0 && GameInfo.dmgCoolDown == 0)
    {
        SceneManager.LoadScene(0);
    }
}
```

Afbeelding 18 - Update

- *De speler is niet in de valscène of in een tussenscène:* Namelijk als de speler "leeft", dan worden 3 functies uitgevoerd, namelijk Movement, Parenting en VarUpdate. Meer hierover later in de cursus.
- *Als de vorige conditie niet voldaan is en de speler is in de valscène maar niet in een andere scène:* Doe niets
- *Als de speler onder een bepaalde hoogte of zonder levens zit:* Laad het level opnieuw, ofwel "doodgaan".

Zoals in de vorige functie gezien is, indien de speler nog leeft wordt de VarUpdate functie opgeroepen. Deze is een zelfgemaakte functie die ervoor zorgt dat er enkele variabelen worden aangepast.

```
void VarUpdate()
{
    GameInfo.dmgCoolDown = Mathf.Clamp(GameInfo.dmgCoolDown - Time.deltaTime, 0f, 9999f);

    if (GameInfo.exp >= 100 * GameInfo.lvl)
    {
        GameInfo.exp -= 100 * GameInfo.lvl;
        GameInfo.lvl++;
        GameInfo.health += 10;
    }
}
```

Afbeelding 19 - VarUpdate

- *DmgCooldown:* Zoals elke cooldown wordt deze per seconde lager. De Clamp functie zorgt ervoor dat de waarde hiervan tussen 0 en 9999 blijft.
- *Indien er meer dan 100 exp verdient is:* verhoog het level en de levens van de speler.

De Parenting functie zend een straal (een Raycast in Unity) naar beneden uit om na te kijken of de speler op iets loopt. Hierbij is *transform.position + transform.up (één hoger dan de positie van de speler)* het beginpunt, *-transform.up (naar beneden)* de richting, *out hit* de variabelen als er iets geraakt wordt en *1.1f* de lengte van de straal.

```
void Parenting()
{
    RaycastHit hit;

    if (Physics.Raycast(transform.position + transform.up, -transform.up, out hit, 1.1f))
    {
        transform.parent.parent = hit.transform;
    }
    else
    {
        transform.parent.parent = null;
    }
}
```

Afbeelding 20 - Parenting

De Movement functie bestaat uit verschillende onderdelen. Eerst een vooral worden ook hierin enkele variabelen gedefinieerd. Deze variabelen zijn enkele beschikbaar binnen deze functie.

```
void Movement()
{
    bool isGroundedNow = PlayerCC.isGrounded;
    float XSpeed = 0;
    float ZSpeed = 0;
```

Afbeelding 21 – Movement variabelen

- isGroundedNow: Zegt of de speler momenteel op de grond staat.
- XSpeed: De zijwaartse snelheid van de speler.
- ZSpeed: De voorwaartse snelheid van de speler.

Als de Speler op de grond staat kan volgend onderdeel uitgevoerd wordt. Hierin wordt nagekeken of de speler op de spring-knop duwt (de spatiebalk). Indien dit gebeurd wordt isGroundedNow op "niet waar" gezet en wordt de verticale snelheid gelijk aan de springkracht. Indien niet wordt de verticale snelheid gelijk aan de zwaartekracht, waardoor de speler valt als deze niet op de grond staat.

```
if (isGroundedNow)
{
    if (Input.GetButtonDown("Jump"))
    {
        isGroundedNow = false;
        YSpeed = JumpForce;
    }
    else
    {
        YSpeed = Physics.gravity.y;
    }
}
else
{
    YSpeed += Physics.gravity.y * Time.deltaTime;
}
```

Afbeelding 22 - Beweging

Indien de Fire2-knop niet wordt ingeduwd of de speler is niet in Debugmode, wordt onderstaand stuk code uitgevoerd.

Hierin wordt eerst gecontroleerd of de knoppen voor het rondbewegen ingeduwt worden. Indien dit gebeurd, wordt de ZSpeed aangepast zodat de speler kan rondbewegen. *PlayerCC.Move* laat dan de speler rondbewegen in de juiste richting met de juiste snelheden. Hierna wordt de verticale snelheid op 0 gezet indien de speler op de grond staat. Tot slotte wordt ook nog de richting naarwaar de speler kijkt bepaald.

```
if (!Input.GetButton("Fire2") || !DEBUGMODE)
{
    CharAM.SetBool("IsFlying", false);

    if (Input.GetAxis("Vertical") > 0)
        ZSpeed += Input.GetAxis("Vertical");

    if (Input.GetAxis("Vertical") < 0)
        ZSpeed += -Input.GetAxis("Vertical");

    if (Input.GetAxis("Horizontal") > 0)
        ZSpeed += Input.GetAxis("Horizontal");

    if (Input.GetAxis("Horizontal") < 0)
        ZSpeed += -Input.GetAxis("Horizontal");

    ZSpeed = Mathf.Clamp(ZSpeed, 0, 1);

    CharAM.SetFloat("ZSpeed", ZSpeed);

    PlayerCC.Move(transform.rotation * new Vector3(XSpeed * MovementSpeed, YSpeed, ZSpeed * MovementSpeed) * Time.deltaTime);

    XSpeed = Input.GetAxisRaw("Horizontal");
    ZSpeed = Input.GetAxisRaw("Vertical");

    if (isGroundedNow)
    {
        YSpeed = 0;
    }

    CharAM.SetBool("IsGrounded", isGroundedNow);

    if (XSpeed != 0 || ZSpeed != 0)
    {
        Quaternion targetRotation = Quaternion.LookRotation(CameraRoot.rotation * new Vector3(XSpeed, 0, ZSpeed));
        transform.rotation = Quaternion.Slerp(transform.rotation, targetRotation, 0.1f);
    }
}
```

Afbeelding 23 - Bewegen deel 2

Indien de Fire2-knop wel wordt ingedrukt en de speler ook in Debugmode zit wordt er bijna exact hetzelfde gedaan. Het enige verschil is dat de speler deze keer kan vliegen, deze functie zal waarschijnlijk voor testen gemaakt zijn. Persoonlijk had ik het grote bewegings-gedeelte nog in een aparte functie gestoken aangezien er veel dezelfde code hergebruikt wordt.

```
else if (Input.GetButton("Fire2") && DEBUGMODE)
{
    CharAM.SetBool("IsFlying", true);

    if (Input.GetAxis("Vertical") > 0)
        ZSpeed += Input.GetAxis("Vertical");

    if (Input.GetAxis("Vertical") < 0)
        ZSpeed += -Input.GetAxis("Vertical");

    if (Input.GetAxis("Horizontal") > 0)
        ZSpeed += Input.GetAxis("Horizontal");

    if (Input.GetAxis("Horizontal") < 0)
        ZSpeed += -Input.GetAxis("Horizontal");

    ZSpeed = Mathf.Clamp(ZSpeed, 0, 1);

    PlayerCC.Move(transform.rotation * new Vector3(XSpeed, 0f, ZSpeed) * 30 * Time.deltaTime);

    CharAM.SetFloat("ZSpeed", ZSpeed);
    YSpeed = 0f;

    XSpeed = Input.GetAxisRaw("Horizontal");
    ZSpeed = Input.GetAxisRaw("Vertical");

    if (XSpeed != 0 || ZSpeed != 0)
    {
        Quaternion targetRotation = Quaternion.LookRotation(CameraRoot.GetChild(0).rotation * new Vector3(XSpeed, 0, ZSpeed));
        transform.rotation = Quaternion.Slerp(transform.rotation, targetRotation, 0.1f);
    }
    else
    {
        Quaternion targetRotation = Quaternion.LookRotation(CameraRoot.forward);
        transform.rotation = Quaternion.Slerp(transform.rotation, targetRotation, 0.1f);
    }
}
```

Afbeelding 24 - Bewegen deel 3

2.1.1.5 De camera

Al dit zou natuurlijk niet veel nut hebben als je niets kan zien van de game. Daarvoor hebben we in Unity een camera nodig. Deze camera staat normaal gezien gewoon stil dus hebben we hiervoor een script nodig, namelijk het Camera Script.

Net zoals in het Player Script begint ook het Camera Script eerst met enkele variabelen.

```
public Transform TransToFollow;
public Transform TransCamX;
public Transform TransCamera;
public Vector3 FollowOffset;

public float Xrot, Yrot, CamDist = 5;
```

Afbeelding 25 - Camera variabelen

- *TransToFollow:* De positie van een object dat de camera moet volgen. Hier is dat de speler.
- *TransCamX:* De locatie van de camera op de X-as.
- *FollowOffset:* De locatie die de camera moet hebben ten opzichte van de locatie die gevolgd dient te worden. Deze wordt uitgedrukt in X,Y en Z coördinaten.
- *XRot:* De rotatie rond de X-as.
- *YRot:* De rotatie rond de Y-as.
- *CamDist :* De afstand van de camera tot de speler.

Net zoals de Awake functie is er ook een Start functie ingebouwd in Unity. Deze wordt opgeroepen als het script geactiveerd wordt. Dit kan meerdere keren gebeuren als het script ook eens wordt gedesactiveerd. Deze functie wordt, net zoals Awake, meetal gebruikt om bovenstaande variabelen te definieren. In dit geval wordt het echter gebruikt om het muispijltje uit te schakelen zodat deze niet in de weg van de game zit. Ook wordt ervoor gezorgd dat de muis binnen het venster van de game blijft.

```
private void Start()
{
    Cursor.visible = false;
    Cursor.lockState = CursorLockMode.Locked;
}
```

Afbeelding 26 - Start

LateUpdate wordt net zoals Update op elke frame van de game opgeroepen. LateUpdate wordt echter pas opgeroepen na dat alle Update functies zijn uitgevoerd. De LateUpdate in het Camera Script zorgt voor veel verschillende onderdelen.

Ten eerste wordt er gecontroleerd of de F1 knop wordt ingedrukt. Indien deze ingedrukt wordt, word de muis zichtbaar of terug onzichtbaar gezet. Dit wordt vooral gebruikt om te testen.

```
//DEBUG//
if (Input.GetKeyDown(KeyCode.F1))
{
    if (Cursor.visible)
    {
        Cursor.visible = false;
        Cursor.lockState = CursorLockMode.Locked;
    }
    else
    {
        Cursor.visible = true;
        Cursor.lockState = CursorLockMode.None;
    }
}
```

Afbeelding 27 - Camera deel 1

Vervolgens wordt de rotatie van de camera bepaalt. In dit deel wordt eerst gezocht naar de locatie van de muis. Vervolgens wordt de rotatie rond de X-as (dus naar boven en beneden) gelimiteerd tot 70. Dit voorkomt dat je heel rare kijkhoeken zou hebben. Daarna wordt de Y-rotatie omgezet tot een waarde tot en met 360. Unity gebruikt een zeer moeilijk woord voor de rotatie/richting waarin een object zich bevind, namelijk Quaternion. De Eulerfuncties zorgen ervoor dat Unity weet wat de vorige waardes zijn, uitgedrukt in graden rond de X-,Y- en Z-as. Daarna wordt dit toegepast op de camera.

```
//Rotation//
Xrot -= Input.GetAxis("Mouse Y") * 2;
Yrot += Input.GetAxis("Mouse X") * 2;
Xrot = Mathf.Clamp(Xrot, -70, 70);
Yrot = Mathf.Repeat(Yrot, 360f);

Quaternion NewXRot = Quaternion.Euler(Xrot, 0, 0);
Quaternion NewYRot = Quaternion.Euler(0, Yrot, 0);

TransCamX.localRotation = NewXRot;
transform.localRotation = NewYRot;
```

Afbeelding 28 - Camera deel 2

Hierna wordt de positie van de camera veranderd naar de positie van de speler + de camera afstand.

```
//Follow//
if (TransToFollow != null)
    transform.position = TransToFollow.position + FollowOffset;
```

Afbeelding 29 - Camera deel 3

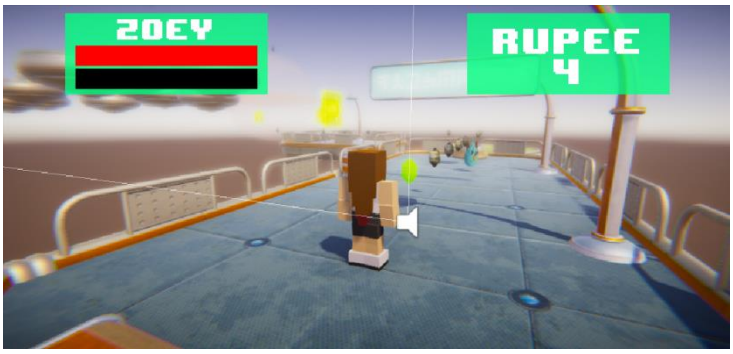
Tot slot wordt er nog gecontroleerd of er een object tussen de speler en de camera zit. Dit wordt op dezelfde manier gedaan als de Parenting functie van het Player Script door een soort straal uit te zenden en te controleren of deze iets raakt. Indien dit gebeurt wordt de camerapositie veranderd zodat deze voor het object komt.

```
//Collision//
RaycastHit hit;

if (Physics.Raycast(TransCamX.position, -TransCamX.forward, out hit, CamDist + 0.3f))
{
    TransCamera.position = hit.point + (TransCamX.forward * 0.3f);
}
else
{
    TransCamera.localPosition = new Vector3(0, 0, -CamDist);
}
```

Afbeelding 30 - Camera deel 5

2.1.1.6 De UI

De UI, ofwel User Interface, is bijna even belangrijk als de camera zelf. Hier kan je alle soorten dingen weergeven die de gebruiker zou moeten weten, zoals hier bijvoorbeeld het aantal levens, exp en het aantal rupees dat de speler al heeft. Natuurlijk kunnen deze waardes niet zomaar aangepast worden en zijn hier ook verschillende scripts verantwoordelijk voor. Gelukkig zijn deze scripts redelijk simpel.



Afbeelding 31 - De UI van Zoey

Als eerste is er het HealthBar Script. Hier wordt de waarde van de overgebleven levens bij elke frame opgehaald uit de GameInfo klasse om vervolgens verdeeld te worden over de balk. De SmoothStep functie zorgt voor een mooie animatie tijdens het opvullen of leeglopen van deze balk. Dit is vooral zichtbaar in het begin van de game. Het ExpBar Script werkt exact hetzelfde als het Healthbar Script, alleen met de Exp in plaats van de levens.

```
public class HealthBarScript : MonoBehaviour {

    public Image HealthProgress;

    void LateUpdate () {

        HealthProgress.fillAmount = Mathf.SmoothStep(HealthProgress.fillAmount, GameInfo.health / (10 * GameInfo.lvl), 0.3f);
    }
}
```

Afbeelding 32 - HealthBar Script

```
public class EXPBarScript : MonoBehaviour {

    public Image EXPProgress;

    private void LateUpdate()
    {
        EXPProgress.fillAmount = Mathf.SmoothStep(EXPProgress.fillAmount, GameInfo.exp / (100 * GameInfo.lvl), 0.3f);
    }
}
```

Afbeelding 33 - ExpBar Script

```
using UnityEngine.UI;
```

Afbeelding 34 - UI gedeelte van Unity

Beide scripts maken gebruiken van de afbeelding, of Image, van de balken. Hiervoor maak je gebruik van het UI-gedeelte van Unity. Door de lijn van afbeelding 34 bovenaan toe te voegen kan je deze gebruiken in het script.

Het Rupee Counter Script werkt ook hetzelfde als de 2 vorige scripts, enkel werkt deze met de tekst in plaats van de afbeelding.

```
public class RupeeCountScript : MonoBehaviour {

    public Text UItext;

    void LateUpdate () {
        UItext.text = "Rupee\n" + GameInfo.rupee.ToString();
    }
}
```

Afbeelding 35 - Rupee Counter Script

2.1.2 Dogoo

Afbeelding 36 - Dogoo

Zonder vijanden zou er ook niet veel te doen zijn in een game, daarvoor zijn er in deze game de Dogoo's. Deze zijn ook gemaakt, geanimeerd en gemodelleerd door Diaz. Net als Zoey bestaat een Dogoo uit verschillende onderdelen:

- *Het model*
- *De animaties*
- *De character controller*
- *Het Enemy AI Script*

Aangezien we het model, de animaties en de character controller al hebben uitgelegd, zullen we het hier rechtstreeks over het script hebben.

2.1.2.1 Enemy AI Script

Aangezien je de vijanden niet zelf zou besturen, wordt een Dogoo bestuurd door Unity door middel van een AI. Maar voor we daarover gaan vertellen, heeft het Enemy AI Script, net zoals bijna alle vorige scripts, enkele variabelen.

```
CharacterController EnemyCC;
Animator CharAM;
Transform PlayerTrans;

public GameObject DeathEffect;

public int LVL = 1, DefEXP = 5;
public float Health = 5, WanderRotSpeed = 2, MovementSpeed = 5, JumpForce = 5;

bool SeenPlayer = false;
private float YSpeed, WaitForActionTimer;
```

Afbeelding 37 - Variabelen

- *EnemyCC:* De character controller van de Dogoo.
- *CharAM:* De animator van de Dogoo.
- *PlayerTrans:* De locatie van de speler.
- *DeathEffect:* Het object dat wordt afgespeeld als de Dogoo "sterft".
- *LVL:* Het level van de Dogoo.
- *DefEXP:* De standaard Exp dat de speler krijgt indien de Dogoo "sterft".
- *Health:* De levens van de Dogoo.
- *WanderRotSpeed:* De snelheid waarmee de Dogoo kan ronddraaien.
- *MovementSPeed:* De snelheid waarmee de Dogoo kan bewegen.
- *JumpForce:* De kracht waarmee de Dogoo kan springen.
- *SeenPlayer:* Om na te kijken of de Dogoo de speler heeft gezien of niet.
- *YSpeed:* De snelheid van de Dogoo in de Y-richting (boven of onder).
- *WaitForActionTimer:* Hoelang de Dogoo moet wachten vooraleer hij naar de speler gaat.

In de Awake functie worden ook deze keer enkele variabelen gedefineerd, zoals bijvoorbeeld de character controller en de animator. Ook wordt naar de speler gezocht om de locatie hiervan te zoeken. Vervolgens wordt de Dogoo op de grond gezet en wordt de Parenting functie opgeroepen.

```
private void Awake()
{
    EnemyCC = GetComponent<CharacterController>();
    CharAM = GetComponent<Animator>();
    PlayerTrans = GameObject.FindGameObjectWithTag("Player").transform;

    EnemyCC.Move(new Vector3(0, Physics.gravity.y, 0));
    Parenting();
    CharAM.SetBool("IsGrounded", true);
}
```

Afbeelding 38 - Awake

In de Update wordt eerst gecontroleerd of men zich in een andere scène bevind om vervolgens de Movement, Parenting, LookForPlayer en VarUpdate functies uit te voeren. Indien dit niet gebeurt wordt er niets gedaan.

```
private void Update()
{
    if (!GameInfo.isInCutScene)
    {
        Movement();
        Parenting();
        LookForPlayer();
        VarUpdate();
    }
    else
    {

    }
}
```

Afbeelding 39 - Update

In de VarUpdate functie worden net als bij de speler enkele variabelen aangepast. Ten eerste wordt de WaitForAction per seconde lager gemaakt. Hierbij wordt rekening gehouden dat deze waarde zich tussen 0 en 9999 is. Hierna wordt gecontroleerd of de Doggo zich lager dan het -15 niveau bevind. Net als de speler zal de Dogoo dan "sterven". Hierbij wordt het standaard Exp opgeteld bij het Exp dat de speler al heeft en worden het aantal levens op 0 gezet. Vervolgens wordt ook gecontroleerd of de levens 0 zijn. Indien dit gebeurt wordt het "doodgaan"-effect uitgevoerd op de plaats van de Dogoo en worden hierna zowel de Doggo als het effect uit de game verwijderd.

```
void VarUpdate()
{
    WaitForActionTimer = Mathf.Clamp(WaitForActionTimer - Time.deltaTime, 0, 9999f);

    if (transform.position.y <= -15f)
    {
        Health = 0f;
        GameInfo.exp += DefEXP / GameInfo.lvl * LVL * 2.5f;
        PlayerPrefs.SetFloat("P_EXP", GameInfo.exp);
    }

    if (Health == 0)
    {
        GameObject EffectGO = Instantiate(DeathEffect, transform.position, transform.rotation);
        Destroy(EffectGO, 5f);
        Destroy(gameObject);
    }
}
```

Afbeelding 40 - VarUpdate

De LookForPlayer functie controleert eerst of de speler al gezien is. Indien dit waar is word deze functie verder niet gebruikt. Als dit niet zo is wordt er gecontroleert of de speler binnen de 5 meter afstand is van de Dogoo. Indien dit zo is wordt SeenPlayer op waar gezet.

```
void LookForPlayer()
{
    if (SeenPlayer)
        return;

    SeenPlayer = Vector3.Distance(transform.position, PlayerTrans.position) <= 5f;
}
```

Afbeelding 41 - LookForPlayer

De Parenting functie kijkt net als bij het Player script na of de Dogoo op de grond staat of niet door middel van een straal uit te zenden.

```
void Parenting()
{
    RaycastHit hit;

    if (Physics.Raycast(transform.position + transform.up, -transform.up, out hit, 1.3f))
    {
        transform.parent = hit.transform;
    }
    else
    {
        transform.parent = null;
    }
}
```

Afbeelding 42 - Parenting

De TakeDamage functie zorgt ervoor dat een aantal DMG wordt afgetelt van de levens van de Dogoo. Indien het aantal levens 0 is wordt er de standaard Exp waarde toegevoegd aan dat van de speler. Deze functie wordt opgeroepen door een ander script.

```
public void TakeDamage(float DMG)
{
    Health = Mathf.Clamp(Health -= DMG / LVL, 0, 9999999999f);

    if (Health == 0)
    {
        GameInfo.exp += DefEXP / GameInfo.lvl * LVL;
        PlayerPrefs.SetFloat("P_EXP", GameInfo.exp);
    }
}
```

Afbeelding 43 - TakeDamage

Net als bij de speler is bij de Dogoo de Movement functie ook zeer uitgebreid. Ook zoals bij de speler zijn er eerst enkele variabelen die enkel in deze functie beschikbaar zijn.

- isGroundedNow: Of de Dogoo op de grond staat.
- ZSpeed: De snelheid in de Z-richting.

```
bool isGroundedNow = EnemyCC.isGrounded;
float ZSpeed = 0;
```

Afbeelding 44 - Variabelen

Hierna wordt als eerste gecontroleert of de Dogoo op de grond staat. Indien dit het geval is en de Dogoo heeft de speler ook gezien, zal de Dogoo springen en wordt isGroundedNow op niet waar gezet. Anders wordt de verticale snelheid gelijk aan de zwaartekracht.

```
if (isGroundedNow)
{
    RaycastHit hit;

    if (Physics.Raycast(transform.position + transform.up + transform.forward, -transform.up, out hit, 1.3f))
    {
        if ((transform.position.y - hit.point.y) < 0 && SeenPlayer)
        {
            isGroundedNow = false;
            YSpeed = JumpForce;
        }
        else
        {
            YSpeed = Physics.gravity.y;
        }
    }
    else if (SeenPlayer)
    {
        isGroundedNow = false;
        YSpeed = JumpForce;
    }
}
else
{
    YSpeed += Physics.gravity.y / 2 * Time.deltaTime;
}
```

Afbeelding 45 - Bewegen deel 1

```
if (SeenPlayer)
{
    ZSpeed = MovementSpeed;

    if (Vector3.Distance(transform.position, new Vector3(PlayerTrans.position.x, transform.position.y, PlayerTrans.position.z)) >= 2f || !isGroundedNow && WaitForActionTimer == 0)
    {
        CharAM.SetFloat("MoveSpeed", Mathf.SmoothStep(CharAM.GetFloat("MoveSpeed"), 1f, 0.25f));
        EnemyCC.Move(transform.rotation * new Vector3(0, YSpeed, ZSpeed) * Time.deltaTime);
    }
    else
    {
        CharAM.SetFloat("MoveSpeed", Mathf.SmoothStep(CharAM.GetFloat("MoveSpeed"), 0f, 0.25f));

        if (WaitForActionTimer == 0 && Mathf.RoundToInt(CharAM.GetFloat("MoveSpeed")) == 0)
        {
            WaitForActionTimer = 1.3f;
            CharAM.Play("Attack1");
        }
    }

    if (WaitForActionTimer <= 0.25f)
    {
        Quaternion targetRotation = Quaternion.LookRotation((new Vector3(PlayerTrans.position.x, transform.position.y, PlayerTrans.position.z) - transform.position).normalized);
        transform.rotation = Quaternion.Slerp(transform.rotation, targetRotation, 0.1f);
    }
}
else
{
    ZSpeed = MovementSpeed / 2;

    EnemyCC.Move(transform.rotation * new Vector3(0, YSpeed, ZSpeed) * Time.deltaTime);

    transform.Rotate(0, WanderRotSpeed, 0);
}
```

Afbeelding 46 - Bewegen deel 2

Het volgende gedeelte van de Movement functie is een beetje complexer dan de vorige functies. Ik zal het daarom ook proberen zo makkelijk mogelijk als ik kan uit te leggen. Dit gedeelte wordt ten eerste uitgevoerd als de Dogoo de speler gezien heeft.

Vervolgens wordt eerst gecontroleerd of de speler verder dan 2m is of dat de Dogoo niet op de grond is en de WaitForActionTimer 0 is. Dit klinkt een beetje ingewikkeld, maar zorgt er eigenlijk gewoon voor dat als dit waar is dat de

Dogoo een bepaalde snelheid krijgt zodat deze naar de speler kan bewegen (MoveSpeed wordt op 1 gezet met de SmoothStep functie en de Move functie wordt uitgevoerd).

Indien de Dogoo al dicht genoeg is, valt de Dogoo stil (MoveSpeed wordt op 0 gezet in de SmoothStep functie). Hierna wordt gecontroleerd of de WaitForActionTimer op 0 staat en de Dogoo stilstaat. Als dit gebeurt wordt de Timer terug op 1.3 gezet en speelt de "Attack1" animatie.

Als laatste wordt gecontroleerd of de Timer minder dan 0.25 is. Indien dit het geval is zal de Dogoo zich richting de speler draaien. Dit zie je aan het moeilijker woord Quaternation.

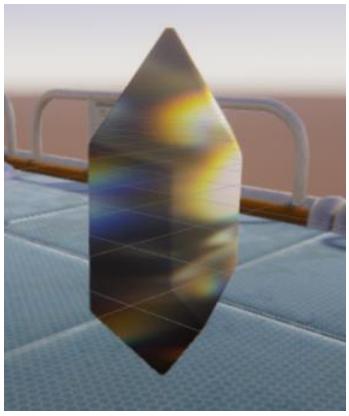
Indien de Dogoo de speler nog niet ziet, gaat de Dogoo gewoon in rondjes rondspringen.

Als allerlaatste is er nog een klein stukje code dat controleert of de Dogoo op de grond staat en indien nodig de YSpeed op 0 zet hiervoor. Daarna wordt er nog voor gezorgd dat de animator weet of de Dogoo op de grond staat of niet.

```
if (isGroundedNow)
{
    YSpeed = 0;
}

CharAM.SetBool("IsGrounded", isGroundedNow);
```

Afbeelding 47 - Bewegen deel 3

2.1.3 Rupee



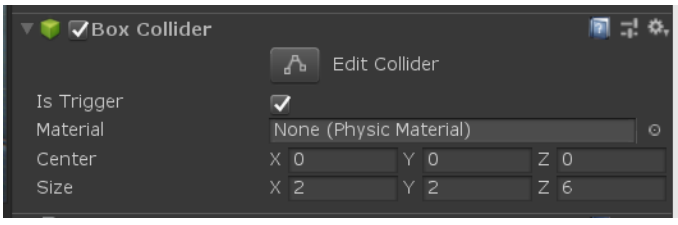
Afbeelding 48 - Rupee

In een game is het natuurlijk ook leuk om dingen te verzamelen terwijl je speelt. In deze game zijn dat rupees. Net zoals Zoey en een Dogoo bestaat deze uit verschillende onderdelen:

- *Het model:* duh, dat weten we ondertussen al dat alles een model nodig heeft.
- *De animaties:* Dit is ook iets dat we al weten.
- *De box collider*
- *Het Rupee Script*

2.1.3.1 De box collider

Een box collider zorgt ervoor dat een object iets vast word, iets waartegen je kan botsen. Deze collider kan in verschillende andere vormen ook voorkomen zoals bijvoorbeeld een bol.



Afbeelding 49 - Box Collider

- *Is Trigger:* Indien dit aangeduid wordt, is de collider geen echte collider maar wordt er enkel gecontroleert of er iets tegen botst voor bijvoorbeeld een script.
- *Material:* Een collider kan eventueel ook van een bepaald materiaal zijn. Zo kan je een zeer botsend materiaal maken voor bijvoorbeeld een trampoline.
- *Center:* Het middelpunt van de collider.
- *Size:* De grootte van de collider, uitgedrukt in X,Y en Z waardes.

2.1.3.2 Het Rupee Script

Het Rupee Script is gelukkig niet zo uitgebreid als de scripts van Zoey en een Dogoo.

Ten eerste wordt het object voor de animatie als de Rupee wordt "opgepakt" gedefineerd.

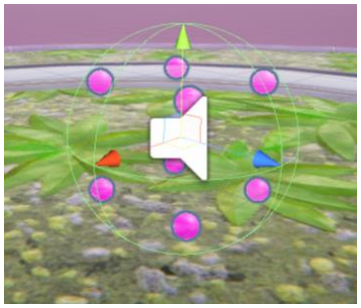
```
public class RupeeScript : MonoBehaviour {

    public GameObject SpawnOBJ;

    private void OnTriggerEnter(Collider other)
    {
        if (other.tag == "Player")
        {
            GameInfo.rupee++;
            PlayerPrefs.SetInt("P_Rupee", GameInfo.rupee);
            GameObject SpawnedOBJ = Instantiate(SpawnOBJ, transform.position, transform.rotation);
            Destroy(SpawnedOBJ, 1f);
            Destroy(gameObject);
        }
    }
}
```

Afbeelding 50 - Rupee Script

De OnTriggerEnter functie is een functie van de box collider. Als je goed opgelet hebt bij het vorige onderdeel heb je gezien dat Is Trigger is aangeduid. Dit zorgt ervoor dat je deze functie kan gebruiken om te controleren of er iets of iemand tegen de Rupie botst.

Als er iets tegen botst wordt er eerst gecontroleerd of dit iets de speler is door middel van de tag die je een alle objecten kan geven. Als dit waar is wordt het aantal Rupees van de speler verhoogd en wordt de "oppak" animatie afgespeeld om vervolgens de animatie en de Rupee uit het spel te verwijderen.

2.1.4 Dim Crystal



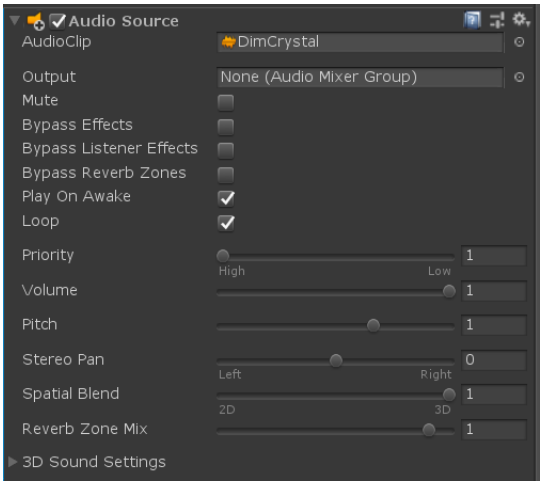
Afbeelding 51 - Dim Crystal

Een eindeloze game kan ook niet zo heel fijn zijn, daarom heeft deze game een Dim Crystal op het einde van het level. Deze bestaat ook uit enkele onderdelen.

- Het model
- *De animaties*
- *De sphere collider:* Hetzelfde als een box collider, maar dan in bolvorm.
- *Een geluidsbron*
- *Het Dim Crystal Script*

2.1.4.1 Geluidsbron

Om een Dim Crystal, of andere objecten, iets interessanter te maken kan je er een geluidsbron aan toevoegen. Het geluid dat deze moet afspelen is een AudioClip. Je kan voor bepaalde soorten geluiden ook een Audio Mixer Group aanmaken, deze kan je toevoegen aan het Output gedeelte. Verder zijn er nog verschillende effecten die je kan toevoegen aan het geluid. Play On Awake zorgt ervoor dat het geluid eventueel wordt afgespeeld van het moment dat het object bestaat. Loop zorgt ervoor dat het geluid blijft herhaald worden.



Afbeelding 52 - Geluidsbron

2.1.4.2 Dim Crystal Script

```
using UnityEngine.SceneManagement;

public class DimCrystalScript : MonoBehaviour {

    private void OnTriggerEnter(Collider other)
    {
        if (other.tag == "Player")
        {
            if ((GameInfo.levelid + 1) < SceneManager.sceneCountInBuildSettings)
            {
                GameInfo.levelid++;
                PlayerPrefs.SetInt("lvl_id", GameInfo.levelid);
                SceneManager.LoadScene(0);
            }
            else
            {
                PlayerPrefs.DeleteAll();
                GameInfo.exp = PlayerPrefs.GetFloat("P_EXP", 0f);
                GameInfo.lvl = PlayerPrefs.GetInt("P_LVL", 1);
                GameInfo.rupee = PlayerPrefs.GetInt("P_Rupee", 0);
                GameInfo.levelid = PlayerPrefs.GetInt("lvl_id", 1);
                SceneManager.LoadScene(0);
            }
        }
    }
}
```

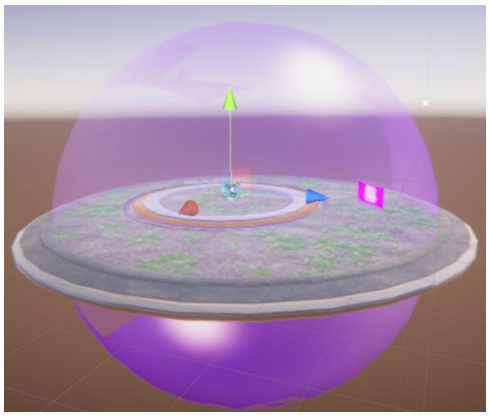
Afbeelding 53 - Dim Crystal Script

Net zoals alle andere objecten heeft het Dim Crystal ook een script nodig om dingen te kunnen doen.

Ten eerste is voor dit script het SceneManagement gedeelte nodig. Unity werkt met scènes om de verschillende levels of menus van mekaar te onderscheiden.

De enige functie in het Dim Crystal is de OnTriggerEnter functie. Net als bij een Rupee controleert dit of er iets met het Dim Crystal botst en kijkt het na of dit de speler is. Daarna wordt er gecontroleerd of er nog levels na deze scène voorkomen om daarbij het levelid hiervoor aan te passen indien dit waar is. Daarna wordt een andere scène geladen, in dit geval is dit opnieuw hetzelfde level. Als er geen andere levels voorkomen worden ook alle waardes van de GameInfo verwijdert en voor enkele teruggezet naar hun oorspronkelijke waardes. Daarna wordt ook hier het level opnieuw geladen.

2.1.5 Safezone Shield



Afbeelding 54 - Safezone Shield

Rond het Dim Crystal bevind zich ook een Safezone Shield. Net als alle andere objecten bestaat dit ook uit enkele onderdelen.

- Het model
- De animaties
- De sphere collider
- Het Damage Trigger Script

2.1.5.1 Het Damage Trigger Script

Het Damage Trigger Script zorgt ervoor dat het Safezone Shield kan gebruikt worden als schild tegen Vijanden zoals Dogoos of juist omgekeerd tegen spelers zoals Zoey.

```
public class DamageTriggerScript : MonoBehaviour {

    public EnemyAIScript EnemyAI;
    public bool isFriendly;
    public float Damage;

    private void OnTriggerStay(Collider other)
    {
        if (other.tag == "Player" && !isFriendly)
        {
            if (GameInfo.dmgCoolDown == 0)
            {
                GameInfo.dmgCoolDown = 1.35f;
                GameInfo.health = Mathf.Clamp(GameInfo.health -= Damage * EnemyAI.LVL / GameInfo.lvl, 0, 99999999999f);
            }
        }
        else if (other.tag == "Enemy" && isFriendly)
        {
            EnemyAIScript HitedEnemy = other.GetComponent<EnemyAIScript>();

            HitedEnemy.TakeDamage(Damage * GameInfo.lvl);
        }
    }
}
```

Afbeelding 55 - Het Damage Trigger Script

Hierbij worden, net zoals bij de meeste vorige scripts, enkele variabelen gedefinieerd.

- *EnemyAI:* De vijandelijke AI.
- *isFriendly:* Of het schild "vriendelijk" is of niet.
- *Damage:* Hoeveel schade het schild aanricht

Hierbij wordt zoals bij het Dim Crystal en een Rupee gebruik gemaakt van een OnTriggerEnter functie. Hierin wordt eerst gecontroleerd of de speler tegen het schild botst. Als het schild daarbij ook op niet vriendelijk staat wordt de Damage van de levens van de speler afgenomen. Indien het niet de speler maar een vijand is en het schild op vriendelijk staat, wordt hetzelfde gedaan maar dan tegen de vijand.

2.1.6 GameInfo

GameInfo is geen script, maar een klasse. Dit kan je zien aan het ontbreken van "MonoBehaviour" aan het begin van het bestand. GameInfo wordt gebruikt om enkele variabelen te definieren die over heel de game gebruikt worden door verschillende scripts.

```
public class GameInfo {

    public static float exp = PlayerPrefs.GetFloat("P_EXP", 0f);
    public static int lvl = PlayerPrefs.GetInt("P_LVL", 1);
    public static int rupee = PlayerPrefs.GetInt("P_Rupee", 0);
    public static int levelid = PlayerPrefs.GetInt("lvl_id", 1);

    public static float health = 10f * lvl;
    public static float dmgCoolDown = 0f;
    public static bool isInCutScene = false;
}
```

Afbeelding 56 - GameInfo

- *Exp:* Het aantal exp van de speler.
- *Lvl:* Het level van de speler.
- *Rupee:* Het aantal Rupees dat de speler heeft.
- *Levelid:* Het id van de level scène.
- *Health:* Het aantal levens van de speler.
- *dmgCoolDown:* De damage cooldown.
- *isInCutScene:* Of de speler in een andere scène is.

2.1.7 Loading Screen

De Loading Screen scène is een scène die wordt opgeroepen voor het laden van het level en als het level herstart wordt. Deze scène zorgt ervoor dat je een laadbalk krijgt terwijl dat het echte level geladen wordt. Hiervoor wordt gebruik gemaakt van het Load Scene Script.

Afbeelding 57 - Loading Screen

2.1.7.1 Load Scene Script

Omdat het Loading Scene Script met scènes laden werkt, heeft deze ook het SceneManagement gedeelte van Unity nodig. Ook is het UI gedeelte nodig voor de balk op te vullen tijdens het laten.

Hierna zijn er weeral de variabelen.

- asyncLoad: Hoe ver Unity is met het laden van de scène.
- Progress: De laadbalk afbeelding.

In de Start functie wordt een Coroutine gestart. Een Coroutine is iets te moeilijk om in detail uit te leggen, maar het script wacht hier tot de LoadScenAsync functie een waarde terugstuurt.

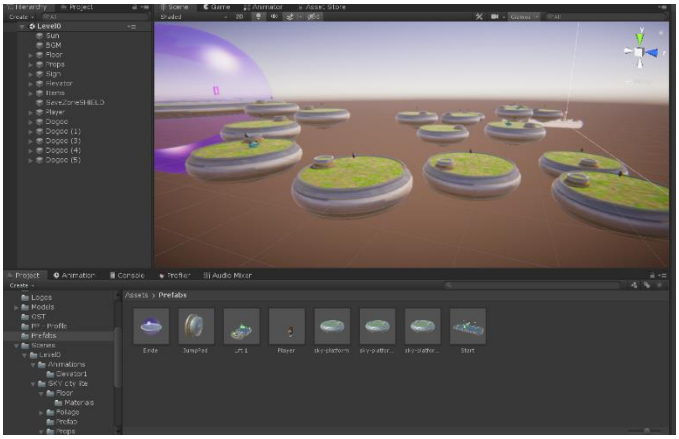
```
using UnityEngine.UI;
using UnityEngine.SceneManagement;

public class LoadSceneScript : MonoBehaviour {

    AsyncOperation asyncLoad;

    public Image Progress;

    void Start() {
        StartCoroutine(LoadSceneAsync());
    }

    void Update() {
        Progress.fillAmount = asyncLoad.progress;

    }

    IEnumerator LoadSceneAsync() {

        asyncLoad = SceneManager.LoadSceneAsync(GameInfo.levelid);

        while (!asyncLoad.isDone)
        {
            yield return null;
        }
    }
}
```

Afbeelding 58 - Load Scene Script

In de Update functie wordt elke frame de hoeveelheid dat de laadbalk opgevuld is aangepast aan hoever de volgende scène geladen is.

2.2 Prefabs maken

Prefabs zijn zeer handige onderdelen van Unity. Dit zijn eigenlijk objecten die je kan hergebruiken doorheen de game en in verschillende scripts. In deze game wordt daar bijvoorbeeld al van gebruik gemaakt voor de effecten van het doodgaan van de Dogoo en voor het oppakken van een Rupee. Dit kan natuurlijk uitgebreid worden voor elk ander object dat je wil hergebruiken, zoals bijvoorbeeld de platformen van het level.



Afbeelding 59 - Prefabs

Geen paniek, ik heb deze prefabs al zelf gemaakt om te gebruiken tijdens de les. De prefabs die ik gemaakt heb zijn het startgedeelte, het liftgedeelte, het eindplatform, een skyplatform zonder iets speciaals, een skyplatform met een dogoo en tot slotte een skyplatform met een jumppad.

Indien je zelf nog prefabs wil maken, kan dit heel gemakkelijk gebeuren. Je sleept gewoon een object van in het levelmenu in het bestandsmenu. Daarna kan je natuurlijk ook nog alles aan de prefabs aanpassen zodat deze zijn zoals jij ze wil gebruiken.

In afbeelding 59 kan je linksboven het levelmenu zien met alle objecten die aanwezig zijn in het level en in het midden onderaan de prefabs die ik al klaargemaakt heb in het bestandsmenu.

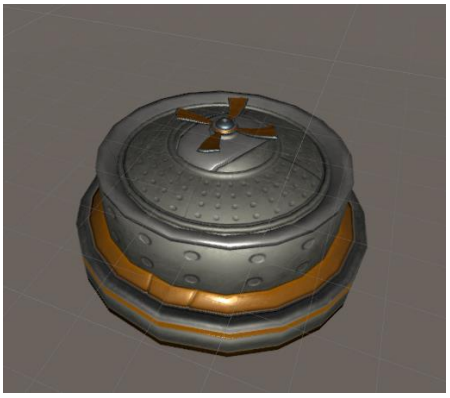
3 JUMPPAD

Nu dat je alles terug goed kent van de game, kan het echte werk beginnen. We zullen eerst beginnen met een iets makkelijker object om te maken in Unity, namelijk een jumppad. Een jumppad komt in zeer veel games, zoals bijvoorbeeld Fortnite, voor en is eigenlijk gewoon een trampoline die je kan gebruiken om bijvoorbeeld op nieuwe plaatsen te raken.

3.1 Prefabs

3.1.1 Jumppad

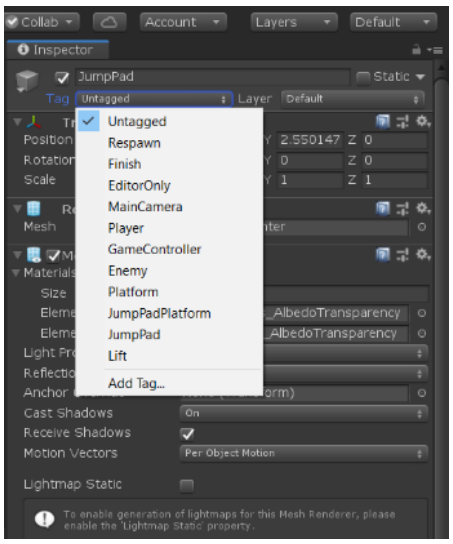
We zouden het in de voorbereiding niet over prefabs hebben gehad als we ze niet zouden gebruiken. Deze heb ik gelukkig al voor jou gemaakt. Als je goed kijkt kan je misschien de modellen herkennen die ik heb gebruikt, namelijk de planter die op elk platform staat en een mini omgekeerde versie van een platform.



Afbeelding 60 - Jumppad

Ook is het handig om deze prefab van een tag te voorzien. Net als Zoey een Player tag heeft en een Dogoo een Enemy tag, zal de Jumppad een JumpPad tag krijgen om het makkelijk te herkennen tijdens het programmeren.

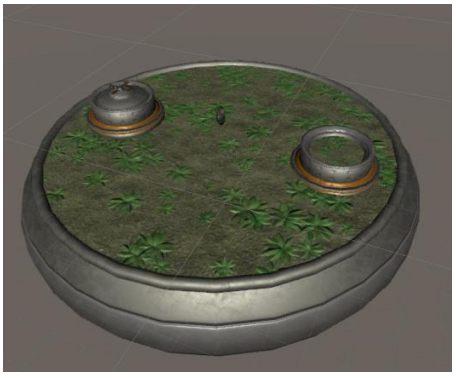
Indien de JumpPad tag, of een andere tag die je zou willen gebruiken, er niet tussen staat dan kan je makkelijk een nieuwe toevoegen. Je klikt gewoon op "Add Tag.." en op het plusje in het venster dat zich dan opent. Daarna kan je een naam invullen voor je nieuwe tag en die kan je dan vervolgens gebruiken in je game.



Afbeelding 61 - Tag

3.1.2 Jumppad platform

Een jumppad op zichzelf gaat redelijk moeilijk zijn om op te mikken om op te kunnen springen zonder eraf te vallen. Daarom heb ik ook nog een tweede prefab gemaakt met de jumppad op een platform. Hierdoor kan je eerst makkelijk op het platform landen om vervolgens op de jumppad te springen.



Afbeelding 62 - Jumppad platform

Prefabs kan je zo ook makkelijk combineren. Deze prefab is bijvoorbeeld een kopie van de prefab van een gewoon platform waar dan ook de prefab van de jumppad is in gezet. Indien de jumppad prefab dan aangepast wordt, zal deze ook hier in deze prefab aangepast worden zonder dat je daar zelf iets aan moet doen.

Deze prefab is ook van een tag voorzien, namelijk de JumpPadPlatform tag. Deze tag zal later van pas komen bij het volgende deel van de les.

3.2 Script

```
if (isGroundedNow)
{
    if (Input.GetButtonDown("Jump"))
    {
        isGroundedNow = false;
        YSpeed = JumpForce;
    }
    else
    {
        YSpeed = Physics.gravity.y;
    }
}
else
{
    YSpeed += Physics.gravity.y * Time.deltaTime;
}
```

Afbeelding 63 - het "spring" gedeelte

Een jumppad is heel gemakkelijk te programmeren. Het is zelfs zo makkelijk dat een jumppad niet eens een eigen script nodig heeft. In plaats daarvan gaan we in het Player Script werken.

Hiervoor gaan we werken in het "spring" gedeelte van het Player Script. Om na te gaan of de speler op een jumppad staat, gaan we een ander stukje code hergebruiken. Zoals je misschien nog weet wordt er in de Parenting functie nagekeken of de speler op de grond staat door middel van een Raycast naar beneden. Als we deze nu gebruiken om na te kijken wat er onder de voeten van de speler is en of deze "grond" de "JumpPad" tag heeft, dan hebben we het jumppad gevonden.

```
if (Input.GetButtonDown("Jump"))
{
    isGroundedNow = false;
    YSpeed = JumpForce;
}
else if (Physics.Raycast(transform.position + transform.up, -transform.up, out hit, 1.1f) && hit.transform.tag == "JumpPad")
{
    isGroundedNow = false;
    YSpeed = JumpForce * 5;
}
else
{
    YSpeed = Physics.gravity.y;
}
```

Afbeelding 64 - Jumppad code

Om het springen na te maken hoeven we ook enkel de YSpeed aan te passen. Omdat een jumppad natuurlijk je hoger laat springen dan dat je normaal zou springen, vermenigvuldigen we de springkracht met bijvoorbeeld 5. Natuurlijk wordt isGroundedNow ook op niet waar gezet aangezien de speler omhoog springt.

Bij het programmeren gebruik je && om ervoor te zorgen dat het eerste deel *en* het tweede deel allebei waar moeten zijn. == wordt gebruikt om na te kijken of iets gelijk is aan iets anders. Voor andere vergelijking kan je bijvoorbeeld > en > of >= en <= voor groter/kleiner en groter of gelijk aan/kleiner of gelijk aan. != wordt gebruikt voor na te kijken of iets *niet* gelijk is aan iets anders.

Het stukje code wordt ook tussen de 2 delen gezet omdat deze eerst nog moet nakijken of er een jumppad is voor dat het script naar het volgende deel kan gaan. Anders zou dit rechtstreeks naar het else gedeelte gaan en de zwaartekracht als YSpeed gebruiken.

3.3 Uittesten

Afbeelding 65 - Tools

Nadat je dit allemaal gedaan hebt is het natuurlijk tijd om dit uit te testen. Het jumppad platform kan je makkelijk toevoegen aan de game door deze gewoon in het levelmenu of in de scène zelf te slepen. Daarna kan je deze makkelijk verplaatsen naar waar je wil met de pijltjes als je het platform selecteert. Indien je geen pijltjes hebt, kan je deze linksboven selecteren zoals in foto 65.

4 AUTOMATISCH LEVEL GENEREREN

Een level zelf volledig maken kan handig zijn om alles te zetten zoals jij wil, waar jij het wilt en hoeveel je van elk ding wil. Maar dit kan erg lang duren, zeker als je grote of veel levels wil maken. Dus waarom zouden we gewoon Unity dit kunnen laten doen voor ons. Aangezien dit niet al te makkelijk is om in een keer te doen, gaan we het in verschillende stappen maken.

4.1 Platformen genereren

4.1.1 Script

Om te beginnen gaan we een basisscriptje maken dat gewoon een bepaald aantal platformen in een rij genereerd.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class LevelManager : MonoBehaviour
{
    // Start is called before the first frame update
    void Start()
    {

    }

    // Update is called once per frame
    void Update()
    {

    }
}
```

Afbeelding 66 - Een "leeg" LevelManager script

Als eerste moeten we natuurlijk het script aanmaken. Zorg hiervoor dat je in de "scripts" map staat, hierdoor blijft alles netjes bijeen. Rechtsklik in het bestandsmenu, ga naar "Create" en klik daarna op C# script. Hierna kan je een naam kiezen voor je nieuwe script. Hiervoor gebruik je best duidelijke namen die zeggen wat het script doet of waarover het gaat, daarom heb ik dit script LevelManager genoemd.

Als je het script geopend hebt, ziet dit er normaal zoals in afbeelding 66 uit. Dit is een "leeg" script omdat geen enkele functie iets doet en er ook geen variabelen aanwezig zijn. De Update functie mag je uit het script verwijderen, deze gaan we toch niet gebruiken.

```
[SerializeField] private GameObject skyplatform;
```

Afbeelding 67 - SerializeField

Programmeertip: Als je je variabelen enkel in het script wil gebruiken waar ze gedefinieerd zijn (private), maar ze toch nog wilt aanpassen of definiëren in Unity zelf (normaal gezien enkel bij "public" variabelen), kan je gebruik maken van [SerializeField]. Door gebruik te maken van "private" variabelen kunnen er ook veel moeilijker misvattingen voorkomen als meerdere scripts dezelfde namen voor variabelen gebruiken.

Enkele variabelen die we nodig hebben voor dit deel:

- *Skyplatform:* Dit is van het GameObject type, zoals de animaties van een Dogoo en Rupee. Hierdoor kan je je prefab gebruiken.
- *startPositie:* Dit is van het Vector3 type, een punt met een X,Y en Z waarde.
- *Aantal:* Het aantal platformen dat je wil genereren.

```
[SerializeField] private GameObject skyplatform;

[SerializeField] private int aantal;

Vector3 startPositie;
```

Afbeelding 68 - Variabelen

Het aantal platformen en het skyplatform gaan we in Unity zelf definiëren door een getal bij Aantal in te vullen en de skyplatform prefab in het ander vakje te slepen. Hierdoor kunnen we deze variabelen later nog makkelijk aanpassen indien dit nodig zou zijn.

In de Start functie definiëren we de startPositie als het nulpunt van de game en roepen we de SpawnPlatform functie op. Deze gaan we direct maken zodat de rode lijn verdwijnt.

```
void Start()
{
    startPositie = new Vector3(0, 0, 0);

    SpawnPlatforms(aantal);
}
```

Afbeelding 69 - Start

Een functie die niets terugstuurt wordt een "void" functie genoemd. Zoals je zag in de start functie wordt er ook een waarde meegegeven aan de functie. Deze wordt dan beschikbaar binnen de functie en kan je zelf ook een andere naam geven als je wilt. Ik heb ze om het makkelijk te maken ook gewoon aantal genoemd.

```
void Start()
{
    startPositie = new Vector3(0, 0, 0);

    SpawnPlatforms(aantal);
}

private void SpawnPlatforms(int aantal)
{

}
```

Afbeelding 70 - Een "lege" SpawnPlatforms functie

Als eerste gaan we in deze functie ook weer enkele variabelen aanmaken. Voorlopig is dit nog maar 1 variabelen, namelijk positie. Deze gebruiken we om de posities van de platformen te bepalen en maken we eerst gelijk met de startPositie.

Vervolgens maken we een for-loop. Dit kan je in Visual Studio makkelijk doen door for te typen en dan 2 keer snel op de tabknop te duwen. Deze for loop gaat enkele keren herhaald, namelijk "length" keren. Deze "length" mag je vervangen door "aantal" aangezien dit het aantal keer is dat we deze code willen uitvoeren.

```
private void SpawnPlatforms(int aantal)
{
    Vector3 positie = startPositie;

    for (int i = 0; i < aantal; i++)
    {

    }
}
```

Afbeelding 71 - positie en for-loop

In deze for-loop gaan we eerst de Z-positie, of voorwaartse positie, aanpassen zodat de platformen niet allemaal op dezelfde plaats terecht komen. Dit wordt voorlopig door het getal 15 gedaan, maar wordt later aangepast naar een variabele. Als de positie aangepast is wordt het skyplatform project in de game gezet door middel van de Instantiate functie. Hierbij is "skyplatform" het object dat in de game gezet moet worden, "positie" de positie waar dit object gezet moet worden en "Quaternation.identy" de hoek waarin dit in de game word gezet. Quaternation.identy is de originele hoek van de prefab die Unity ophaalt en hergebruikt.

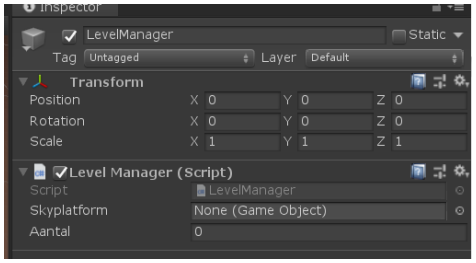
```
private void SpawnPlatforms(int aantal)
{
    Vector3 positie = startPositie;

    for (int i = 0; i < aantal; i++)
    {
        positie.z += 15;

        Instantiate(skyplatform, positie, Quaternion.identity);
    }
}
```

Afbeelding 72 - SpawnPlatforms

Hiermee is het script voorlopig klaar.

4.1.2 Script in de game zetten

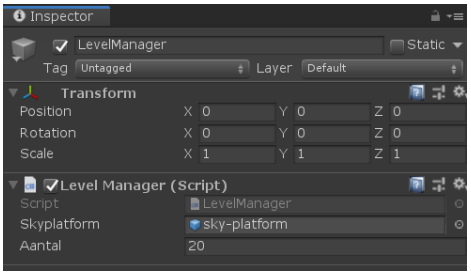
Om het script in de game zelf te kunnen zetten, moeten we dit aan een object kunnen hangen. Gelukkig heeft Unity hiervoor het perfecte object aangezien we dit niet zomaar aan eentje met een model kunnen hangen, namelijk een Empty (leeg) object.



Afbeelding 73 - Het LevelManager script is toegevoegd aan het object

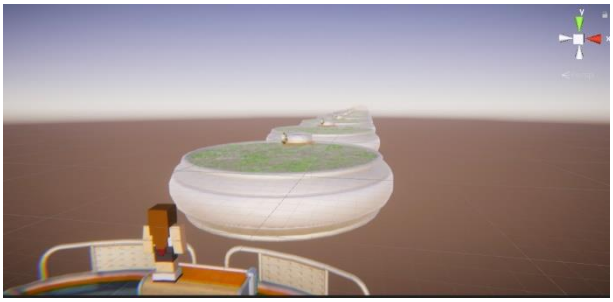
Als je rechtsklikt in het levelmenu kun je op "Create Empty" klikken. Dit object kan je dan een passende naam, zoals bijvoorbeeld LevelManager, geven. Als je daarna dit object selecteert, kan je rechts op "Add Component" klikken, "LevelManager" in de zoekbalk invullen en dan dubbelklikken op LevelManager om dit toe te voegen.

Hierna kan je de skyplatfrom prefab in het juiste vak slepen en het aantal skyplatforms dat je wil ingeven.



Afbeelding 74 - Skyplatform en aantal ingevuld

Als alles goed gedaan is en je nu op de "play" knop duwt zou er een hele rij platformen gegenereerd moeten zijn in de Z-richting.



Afbeelding 75 - Hoera! Een rij platformen

4.2 Meer soorten platformen

4.2.1 Script

Een lange rij platformen kan wel leuk zijn, maar er is geen variatie met welke platformen er zijn. Zo zijn er bijvoorbeeld geen platformen met een Dogoo of geen platformen met een jumppad. Daar gaan we nu verandering in brengen.

Zoals je waarschijnlijk al geraden had, gaan we eerst bovenaan enkele variabelen invoegen.

- *skyplatformMetDogoo:* Een GameObject.
- *skyplatformMetJumppad:* Een GameObject.
- *vorigPlatform:* Een GameObject om na te kijken wat het vorige platform is. Dit wordt gebruikt omdat je met een jumppad natuurlijk veel hoger springt. Ook wordt dit gebruikt als we later een lift gaan toevoegen.
- *kansDogoo:* Een getal voor de kans te bepalen dat het een platform met een Dogoo is. Deze wordt uitgedrukt in procent.
- *kansJumppad:* Hetzelfde als kansDogoo, maar dan met een Jumppad.

```
[SerializeField] private GameObject skyplatform;
[SerializeField] private GameObject skyplatformMetDogoo;
[SerializeField] private GameObject skyplatformMetJumppad;

[SerializeField] private int aantal;
[SerializeField] private int kansDogoo;
[SerializeField] private int kansJumppad;

private Vector3 startPositie;
private GameObject vorigPlatform;
```

Afbeelding 76 - Variabelen deel 2

Het vorigPlatform wordt in start eerst null (niet bestaand) gemaakt omdat dan er nog geen vorigPlatform bestaat.

```
// Start is called before the first frame update
void Start()
{
    startPositie = new Vector3(0, 0, 0);

    vorigPlatform = null;

    SpawnPlatforms(aantal);
}
```

Afbeelding 77 - Start deel 2

Eerst gaan we een functie maken om met de kansen te berekenen of een object gaat gegenereerd worden of niet. Aangezien deze functie een waar of niet waar moet teruggeven wordt dit een bool functie. Deze heb ik toepasselijk SetBool genoemd. We maken hier een functie van zodat we dit kunnen hergebruiken voor al de kansen.

```
private bool SetBool(int kans)
{

}
```

Afbeelding 78 - SetBool

Om de kans te bereken gaan we deze functie een willekeurig getal laten maken tussen 0 en 100 procent. Als dit getal dan lager is dan de kans dat een object genegereerd wordt, zal de functie waar terugsturen.

```
using Random = UnityEngine.Random;
```

Afbeelding 79 - Random

Voor een willekeurig getal te maken, maken we gebruik van de Random.Range functie van Unity. Hiervoor moet je eerst nog boven het lijntje uit afbeelding 79 toevoegen omdat er meerdere soorten "Random" bestaan en daardoor het script zich zou kunnen vergissen.

Als dit gemaakt is, gaan we terug in de SpawnPlatform functie werken. Hierin gaan we eerst een "gameObject" variabele maken en direct als null definiëren. Dit GameObject gaan we later in de functie gebruiken om in de Instantiate functie te gebruiken.

```
private bool SetBool(int kans)
{
    int getal = Random.Range(0, 100);

    if (getal <= kans)
    {
        return true;
    }
    else
    {
        return false;
    }
}
```

Afbeelding 80 - SetBool versie 2

Daarna gaan we in de for-loop ook enkele variabelen maken.

- *isDogooPlatform:* Bool die gebrukt maakt van de SetBool functie om te bereken of dit een platform met een Dogoo kan zijn.
- *isJumppadPlatform:* Zelfde als bij isDogooPlatform, maar dan met een platform met een jumppad.

```
private void SpawnPlatforms(int aantal)
{
    Vector3 positie = startPositie;
    GameObject gameObject = null;
```

Afbeelding 81 - Variabelen in SpawnPlatform

```
for (int i = 0; i < aantal; i++)
{
    bool isDogooPlatform = SetBool(kansDogoo);
    bool isJumpPadPlatform = SetBool(kansJumppad);
```

Afbeelding 82 - Variabelen in de for-loop

Nu gaat het echte werk beginnen. Omdat bij het platform met een jumppad het volgende platform een verschillende positie heeft dan normaal, gaan we de positiebepaling eerst een beetje aanpasen.

In de for-loop gaan we nu eerst controleren of de tag van het vorigPlatfrom gelijk is aan die van de skyplatfromMetJumpad prefab. Deze tag heb ik al toegevoegd aan de prefab zelf en noemt "JumpPadPlatform". Indien dit waar is wordt ook de Y waarde van de positie aangepast zodat het volgende platform hoger komt te liggen.

```
if (vorigPlatform.CompareTag("JumpPadPlatform"))
{
    positie.y += 10;
    positie.z += 15;
}
else
{
    positie.z += 15;
}
```

Afbeelding 83 - Positie bepalen

Nadat de positie bepaald is kunnen we nu ook gaan bepalen welk object we moeten gaan genereren. De kans dat dit gebeurt of niet hebben we al in het begin van de for-loop gedefinieerd, dus moeten we nu enkel nog controleren of dit waar is of niet. Eerst controleren we of het een platform met een jumppad is. Als dit waar is wordt het gameObject een skyplatfromMetJumppad. Indien dit niet waar is wordt er vervolgens gecontroleerd of het object een skyplatform met een Dogoo kan zijn. Als dit ook niet waar zou zijn, dan wordt het gameObject een doodnormaal skyplatform.

```
if (isJumpPadPlatform) gameObject = skyplatformMetJumppad;
else if (isDogooPlatform) gameObject = skyplatformMetDogoo;
else gameObject = skyplatform;
```

Afbeelding 83 - Welk platform?

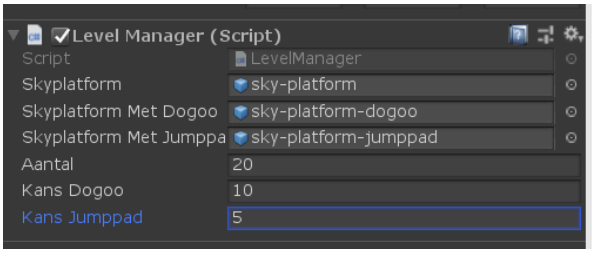
Hierna gaan we het object in de game genereren, ofwel instantiëren. Hiervoor gebruiken we de gameObject variabele die we met de vorige stap gedefinieerd hebben. Ook wordt de Instantiate functie toegewezen aan het platform object zodat we dit als laatste stap als vorigPlatform kunne definiëren op het einde van de for-loop. Hierdoor ziet de for-loop welk platform er er voor kwam in de volgende platform dat gegenereerd wordt.

```
GameObject platform = Instantiate(gameObject, positie, Quaternion.identity) as GameObject;
vorigPlatform = platform;
```

Afbeelding 84 - Platform genereren

4.2.2 Script testen

Als je alles goed gevolgd hebt, kan je nu in Unity enkele vakken erbij zien komen bij het script in het LevelManager object. Deze kan je invulllen door de juiste prefabs in de juiste vakken te slepen en vervolgens een kans tussen 0 en 100 in te geven bij de 2 vakken met kansen. Ik heb hierbij 10% kans op een eiland met een Dogoo en 5% kans op een eiland met een jumppad ingevuld.



Afbeelding 85 - Level Manager Script in Unity

Hierna kan je weer op de “play” knop duwen om je nieuwe script uit te testen.

Afbeelding 86 - Hoera! Een platform met een Dogoo

Afbeelding 87 - En een platform met een jumppad

4.3 Nog meer soorten platformen

4.3.1 Script

Nadat we al de gewone platformen, de platformen met een Dogoo en de platformen met een jumppad kunnen generen is het nu ook aan de beurt voor alle andere soorten platformen en ook de speler. Aangezien deze platformen ook niet allemaal even lang of breed zijn, gaan we ook nog een functie moeten toevoegen die de lengte of breedte van een platform kan berekenen.

Maar als eerste gaan we natuurlijk weer enkele variabelen aan ons script toevoegen.

- *Speler:*Een GameObject. Dit is de prefab van de speler.
- *Start:* Een GameObject. De prefab van het startplatform.
- *Einde:* Een GameObject. De prefab van het eindplatform.
- *Lift:* Een GameObject. De prefab van het liftplatform.
- *LengteStart:* De lengte van het startplatfrom.
- *BreedteEinde:* De breedte (en lengte omdat het een cirkel is) van het eindplatform.
- *kansLift:* De kans dat het platform een lift gaat zijn. Uitgedrukt in procent.

```
[SerializeField] private GameObject speler;
[SerializeField] private GameObject start;
[SerializeField] private GameObject skyplatform;
[SerializeField] private GameObject skyplatformMetDogoo;
[SerializeField] private GameObject skyplatformMetJumppad;
[SerializeField] private GameObject lift;
[SerializeField] private GameObject einde;

[SerializeField] private int aantal;
[SerializeField] private int kansDogoo;
[SerializeField] private int kansLift;
[SerializeField] private int kansJumppad;

private float breedteEinde;
private float lengteStart;

private Vector3 startPositie;
private GameObject vorigPlatform;
```

Afbeelding 88 - Variabelen

Zoals je ziet in afbeelding 88 begint het aantal variabelen toch groot te worden.

In de Start functie gaan we hiervoor enkele nieuwe functies oproepen. Ten eerste wordt de lengte van het startplatform en de breedte van het eindplatfrom bepaald met de GetLengte functie die we in de volgende stap gaan maken. Ten slotte worden ook nog de SpawnStart, SpawnSpeler en SpawnEinde functies opgeroepen. Deze functies gaan we ook nog maken. De volgorde van deze functies is zeer belangrijk aangezien we eerst een startplatform nodig hebben zodat onze speler erop kan staan. Vervolgens hebben we eerst een speler nodig voor dat de platformen met Dogoos gegenereerd worden zodat zei ook de speler kunnen "zien" en ten slotte moet het eindplatform ook helemaal op het einde, na de platformen, komen.

```
void Start()
{
    startPositie = new Vector3(0, 0, 0);

    vorigPlatform = null;

    breedteEinde = getLengte(einde, "x");
    lengteStart = getLengte(start, "z");

    SpawnStart();
    SpawnSpeler();
    SpawnPlatforms(aantal);
    SpawnEinde();
}
```

Afbeelding 89 - Start

De eerste functie die we nu gaan maken is de GetLengte functie. Deze functie heeft 2 variabelen nodig om een lengte te kunnen meten, namelijk het object dat gemeten moet worden en in welke richting dit moet gemeten worden. Verder moet het ook een kommagetal kunnen teruggeven, dus gebruiken we een float.

```
private float GetLengte(GameObject gameObject, string richting)
{

}
```

Afbeelding 90 - GetLengte

Hierin gaan we als eerste de modellen ophalen van het object. Het onderdeel in Unity dat de modellen zichtbaar maakt noemt men een Renderer. Omdat een prefab uit meerdere modellen bestaat gaan we een array (een soort van lijst) maken met alle renderers die in de prefab voorkomen en in alle objecten (children) die in deze prefab thuishoren.

```
Renderer[] renderers = gameObject.GetComponentsInChildren<MeshRenderer>();
```

Afbeelding 91 - Renderers zoeken

Daarna controleren we eerst of de lijst wel bestaat vooraleer we verder gaan met berekenen door te kijken of de lengte van de lijst gorter is dan 0. Als dit waar is kunnen we met het rekenen beginnen. Eerst gaan we een variabele maken van de Bounds (de afmetingen) van de eerste Renderer om vervolgens de afmetingen van alle Renderers in de lijst hierbij op te tellen Vervolgens kijken we welke afmeting we willen hebben met de "richting" variabele om dan de juiste afmeting terug te sturen.

Programmeertip: Als je heel veel verschillende if en else if lijnen achter mekaar hebt waarin telkens gecontroleerd wordt of iets gelijk is aan iets anders, kan je dit ook vervangen door een switch zoals in afbeelding 92 te zien is.

```
if (renderers.Length > 0)
{
    Bounds bounds = renderers[0].bounds;
    for (int i = 1, ni = renderers.Length; i < ni; i++)
    {
        bounds.Encapsulate(renderers[i].bounds);
    }
    switch (richting)
    {
        case "z":
            return bounds.size.z;
        case "y":
            return bounds.size.y;
        case "x":
            return bounds.size.x;
        default:
            return 0;
    }
}
else
{
    return 0;
}
```

Afbeelding 92 - Afmetingen van de Renderers bepalen en de juiste afmeting terugsturen

Als er een verkeerde richting is meegegeven of als er geen Renderers zijn, wordt er gewoon 0 teruggestuurd.

Nadat de GetLengte functie afgewerkt is, kunnen we beginnen aan al de Spawn-functies. We zullen deze in dezelfde volgorde maken of aanpassen zoals ze in de Start functie voorkomen.

De eerste en makkelijkste functie wordt SpawnStart. In deze functie moeten er maar 2 dingen gebeuren, namelijk start genereren op de startPositie en start 180 graden draaien. Dit laatste heb je misschien al gemerkt toen je naar achter moest kijken om de rij met platformen te zien.

Hoe je een object genereereerd weet je nu al wel, maar om dit daarna te draaien werken we met de Rotate functie. Deze functie werkt op de transform (de positie/rotatie van het object) en draait het object dan volgens een een aantal graden rond de X,Y of Z as. Om 180 graden te draaien wordt het startplatform hierdoor rond de Y-as gedraaid.

```
private void SpawnStart()
{
    GameObject startObject = Instantiate(start, startPositie, Quaternion.identity);
    startObject.transform.Rotate(new Vector3(0,180,0));
}
```

Afbeelding 93 - SpawnStart

Vervolgens gaan we de SpawnSpeler functie maken. Deze werkt net hetzelfde als de SpawnStart functie, alleen wordt de Y-positie met 1 verhoogd omdat de speler anders in het platform gegenereerd zou worden.

```
private void SpawnSpeler()
{
    Vector3 spelerpositie = startPositie;
    spelerpositie.y += 1;

    GameObject Speler = Instantiate(speler, spelerpositie, Quaternion.identity) as GameObject;
    Speler.transform.Rotate(new Vector3(0, 180, 0));
}
```

Afbeelding 94 - SpawnSpeler

Als dit gedaan is, gaan we de SpawnPlatforms functie aanpassen. Ten eerste wordt bij de positie variabele in het begin van de functie al meteen de lengte van het startplatform opgetelt.

```
Vector3 positie = startPositie;
positie.z += lengteStart;

GameObject gameObject = null;
```

Afbeelding 95 - Variabelen

Vervolgens voegen we in de for-loop ook nog een extra variabele toe, namelijk de bool isLift. Deze wordt ook direct berekend met de SetBool functie.

Hierna gaan we eerst controleren of we het eerste platform gaan genereren. Indien dit waar is, maken we de Z-positie een klein beetje hoger zodat het platform niet in het startplatfrom kan terecht komen. De positiebepaling die we in 4.2.1 hebben gemaakt moet in het gedeelte gezet worden indien dit niet waar is (de for-loop maakt het tweede platfrom enzovoort). Vervolgens wordt hier nog een extra controle bijgeplaatst om te controleren of het vorige platform een lift is. Indien dit waar is zal het platform een beetje hoger en verder gegenereerd worden.

Ten slotte wordt er bij het genereren van de platformen ook eerst gecontroleert of het een lift is dat gegenereerd moet worden. Indien deze gegenereerd is, wordt deze ook 90 graden gedraaid.

```
bool isDogooPlatform = SetBool(kansDogoo);
bool isJumpPadPlatform = SetBool(kansJumppad);
bool isLift = SetBool(kansLift);

if (i == 0)
{
    positie.z += 6;
}
else
{
    if (vorigPlatform.CompareTag("JumpPadPlatform"))
    {
        positie.y += 30;
        positie.z += 15;
    }
    else if (vorigPlatform.CompareTag("Lift"))
    {
        positie.y += 10;
        positie.z += 25;
    }
    else
    {
        positie.z += 15;
    }
}
```

Afbeelding 96 - Positie bepalen

```
if (isJumpPadPlatform) gameObject = skyplatformMetJumppad;
else if (isLift) gameObject = lift;
else if (isDogooPlatform) gameObject = skyplatformMetDogoo;
else gameObject = skyplatform;


GameObject platform = Instantiate(gameObject, positie, Quaternion.identity) as GameObject;
if (platform.CompareTag("Lift")) platform.transform.Rotate(new Vector3(0, -90, 0));

vorigPlatform = platform;
```

Afbeelding 97 - Platform genereren

Ten slotte gaan we nog de SpawnEinde functie maken. Deze functie haalt eerst de positie van het vorige platform op om vervolgens een beetje verder een eindplatform te genereren.

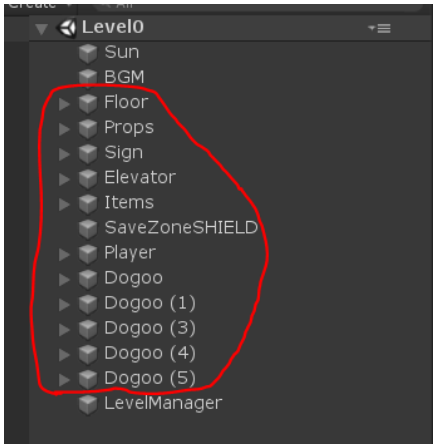
```
private void SpawnEinde()
{
    Vector3 eindePositie = vorigPlatform.transform.position;
    eindePositie.z += breedteEinde - breedtePlatform;
    Instantiate(einde, eindePositie, Quaternion.identity);
}
```

Afbeelding 98 - SpawnEinde

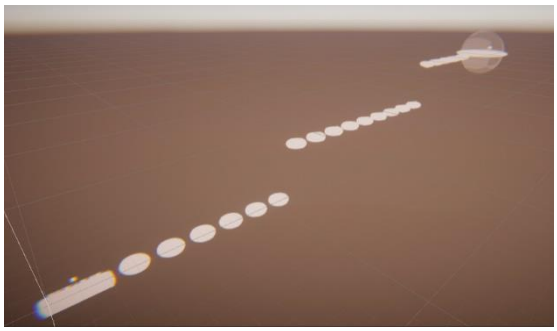
4.3.2 Script testen

Om het script te kunnen testen moeten we zoals bij de vorige 2 hoofdstukken eerst de nodige variabelen toevoegen in Unity. Na dat dit gebeurt is kan je best nog niet op de "play" knop duwen want we moeten eerst nog enkele dingen aanpasssen aan de scène.

Zo mag je uit de scène alles buiten de Sun, BGM en LevelManager objecten verwijderen. Deze zullen automatisch aangemaakt worden door het LevelManager Script.



Afbeelding 99 - Objecten verwijderen



Afbeelding 100 - Hoera! Een heel level

4.4 Willekeurige posities

4.4.1 Script

Als allerlaatste onderdeel van het automatisch level genereren gaan we de posities ook automatisch laten genereren op een iets meer willekeurigere plaats zodat niet alle platformen in een mooie rechte lijn liggen.

Zoals altijd gaan we eerst enkele nieuwe variabelen toevoegen aan het begin van het script.

```
[SerializeField] private GameObject speler;
[SerializeField] private GameObject start;
[SerializeField] private GameObject skyplatform;
[SerializeField] private GameObject skyplatformMetDogoo;
[SerializeField] private GameObject skyplatformMetJumppad;
[SerializeField] private GameObject lift;
[SerializeField] private GameObject einde;

[SerializeField] private int aantal;
[SerializeField] private int kansDogoo;
[SerializeField] private int kansLift;
[SerializeField] private int kansJumppad;

[SerializeField] private float afstandX;
[SerializeField] private float afstandY;
[SerializeField] private float afstandZ;

private float breedteEinde;
private float lengteStart;
private float breedtePlatform;
private float hoogtePlatform;

private Vector3 startPositie;
private GameObject vorigPlatform;
```

Afbeelding 101 - Veel variabelen

- *afstandX:* De maximum afstand tussen de platformen op de X-as.
- *astandY:* De maximum afstand tussen de platformen op de Y-as.
- *afstandZ:* De maximum afstand tussen de platformen op de Z-as.
- *breedtePlatform:* De breedte van een normaal platform.
- *hoogtePlatform:* De hoogte van een normaal platform.

De bredtePlatform en hoogtePlatform worden vervolgens in Start gedefinieerd zoals we met lengteStart en breedteEinde hebben gedaan.

```
void Start()
{
    startPositie = new Vector3(0, 0, 0);

    vorigPlatform = null;

    breedteEinde = GetLengte(einde, "x");
    lengteStart = GetLengte(start, "z");
    breedtePlatform = GetLengte(skyplatform, "x");
    hoogtePlatform = GetLengte(skyplatform, "y");

    SpawnStart();
    SpawnSpeler();
    SpawnPlatforms(aantal);
    SpawnEinde();
}
```

Afbeelding 103 - Start

Hierna gaan we eerst het gedeelte aanpassen van de positie bepaling van de gewone (niet lift of jumppad) platformen. Daar gaan we eerste de vorige positiebepaling verwijderen (zie onderaan afbeelding 96). Vervolgens maken we een nieuwe variabele, namelijk de float afstand. En definiëren we deze met de SetAfstand functie die we hierna gaan maken. De SetAfstand functie heeft hiervoor de positie nodig.

Programmertip: Indien je bij een functie een waarde terugstuurd maar tegelijk ook de waarde van de variabele wil aanpassen, kan je gebruik maken van *ref*.

```
float afstand = SetAfstand(ref positie);
```

Afbeelding 104 - Variabelen

Vervolgens gaan we de SetAfstand functie maken. Deze heeft een positie variabele (Vector3) die ook van ref voorzien is zodat deze mee aanpast.

In deze functie gaan we eerst het hoogteverschil tussen de 2 platformen laten berekenen. Hiervoor maken we opnieuw gebruik van Random.Range om deze een willekeurige waarde te laten aanemen. Deze ligt tussen -afstandY en de helft van hoogtePlatform. Dit wil zeggen dat het platform tot en met de maximum afstand lager kan liggen dan het vorige platform en een half platform hoger kan liggen dan het vorige platform. Dit laatste is om ervoor te zorgen dat de speler nog steeds op het platform kan springen.

```
private float SetAfstand(ref Vector3 positie)
{
    float y = vorigPlatform.transform.position.y + Random.Range(-afstandY, hoogtePlatform/2);
```

Afbeelding 105 - Hoogteverschil berekenen

Het volgende deel van de functie is iets ingewikkelder. Hierbij gaan we eerst de X en Z positie van het vorige platform ophalen om vervolgens hiermee een nieuwe positie te bereken. De X positie kan door de breedtePlatform variabele links of rechts van het vorige platform liggen. De Z positie kan van 0 (naast het vorige platform) tot de maximum afstand gaan.

```
float vorigPlatformX = vorigPlatform.transform.position.x;
float vorigPlatformZ = vorigPlatform.transform.position.z;

float x = vorigPlatformX + Random.Range(-breedtePlatform, breedtePlatform);
float z = vorigPlatformZ + Random.Range(0, afstandZ);

positie = new Vector3(x, y, z);
```

Afbeelding 106 - Nieuwe positie bepalen

Het laatste stuk van deze functie berekent vervolgens de afstand tussen de 2 posities (zonder rekening te houden met het hoogteverschil. Hierbij wordt gebruik gemaakt van de SqrMagnitude methode. Deze voert een wiskundige berekining uit die iets nauwkeuriger is dan de Distance methode. Als je meer uitleg wilt over deze methode mag je mij altijd een mailtje sturen.

```
Vector2 positieVerschil = new Vector2(vorigPlatformX, vorigPlatformZ) - new Vector2(x, z);
float afstand = positieVerschil.SqrMagnitude();

return afstand;
```

Afbeelding 107 - Afstand berekenen

Ten slotte voegen we bij het positie bepalen gedeelte van de SpawnPlatforms functie nog enkele lijnen toe.

```
float afstand = SetAfstand(ref positie);

while (afstand < breedtePlatform * breedtePlatform || afstand > afstandX * afstandX || positie.y <= -15)
{
    afstand = SetAfstand(ref positie);
}
```

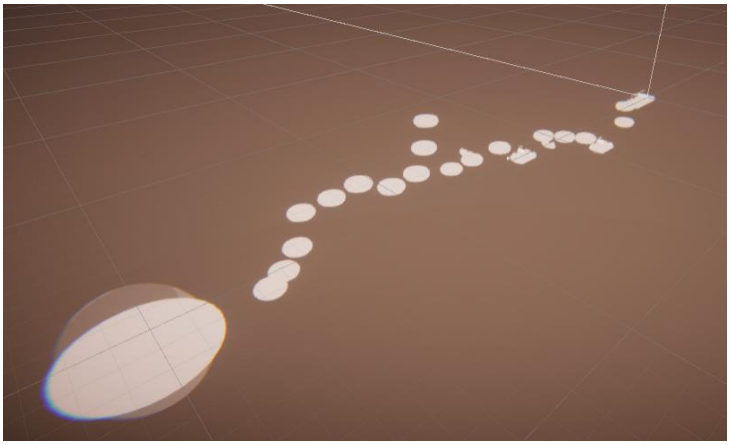
Afbeelding 108 - Positie bepalen in SpawnPlatforms

Deze zijn ook iets te moeilijk om in detail uit te leggen. Maar om het in makkelijke woorden uit te leggen zorgt dit deel ervoor dat de afstand nooit kleiner kan zijn dan de breedte van het platform of de maximum afstand. Ook gaat het platform nooit onder het -15 punt gegenereerd worden.

4.4.2 Script testen

Als je alles hiervan hebt afgewerkt is het weeral tijd om alle variabelen in te vullen. In afbeelding 109 kan je de waardes zien die ik hiervoor gebruikt heb. Deze mag je natuurlijk zelf aanpassen naar hoe jij ze wilt hebben.

Afbeelding 109 - Variabelen in Unity



Afbeelding 110 - En voila! Het automatisch level genereren is klaar!

5 MENU'S

Een game zonder pauze- of hoofdmenu is ook niet zo fijn. Anders wordt je meteen in het spel gezet van het moment dat je je game opzet en kan je ook niet even pauzeren.

5.1 Hoofdmenu

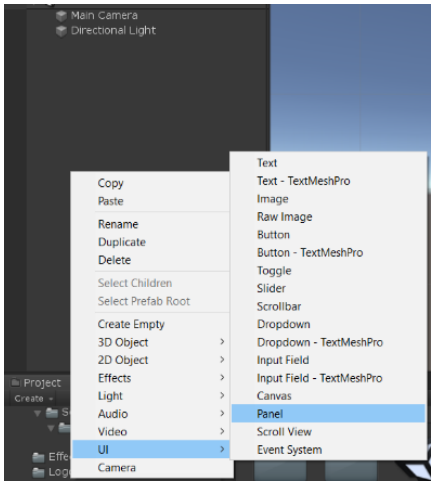
Allereerst zullen we beginnen met het hoofdmenu te maken. Dit is normaal gezien het eerste wat de speler ziet als hij/zij een game opzet. Een hoofdmenu bestaat dan ook uit een paar verschilllende knoppen:

- *Play:* Om het spel te starten.
- *Options:* Om enkele instellingen van het spel te veranderen.
- *Quit:* Om het spel terug af te sluiten.

Afbeelding 111 - Het hoofdmenu dat we gaan maken

5.1.1 Het uitzicht

Om het hoofdmenu te maken, gaan we eerst een nieuwe scène hiervoor aanmaken. Hiervoor kan je bijvoorbeeld in de scenes map een nieuwe map aanmaken met daarin de nieuwe scene. Ik heb ze allebei Main Menu genoemd.

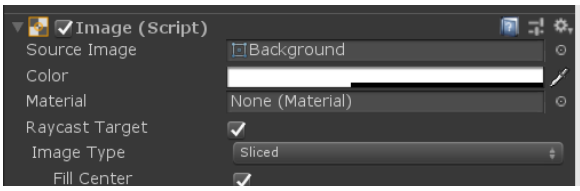


Afbeelding 112 - Een panel aanmaken

5.1.1.1 Achtergrond

De makkelijkste manier om een mooie achtergrond te krijgen die op het volledige scherm past is door een Panel te maken. Dit doe je door rechts te klikken in het levelmenu, naar UI te gaan met je muis en vervolgens op Panel te klikken. Als je panel gemaakt is, kan je de naam naar bijvoorbeeld "achtergrond" veranderen zodat je dit makkelijk kan herkennen.

Nadat dit gebeurt is, gaan we enkele instellingen van deze panel aanpassen zodat we een mooie achtergrond krijgen. Hiervoor selecteer je eerst het panel object in het levelmenu zodat je rechts de eigenschappen kan zien. Daar gaan we de Image (afbeelding) van het panel aanpassen.



Afbeelding 113 - De afbeeldingeigenschap

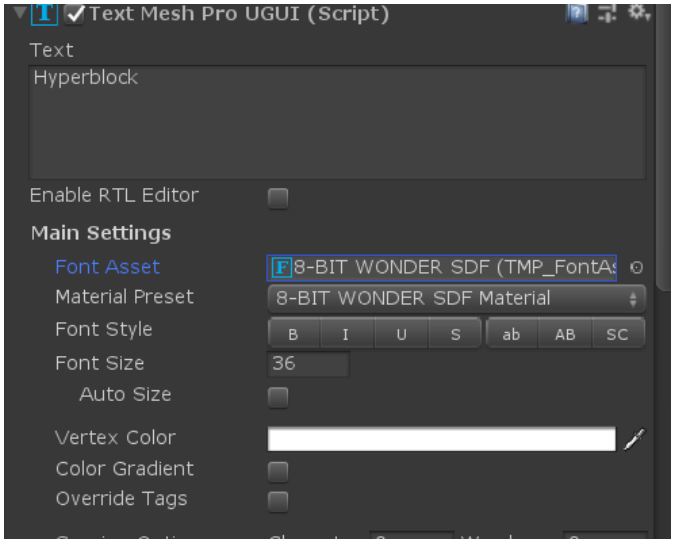
Ten eerste gaan we zorgen dat we een mooie achtergrond krijgen. Dit kan je doen door op het kleine cirkeltje naast het vak van de Source Image (bronafbeelding) te klikken. Tussen de lijst die daar verschijnt zou normaal de achtergrondafbeelding tussen staan.

Vervolgens lijkt de afbeelding een beetje doorzichting. Dit kunnen we aanpassen door de Alpha (doorzichtigheid) van de kleur aan te passen. Bij het aanmaken van de panel is dit op 100 gezet, dit mag je veranderen naar 255. Dit is de maximum waarde waardoor de achtergrond volledig zichtbaar word.

Afbeelding 114 - De Alpha staat hier op 100

5.1.1.2 Titel

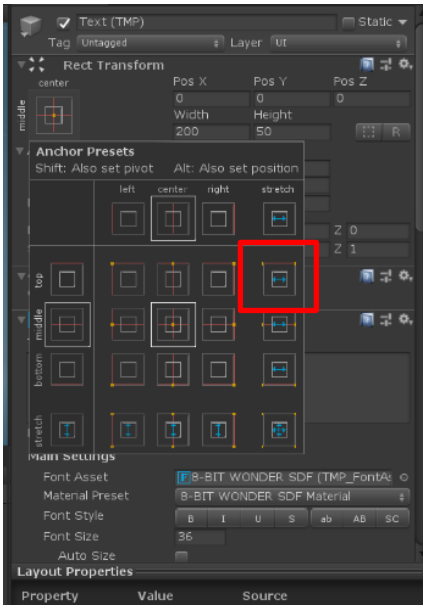
Na dat we een mooie achtergrond hebben, kunnen we de titel en knoppen maken. Voor deze onderdelen gaan we gebruik maken van TextMesh Pro. Dit is een verbeterd alternatief van de gewone Text van Unity.



Afbeelding 115 - Tekstvak aanpassen

Net zoals een panel, kan je nu ook een tekstvak toevoegen. Dit doe je door rechts te klikken op het canvas object in plaats van in het levelmenu en "Tekst – TextMesh Pro" te selecteren. Als dit pakket nog niet geïnstalleerd is, zal Unity automatisch vragen of je dit wilt toevoegen.

Als je het tekstvak object selecteert, kan je opnieuw rechts alle waardes aanpassen. Zo kan je daar de tekst aanpassen naar bijvoorbeeld "Hyperblock" en ook het letterype door op het cirkeltje naast het Font Asset vak te klikken. Hiervoor heb ik het letterype van de oorspronkelijke UI gebruikt.



Afbeelding 116 - Positie aanpassen

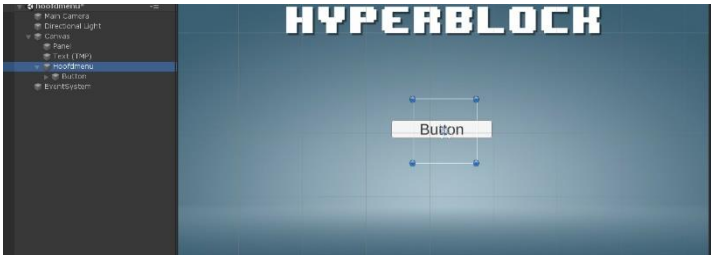
Als je dit hebt aangepast kan je bovenaan ook nog de positie van het tekstvak vastleggen. Door op het vierkantje linksboven de drukken kan je kiezen uit verschillende vastliggende posities. Als je hierbij ALT indrukt wordt ook de positie en grootte van het tekstvak automatisch aangepast. Om het tekstvak bovenaan vast te leggen en het de hele breedte van het scherm te laten innemen, kan je klikken op het rood omcirkelde icoon van afbeelding 116.

Natuurlijk kan je hier ook de naam van het tekstvak veranderen naar bijvoorbeeld "Titel" om dit makkelijker te herkennen.

5.1.1.3 Knoppen

Het belangrijkste onderdeel van een menu zijn natuurlijk de knoppen.

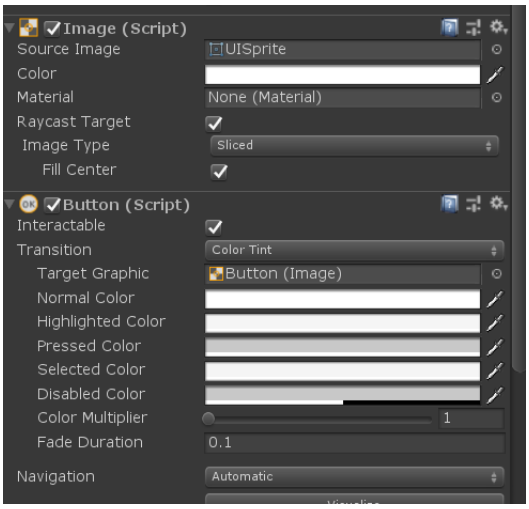
Omdat het menu uit 2 delen bestaat, namelijk het hoofdmenu en het opties menu, gaan we eerst 2 Empty objecten aanmaken om als groep te dienen. Hiervan gaan we eerst enkel de eerste aanmaken, namelijk "Hoofdmenu".



Afbeelding 117 - Een knop zonder aangepast uiterlijk

Als je dit object hebt aangemaakt kan je hierin knopppen gaan aanmaken. Dit doe je op dezelfde manier als een panel of tekstvak door rechts te klikken op het Hoofdmenu Object, op UI te klikken en dan op Button – TextMeshPro te klikken. Deze knop ziet er natuurlijk niet zo mooit uit, dus gaan we eerst het uiterlijk hiervan aanpassen.

Om hetzelfde uiterlijk en effect te krijgen als de knoppen die ik gemaakt heb moet je enkele dingen aanpassen.



Afbeelding 118 -De knopeigenschappen

Eerst wordt de kleur van de Image (achtergrond) op zwart gezet. Deze zorgt voor het effect dat de achtergrond een beetje donkerder wordt als je de knoppen selecteerd of er op drukt.

Hierna gaan we de kleuren van het Button gedeelte aanpassen. Eerst moet je de Alpha waarden van de Normal Color (als er niets met de knop gebeurd) op 0 zetten. Dit zorgt voor de onzichtbare achtergrond. Vervolgens zet je de Alpha waarde van Highlighted Color lichtjes hoger, ik heb deze rond 30 gezet. Als laatste zet je de Alpha waarde van de Pressed Color nog een beetje hoger, ik heb deze rond 70 gezet.

Afbeelding 119 - Het tekstgedeelte

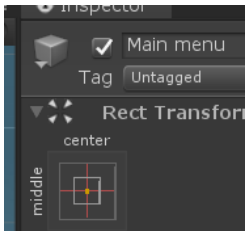
Als je dan op het pijltje links van het button object klikt kan je het tekstobject hiervan aanpassen. Dit kan je op ongeveer dezelfde manier als in 5.1.1.2 doen.

Hierna kan je best de naam van de knop aanpassen naar bijvoorbeeld PlayButton zoals ik gedaan heb. Zo kan je duidelijk zien welke knop voor wat gaat dienen.

Afbeelding 120 - Het hoofdmenu

Als je vervolgens rechtsklikt op de knop en op "Duplicate" klikt, maak je een kopie van de knop. Deze kan je dan op een andere plaats zetten, een andere naam geven en een andere tekst geven. Zo kan je de 3 knoppen "Play", "Options" en "Quit" aanmaken.

Als je deze hebt aangemaakt is het uitzicht van het hoofdmenu klaar. Het optiemenu kan je nu heel gemakkelijk ook maken, door het hoofdmenu object te dupliceren zoals je met de knoppen gedaan hebt. Om duidelijk te weten in welk object je bezig bent, kan je best het hoofdmenu desactiveren. Dit doe je door bij de eigenschappen linksvanboven het vinkje uit te klikken.



Afbeelding 121 - Het activatievinkje

In het optiemenu kan je dan bijvoorbeeld de "Play" en "Options"-knoppen verwijderen en in de plaats daarvan enkele tekstvakken maken in het pausemenu object met een "Options" titel en een "Difficulty" titel zoals in afbeelding 122. In dit optiemenu gaan we de speler kunnen laten kiezen welke moeilijkheid hij/zij wilt voor het spel. Ook kan je de Quit knop aanpassen naar een Back knop.

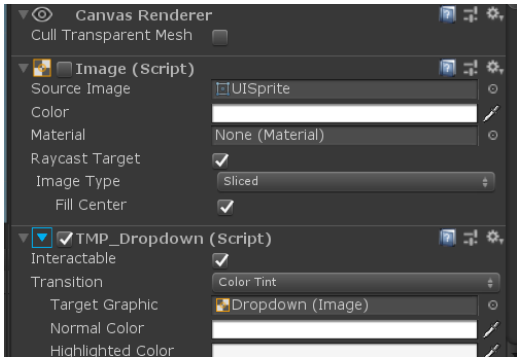
Om de speler te kunne laten kiezen tussen de verschillende moeilijkheden gaan we gebruik maken van een dropdown menu. Dit is een lijstje waaruit de speler kan kiezen tussen bijvoorbeeld "Easy", "Normal" of "Hard". Deze gaan we op ongeveer dezelfde manier doen als alle andere onderdelen toe te voegen door rechts te klikken op het pausemenu object, naar UI te gaan met je muis en op "Dropdown – TextMeshPro" te klikken.

Afbeelding 122 - Het optiemenu

Deze gaat er, net zoals de knoppen, niet zo mooi uitzien. Maar dit gaan we daarom nu aanpassen.

Eerst gaan we hiervoor het Image gedeelte van het dropdown opbject uitschakelen. Hierdoor wordt de achtergrond vna de dropdown verwijdert.

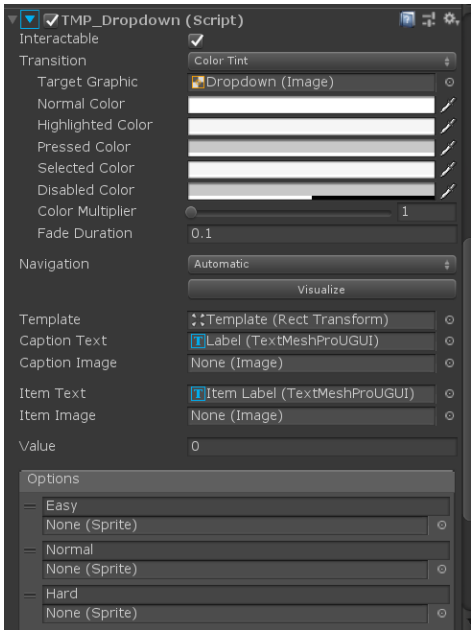
Hierna kan je op het pijltje duwen zodat de onderdelen van de dropdown zichtbaar worden. Daar selecteren we label en passen we de tekst aan zoals we bij de titel en knoppen hebben gedaan.



Afbeelding 123 - Image afzetten

Vervolgens selecteer je ook het Template onderdeel om ook daar het Image gedeelte af te zetten.

Als je daarna terug het dropdown opbject selecteerd, kan je de verschillende opties van de dropdown aanpassen. Als je daar bij de eigenschappen naar beneden scrollt, kom je bij het Options gedeelte. Hier kan je de opties aanpassen naar "Easy", "Normal" en "Hard" zoals in afbeelding 124.



Afbeelding 124 - Opties aanpassen

Ten slotte kan je het optiesmenu terug desactiveren en het hoofdmenu terug activeren zodat het er terug als een normaal hoofdmenu uitziet.

5.1.2 Script

Omdat het hoofdmenu verschillende dingen moet doen heeft het natuurlijk ook een script nodig. Dit script kan je in de UI map van de Scripts map aanmaken. Na het een passende naam te geven kan je het vervolgens toevoegen aan het hoofdmenu object en het openen.

Voor dit script zijn gelukkig geen variabelen nodig, dus deze hoef je niet toe te voegen. Wel moet je bovenaan het UI gedeelte en het SceneManagement gedeelte toevoegen omdat we deze nodig gaan hebben om het menu te laten werken. Ook kan je de Start en Update functie verwijderen omdat we deze niet gaan gebruiken.

```
using UnityEngine.SceneManagement;
using UnityEngine.UI;

public class MainMenu : MonoBehaviour
{

}
```

Afbeelding 125 - Het "lege" hoofdmenu script

Nu kan je als eerste een Awake functie toevoegen. Daarin zorg je ervoor dat je je muis terug kan gebruiken als je naar het hoofdmenu terugkeert vanuit de game.

```
private void Awake()
{
    Cursor.visible = true;
    Cursor.lockState = CursorLockMode.None;
}
```

Afbeelding 126 - Awake

Hierna gaan we enkele eigen functies maken die we aan de knoppen kunnen toevoegen, namelijk PlayGame voor de Play knop, QuitGame voor de Quit knop en ChangeDifficulty (verander de moeilijkheid) voor de dropdown in het opties menu.

Als eerste gaan we PlayGame aanmaken. Dit is een zeer simpele functie die gewoon de scène van het level laad.

```
public void PlayGame()
{
    SceneManager.LoadScene(1);
}
```

Afbeelding 127 - PlayGame

De QuitGame functie is ook heel makkelijk. Deze moet gewoon de applicatie afsluiten door middel van de Appication.Quit functie.

```
public void QuitGame()
{
    Application.Quit();
}
```

Afbeelding 128 - QuitGame

De ChangeDifficulty functie gaat elke keer opgeroepen worden als er een ander item uit de lijst van de dropdown geselecteerd wordt. Deze haalt eerst de tekst op die geselecteerd is om deze vervolgens door te sturen naar een functie in de GameInfo klasse. Deze functie zullen we na het hoofdmenu aanmaken.

```
public void ChangeDifficulty(TMPro.TMP_Dropdown dropdown)
{
    string difficulty = dropdown.options[dropdown.value].text;
    AfgewerktGameInfo.ChangeDifficulty(difficulty);
}
```

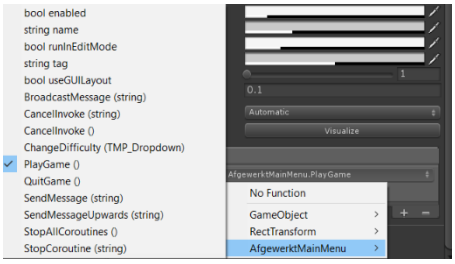
Afbeelding 129 - ChangeDifficulty

5.1.3 Script toevoegen aan het menu

Nadat we het script hebben aangemaakt is het tijd er voor te zorgen dat de knoppen in het menu ook iets kunnen doen.

We zullen eerst beginnen met de makkelijkste knoppen, namelijk Play en Quit.

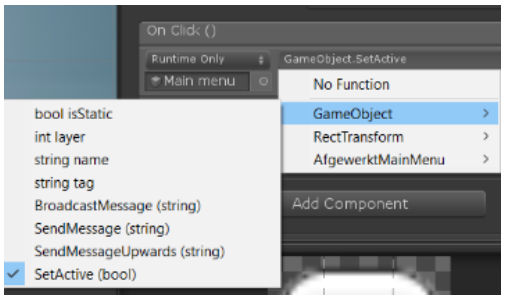
Als je bijvoorbeeld de Play knop selecteerd en naar beneden scrollt in het eigenschappenveld zie je een gedeelte dat "On Click()" noemt. Dit stuk wordt elke keer opgeroepen als er op de knop geduwt wordt. In het linkse vakje hiervan kan je het hoofdmenu object slepen (of op het cirkeltje klikken en het hoofdmenu object uit de lijst selecteren) zodat je aan het script kan dat hier bij hoort. Als je vervolgens op de knop aan de rechterkant klikt, krijg je een klein lijstje waar normaal gezien ook de naam van het script uit 5.1.2 staat. Als je dit selecteerd kan je daarna ook de juiste methode aanklikken, wat hier "PlayGame" is.



Afbeelding 130 - PlayGame toevoegen

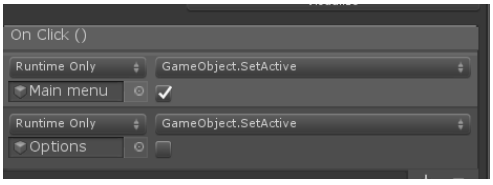
Op dezelfde manier kan je ook de QuitGame methode toevoegen aan de Quit knop.

De Option knop is iets anders, maar daarom ook niet moeilijker. Ook hier sleep je het hoofdmenu object in hetzelfde vak. Maar in plaats van daarna het script te selectere, klik je deze keer op "GameObject" en op "SetActive". Dit zorgt ervoor dat je het vinkje van afbeelding 121 kan aanduiden of niet door op de knop te drukken. Dit vinkje mag je uitzetten aangezien ja naar het optiemenu gaat. Daarna kan je op het "+" icoontje duwen onderaan het "On Click()" gedeelte om een ander object toe te kunne voegen. Hier gaan we het pauzemenu inslepen en het vinkje aanduiden.



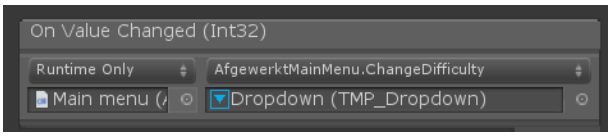
Afbeelding 131 - SetActive

In het pauzemenu gaan we juist het omgekeerde doen met de Back knop. Hier ga je ook het pausemenu en hoofdmenu object toevoegen, maar ga je de vinkjes omgekeerd zetten zoals in afbeelding 132.



Afbeelding 132 - De Back knop

In het dropdown object is er geen "On Click()" gedeelte, maar is dit een "On Value Changed()" gedeelte. Dit deel wordt elke keer opgeroepen als er een ander item word geselecteerd in de lijst. Hierin kan je ook het hoofdmenu object toevoegen en daarna de juiste methode selecteren. Deze keer is dit "ChangeDifficulty". Hierdoor zal er nog een vak verschijnen, waar je gewoon het Dropdown object zelf in mag slepen.



Afbeelding 133 - On Value Changed

Als dit gedaan is zijn alle knoppen correct aangepast.

5.1.4 GameInfo

5.1.4.1 GameInfo

Zoals in 5.1.2 al gezien is, wordt met de dropdown een functie opgeroepen in de GameInfo klasse. Deze gaan we nu aanmaken. Maar eerst is het weer tijd voor enkele variabelen.

```
public static int aantalPlatforms = 20;
public static int kansDogoo = 10;
public static int kansJumppad = 5;
public static int kansLift = 5;
```

Afbeelding 134 - Variabelen

De variabelen komen deze keer misschien bekend voor omdat het enkele variabelen zijn die in het LevelManager script gebruikt worden. In dat script gaan we daarom hierna enkele aanpassingen maken.

Als de variabelen aangemaakt zijn is het tijd om de functie aan te maken die in het hoofdmenu script opgeroepen werd. Deze functie gaat met de tekst die ze ontvangt de waardes aanpassen in de GameInfo klasse. Als je mijn programmeertips hebt gelezen zal je hier ook zien dat je deze code ook perfect kan vervangen door een switch. Ook kan je de verschillende waardes hiervan aanpassen naar de waardes die jij wilt.

```
public static void ChangeDifficulty(string difficulty)
{
    if (difficulty == "Easy")
    {
        aantalPlatforms = 20;
        kansDogoo = 10;
        kansJumppad = 5;
        kansLift = 5;
    }
    else if (difficulty == "Normal")
    {
        aantalPlatforms = 30;
        kansDogoo = 20;
        kansJumppad = 10;
        kansLift = 10;
    }
    else if (difficulty == "Hard")
    {
        aantalPlatforms = 40;
        kansDogoo = 40;
        kansJumppad = 20;
        kansLift = 20;
    }
}
```

Afbeelding 135 - ChangeDifficulty

5.1.4.2 LevelManager

Om de moeilijkheidsgraad ook echt in de game zelf toe te passen moeten we ook het LevelManager script aanpassen. Gelukkig is dit helemaal niet zo moeilijk om te doen.

Het enige wat je hiervoor moet aanpassen is dat je de variabelen in de Start functie gaat definiëren door ze op te halen uit de GameInfo klasse.

```
aantal = AfgewerktGameInfo.aantalPlatforms;
kansDogoo = AfgewerktGameInfo.kansDogoo;
kansLift = AfgewerktGameInfo.kansLift;
kansJumppad = AfgewerktGameInfo.kansJumppad;
```

Afbeelding 136 - Variabelen definiëren

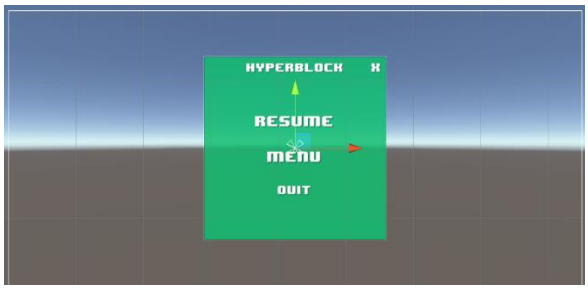
Als je alles goed hebt meegevolgd zou je nu normaal gezien een werkend hoofdmenu hebben waarbij je ook de moeilijkheid van je game kan aanpassen.

5.2 Pauzemenu

Een game moet je natuurlijk ook kunnen pauzeren voor als je even iets anders moet doen. Een pauzemenu maken is gelukkig helemaal niet zo moeilijk.

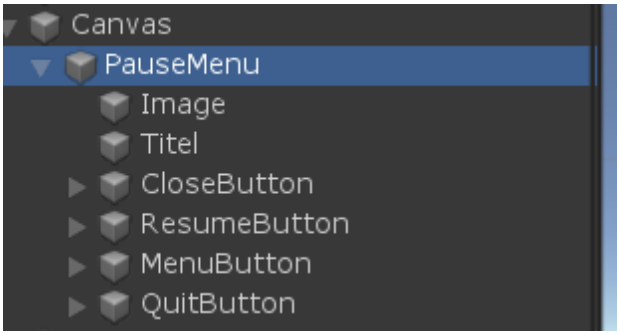
5.2.1 Het uitzicht

Voor het pauzemenu hoef je geen nieuwe scène aan te maken, maar kan je dit gewoon aanmaken in de level scène. Als je meerdere levels zou hebben kan je dit ook eventueel op je speler prefab aanmaken, maar aangezien deze game voorlopig maar 1 level heeft (dat automatisch gegenereerd wordt dus elke keer anders is) gaan we het in de levelscène aanmaken.



Afbeelding -137 - Het pauzemenu

In tegenstelling tot het hoofdmenu gebruik je hier een Image als achtergrond voor het pauzemenu. Deze kan je ook eventueel eerst in een empty object zetten zodat je makkelijk alle elementen van het pauzemenu kan aan- of uitzetten.



Afbeelding 138 - Het pauzemenu object

Op dezelfde manier als in het hoofdmenu kan je hier enkele knoppen aanmaken voor het terug starten van de game, het sluiten van het pauzemenu, om terug te gaan naar het hoofdmenu en om het spel te stoppen.

5.2.2 Script

Net als het hoofdmenu heeft het pauzemenu ook een script nodig om iets te kunnen doen. Dit script gaan we deze keer echter aan het canvas object toevoegen omdat in het script het pauzemenu aan of uit wordt gezet. Als het pauzemenu uit staat, kan het script ook niet werken.

Ook hier maakt het script gebruik van het SceneManagement gedeelte van Unity dus zorg je dat je dit eerst bovenaan toevoegd. Daarna kan je de Start functie ook verwijderen omdat we deze niet gaan gebruiken.

Vervolgens is het weer tijd voor variabelen.

```
public static bool gameIsPaused = false;
[SerializeField] private GameObject pauseMenu;
```

Afbeelding 139 - Variabelen

- *gameIsPause:* Bool om na te kijken of de game gepauzeerd is. Deze staat op public zodat je die nog kan gebruiken in andere scripts.
- *pauseMenu:* GameObject. Het pauzemenu object

-

Daarna gaan we de Update functie aanpasen. Hierin gaan we controleren of er een knop ingeduwd wordt waarmee we de game hebben gepauzeerd. Hiervoor heb ik P gekozen. Als deze knop wordt ingeduwd gaat de game eerst nakijken of de game al gepauzeerd is of niet en dan de Play of Pause functie uitvoeren.

```
void Update()
{
    if (Input.GetKeyDown(KeyCode.P))
    {
        Debug.Log("pause");
        if (gameIsPaused)
        {
            Play();
        }
        else
        {
            Pause();
        }
    }
}
```

Afbeelding 140 - Update

De Play en Pause functies zijn beide heel simpele functies die exact het omgekeerde van elkaar doen. De Play functie zet eerst de tijd terug op een normale snelheid, zorgt dan voor dat het pauzemenu object afgezt wordt en dat het script weet dat de game niet meer gepauzeerd is. Hierna zorgt het er ook nog voor dat de muis terug onzichtbaar en in de game is.

```
public void Play()
{
    Time.timeScale = 1f;
    pauseMenu.SetActive(false);
    gameIsPaused = false;

    Cursor.visible = false;
    Cursor.lockState = CursorLockMode.Locked;
}
```

Afbeelding 141 - Play

De Pause functie doet exact het omgekeerde van al deze dingen. Zo zet het de tijd stil, wordt het pauzemenu geactiveerd en zorgt het dat het script weet dat de game gepauzeerd is. Ook wordt de muis terug zichtbaar gezet.

Voor de andere knoppen maken we ook enkele functies. Deze werken op ongeveer dezelfde manier als de functies van de knoppen in het hoofmenu.

```
private void Pause()
{
    Time.timeScale = 0f;
    pauseMenu.SetActive(true);
    gameIsPaused = true;

    Cursor.visible = true;
    Cursor.lockState = CursorLockMode.None;
}
```

Afbeelding 142 - Pause

GoToMenu zet eerst de tijd terug op normale snelheid aangezien dit op 0 wordt gezet als je naar het pauzemenu gaat. Hierna gaat het de hoofmenuscène laden.

De QuitGame functie werkt op exact dezelfde manier als de QuitGame functie van het hoofdmenu.

Als je al deze functies hebt gemaakt kan je ze gaan toevoegen aan de juiste knoppen.

- *Play:* Deze kan je toevoegen aan de sluitknop en de resumeknop.
- *GoToMenu:* Deze kan je toevoegen aan de menuknop
- *QuitGame:* Deze kan je toevoegen aan de quitknop.

```
public void GoToMenu()
{
    Time.timeScale = 1f;
    SceneManager.LoadScene(0);
}

public void QuitGame()
{
    Application.Quit();
}
```

Afbeelding 143 - GoToMenu & QuitGame

Als je dit allemaal gedaan hebt, heb je ook een pauzemenu in je game.

6 BESLUIT

Als je al de delen van deze lesbundel hebt gevolgd, heb je een bijna volledige game kunnen maken. Natuurlijk kan je aan deze game nog zo veel mogelijk dingen toevoegen of aanpassen dat je zelf wilt of kunt.

Als je nog meer wilt bijleren, wat ik zeker leuk zou vinden, kan je altijd naar een van de bronnen uit hoofdstuk 1.4 gaan. Veel plezier nog met het maken van games!