

Transferable Code Passion Project

Inhoudsopgave

Inleiding.....	1
Unity & Gitlab.....	2
Player Movement.....	2
Player Animations	3
Volgende Camera	5
Parallax Achtergrond Toevoegen	5
Talking NPC's	7
Switching Scenes/Titel Scherm.....	11
Spikes /Respawn.....	12
Camera Transitions.....	13
Attacking + Enemies	13
Muziek + Don't Destroy.....	15
Feedback	15
Reflectie.....	15

Inleiding

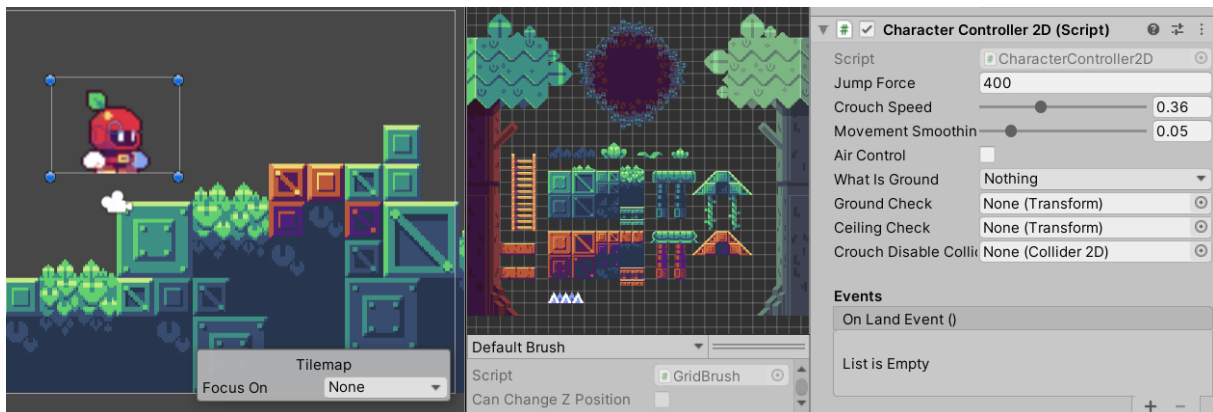
Voor mijn passion project maak ik een game in Unity. C# is een handige programmeertaal hiervoor. Ik heb verschillende scripts gemaakt voor de mechanics in mijn game. Zo moest ik een script maken om het personage te laten springen, lopen, praten, voor camera transitions etc. In dit document staat de code beschreven. Als je naar de code in Gitlab kijkt staan overall comments bij om te laten zien wat de code doet, dat is gedetailleerder dan hoe ik het in dit document beschrijf. De code van mijn Unity Game is te vinden op <https://git.fhict.nl/l437759/semester-3-media/-/tree/master/UnityScripts>

Unity & Gitlab

Door gebruik te maken van een .gitignore push ik niet alle overbodige dingen maar alleen de code van mijn Unity project. Hierdoor zal mijn repository niet overstroomd worden met onnodige dingen en zullen alleen de scripts met code te zien zijn. Ook is het handig om mijn code op Gitlab te zetten zodat ik altijd een backup heb en de docenten mijn code ook kunnen zien.

Player Movement

Om het spel te kunnen spelen moet je personage kunnen lopen en springen. Hiervoor moet je een aantal dingen opzetten voordat je kunt beginnen met coderen. Je moet colliders toevoegen aan de objecten waar je speler op gaat lopen zodat hij niet door het level heen valt. Je maakt een object aan voor je speler met dezelfde colliders, dit kan een blokje zijn of pixel art die je zelf hebt gemaakt. Verder voeg je een character controller toe in je Unity file en uiteindelijk voeg je een script toe. Een script is een stukje code die je kunt toevoegen aan een object, bijvoorbeeld de speler. Hier kunnen we dus beginnen met coderen en bijvoorbeeld coderen wanneer de speler beweegt. Deze sleep je uiteindelijk op je player object zodat Unity weet dat de code wordt toegepast op dat object. Je wilt niet dat je alle blokjes van positie laat veranderen maar alleen het personage.



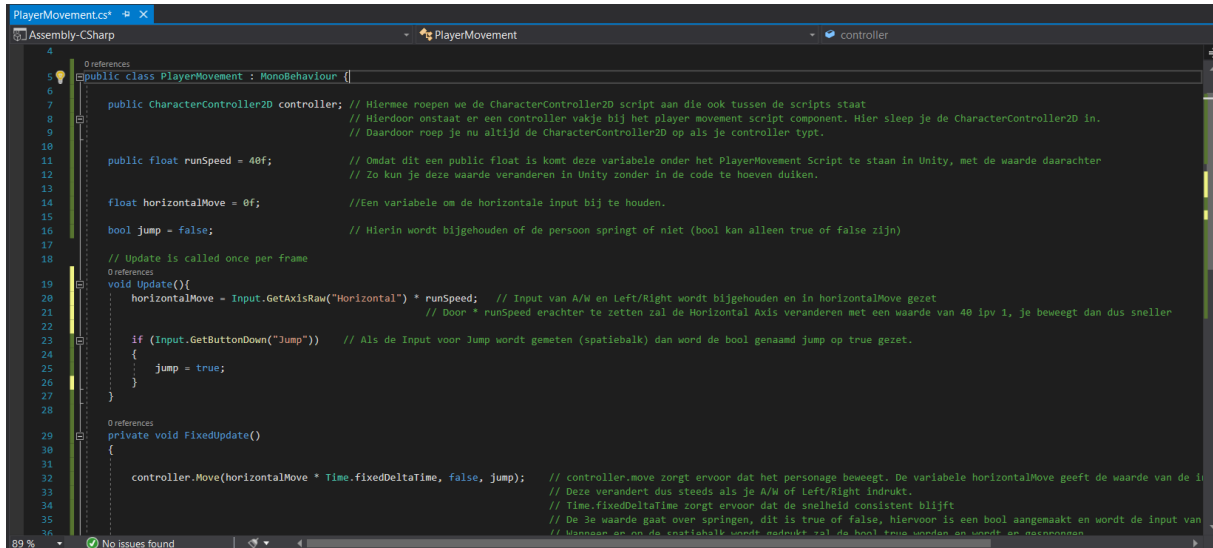
Je ziet dat je in de player controller een aantal variabelen kan veranderen zoals de Jump Force en Air Control. Dit kan omdat je deze variabelen public hebt gemaakt in je script. Je speelt hiermee totdat de movement goed en smooth aanvoelt, ook als je animaties gebruikt moet je ervoor zorgen dat de snelheid overeenkomt met de loop animatie. Als je wilt dat je personage in de lucht bestuurbaar is vink je Air Control aan. Zo zijn er enorm veel variabelen die je aan kan passen om het springen en bewegen beter aan te laten voelen. Hierom zijn mensen vaak erg lang bezig om een player controller te fine tunen.

Voor de Ground en Ceiling check (Check afbeelding) moet je nog extra objecten toevoegen, deze plaats je bij het hoofd van de speler en de voeten. Die bij de voeten sleep je in het vakje met Ground Check en die bij het hoofd sleep je in het vakje met Ceiling Check. Hierdoor kan Unity checken wanneer de speler de grond of een plafond raakt.

Verder blijft de speler hangen als de zijkant een muur raakt, om dit op te lossen maak je een Physics Material 2D aan en geef je deze een friction van 0, deze sleep je vervolgens op Material bij de box colliders van je personage.

Je moet dus in Unity zelf veel dingen regelen en slepen, maar hierachter zitten vaak vele scripts verstopt. In deze scripts staat de code die je schrijft en deze scripts kun je vaak op een object slepen. Hieronder zie je een deel van mijn PlayerMovement script. Hierin maak ik wat variabelen en functies. Ik check bijvoorbeeld of de Jump button wordt ingedrukt. Deze input (spatie) is automatisch verbonden met Jump door Unity. Dit kun je veranderen in Edit > proj. settings > Input.

Wanneer het script ziet dat deze button is ingedrukt voert hij de code eronder uit en zal de speler springen. Veel van deze dingen worden nog uit de PlayerController gehaald.



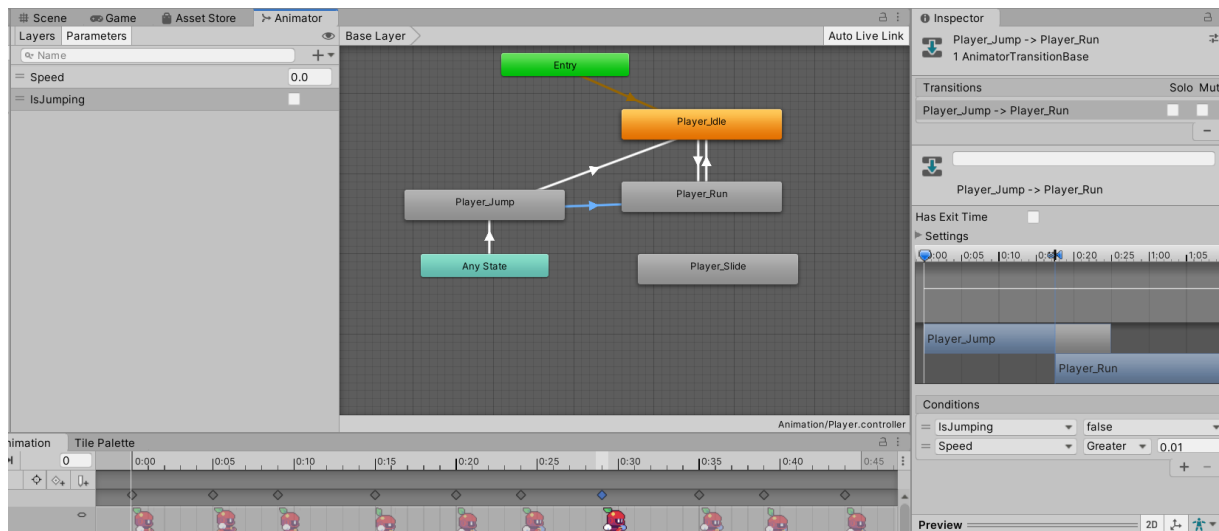
```
4 0 references
5 public class PlayerMovement : MonoBehaviour {
6
7     public CharacterController2D controller; // Hiermee roepen we de CharacterController2D script aan die ook tussen de scripts staat
8     // Hierdoor ontstaat er een controller vakje bij het player movement script component. Hier sleep je de CharacterController2D in.
9     // Daardoor roep je nu altijd de CharacterController2D op als je controller typt.
10
11     public float runSpeed = 40f; // Omdat dit een public float is komt deze variabele onder het PlayerMovement Script te staan in Unity, met de waarde daarachter
12     // Zo kun je deze waarde veranderen in Unity zonder in de code te hoeven duiken.
13
14     float horizontalMove = 0f; // Een variabele om de horizontale input bij te houden.
15
16     bool jump = false; // Hierin wordt bijgehouden of de persoon springt of niet (bool kan alleen true of false zijn)
17
18     // Update is called once per frame
19     0 references
20     void Update(){
21         horizontalMove = Input.GetAxisRaw("Horizontal") * runSpeed; // Input van A/W en Left/Right wordt bijgehouden en in horizontalMove gezet
22         // Door * runSpeed erachter te zetten zal de Horizontal Axis veranderen met een waarde van 40 ipv 1, je beweegt dan dus sneller
23
24         if (Input.GetButtonDown("Jump")) // Als de Input voor Jump wordt gemeten (spatiebalk) dan word de bool genaamd jump op true gezet.
25         {
26             jump = true;
27         }
28     }
29
30     0 references
31     private void FixedUpdate()
32     {
33         controller.Move(horizontalMove * Time.fixedDeltaTime, false, jump); // controller.move zorgt ervoor dat het personage beweegt. De variabele horizontalMove geeft de waarde van de 1
34         // Deze verandert dus steeds als je A/W of Left/Right indrukt.
35         // Time.fixedDeltaTime zorgt ervoor dat de snelheid consistent blijft
36         // De 3e waarde gaat over springen, dit is true of false, hiervoor is een bool aangemaakt en wordt de input van
37         // Wanneer en op de snelheid wordt ingedrukt zal de bool true worden en wordt de input van
```

Player Animations

Nu ziet het personage er nog stijf uit, hij beweegt wel van plek maar de afbeelding beweegt niet. Om meer diepgang te krijgen in het spel maak je gebruik van animaties. Deze animaties roep je op wanneer er een bepaalde input wordt gegeven, zo begint bijvoorbeeld de loop animatie wanneer er wordt gelopen.

Dit doe je door gebruik te maken van de Unity Animator en de Animation Window. Eerst maak je een animatie aan in de Animation Window. Je maakt de naam aan, bijvoorbeeld Player_Jump en sleept hier de juiste sprites op. Hier kun je de duratie en andere dingen aanpassen om de animatie te bewerken. Als je dit eenmaal voor elke animatie gedaan hebt kun je de Animator openen.

Hier kies je welke animaties er op welk moment afgespeeld worden (zie afbeelding hieronder). Dit doe je door transitions toe te voegen (de witte pijltjes). In deze transitions kun je aangeven wanneer deze transitie moet plaatsvinden en dus wanneer je van animatie moet switchen. We maken een parameter aan genaamd "Speed" en vertellen dat de ren animatie moet starten als de speed groter is dan 0.1. Om weer terug naar de idle animatie te gaan maken we een transitie (witte pijl) terug en vertellen dat deze animatie start wanneer de Speed kleiner is dan 0.1. Nu moeten we in het script alleen nog een waarde meegeven aan deze parameter. Gelukkig had ik al een variabele voor de snelheid gemaakt.



Je wilt vanuit elke animatie/state kunnen springen. In plaats van tussen elke animatie een transitie naar de Player_Jump te maken kun je dit doen vanuit de Any State balk (blauwe balk) hierdoor kun je op elk moment switchen naar de spring animatie.

```

animator.SetFloat("Speed", Mathf.Abs(horizontalMove));

if (Input.GetButtonDown("Jump")) // A
{
    jump = true;
    animator.SetBool("IsJumping", true);
}

```

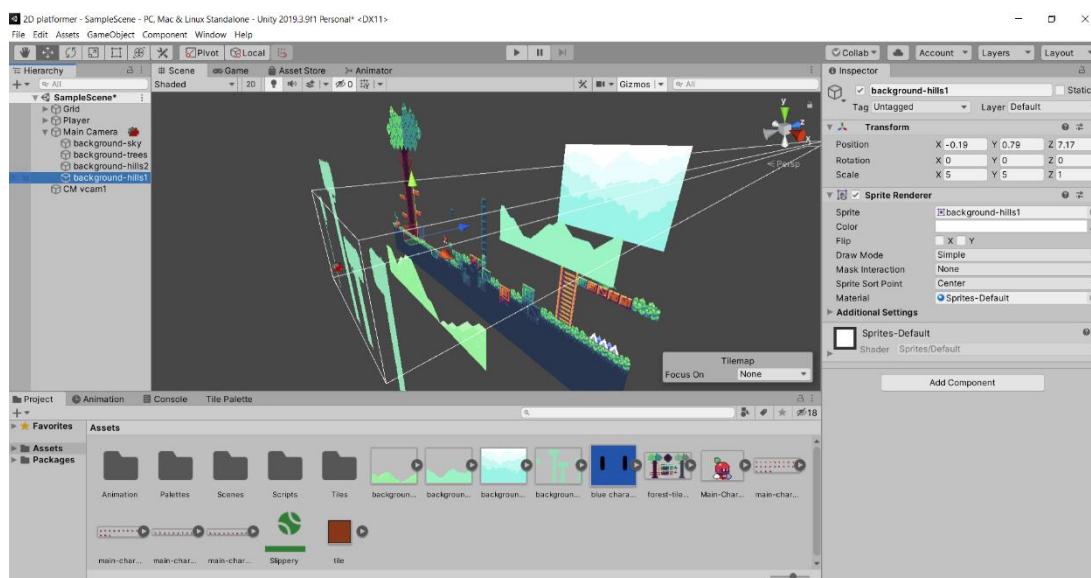
Hier geef je een waarde aan de float in de animator genaamd Speed. de 2e waarde tussen de () geeft aan wat er op speed wordt gezet. Dat betekent dus dat de parameter Speed dezelfde waarde krijgt als horizontalMove. De animator weet nu wanneer het personage beweegt. Mathf.Abs zorgt ervoor dat de snelheid geen - waarde krijgt, omdat naar links lopen een negatieve waarde geeft krijgt de animator hier problemen mee, nu zal de waarde dus altijd positief zijn. Zo zet ik voor elke parameter een waarde, omdat jump true of false is gebruik ik hier animator.SetBool

Volgende Camera

Om ervoor te zorgen dat het personage niet uit het zicht verdwijnt moet je ervoor zorgen dat de camera het personage volgt. Dit kan heel makkelijk door de Main Camera op de speler te slepen en de x,y coördinaten op 0 te zetten. Echter zal de camera niet zo natuurlijk aanvoelen als in de meeste games omdat hij heel blokkerig en direct volgt. Eerst moest je al deze instellingen coderen maar nu heeft Unity daar een package voor die je kunt installeren genaamd CineMachine. Hiermee kun je een CineMachine Camera toevoegen en veel variabelen veranderen. Je kunt bijvoorbeeld de camera laten bewegen wanneer de speler buiten een bepaald gebied gaat of de camera alvast bewegen naar de plek waar de speler naartoe loopt. Ik ga met deze instellingen spelen totdat ik een camera krijg die voor mij goed en natuurlijk aanvoelt. Daarnaast maak ik een aantal opties die ik uiteindelijk laat testen. Door deze feedback en usertesting weet ik uiteindelijk wat goede instellingen zijn voor de Player Camera.

Parallax Achtergrond Toevoegen

Om toch een gevoel van diepte te krijgen verdeel je de achtergrond op in stukken. Deze zet je vast aan de camera maar wel op een ander punt op de Z-as. Als je met een script de voorwerpen die verder weg staan langzamer laat bewegen, zal het eruit zien alsof deze ook verder weg staan. Net als in het echt lijkt het alsof voorwerpen die dichterbij staan sneller verplaatsen. Dit geeft een parallax effect en geeft een gevoel van diepte.



In de code moet je ervoor zorgen dat je een paar variabelen maakt voor de positie en lengte de achtergrond. Ook moet je een gameobject aanroepen en een public float voor het parallax effect aanmaken. Deze maak je public zodra je in Unity zelf nog kunt veranderen hoe sterk dit parallax effect wordt.

Je wilt de startpositie weten, deze kun je makkelijk verkrijgen door `transform.position.x` te gebruiken, deze waarde geef je mee aan de variabele `startpos`. Om achter de lengte van de achtergrond te komen moet je een component genaamd `SpriteRenderer` aanroepen en de x-size opvragen. Dit doe je door `GetComponent<SpriteRenderer>().bounds.size.x`, deze waarde zet je vervolgens in de variabele genaamd `lenght`.

Nu heb je bijna alle variabele de goede waarde gegeven. We geven alleen nog de float genaamd dist de goede waarde voor de distance die bewogen moet worden. Dit wordt gedaan met `float dist = (cam.transform.position.x * parallaxEffect);` Omdat je met `parallaxEffect` vermenigvuldigt zal dit dus meer worden als je een hogere `parallaxEffect` waarde meegeeft.

Nu kan je eindelijk de positie van de verschillende achtergrond lagen veranderen.

`transform.position = new Vector3(startpos + dist, transform.position.y, transform.position.z);`

Dit zorgt ervoor dat de positie van de vectors verandert. Door distance hierin te gebruiken wordt ook automatisch de waarde van `parallaxeffect` meegerekend. Nu hoeft je alleen nog per achtergrond laag een andere `parallaxeffect` waarde toe te voegen.

```
0 references
public class Parallax : MonoBehaviour {

    private float lenght, startpos;           //We hebben een paar variabelen nodig om de lengte en startpositie van de sprites te weten
    public GameObject cam;                   //Hier wordt een gameobject van de camera gemaakt
    public float parallaxEffect;             //Dit wordt de waarde van hoeveel parallax effect wordt toegepast, dit is public zodat je in Unity dalijk de waarde in kunt veranderen indien

}

// Start is called before the first frame update
0 references
void Start() {
    startpos = transform.position.x;         //Hier wordt de startpositie meegegeven door de eerste x waarde in startpos te zetten
    lenght = GetComponent<SpriteRenderer>().bounds.size.x; //Hier wordt een component uit unity opgeroepen genaamd de SpriteRenderer (deze component zit op veel objecten met sprites,
}

// Update is called once per frame
0 references
void FixedUpdate() {
    float dist = (cam.transform.position.x * parallaxEffect); //Om bij te houden hoeveel er bewogen wordt maak je een float variabele voor de distance. Door de x transform (dus hoeveel
    transform.position = new Vector3(startpos + dist, transform.position.y, transform.position.z); //Om daadwerkelijk te achtergrond en camera te bewegen moet je de positie van de vector
}
```

Als je wilt dat deze achtergrond automatisch afspeelt zonder dat de speler loopt én dat de achtergrond automatisch verdwijnt en opnieuw spawnt gebruik je onderstaande code. Dit heb ik gebruikt voor het startscherm, hier loopt de achtergrond dus oneindig door.

```
public float speed;
private float x;
public float PontoDeDestino;                //Het punt waar de achtergrond moet zijn om weer weggehaald te worden
public float PontoOriginal;                 //Het originele punt van de achtergrond

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

    x = transform.position.x;                //De positie van de achtergrond op de X as
    x += speed * Time.deltaTime;            //De snelheid waarmee de positie verandert, deze kun je zelf instelling in Unity
    transform.position = new Vector3 (x, transform.position.y, transform.position.z); //Hier word de positie verandert en wordt er rekening gehouden met de y en z as

    if (x <= PontoDeDestino){                //Als het object links bij het bestemmingspunt komt wordt de code uitgevoerd

        x = PontoOriginal;
        transform.position = new Vector3 (x, transform.position.y, transform.position.z); //Hier wordt de achtergrond weer naar de rechterkant van het scherm gezet voor een oneindige loop
    }
}
```

Talking NPC's

Ik wil in mijn game een ander personage hebben waarmee je kunt praten en die je handige tips geeft. Hiervoor moet ik een dialogue system maken. Hiervoor gebruik je Unity's canvas object met daarin wat tekst en een dialogue box. Ik heb voor de simpele dialogue box hieronder gekozen.



Hierin zet je een tekst object voor de NPC naam en de tekst die hij gaat zeggen. Nu je dit klaar hebt is het tijd om te coderen en ervoor te zorgen dat deze tekst elementen worden aangepast. Er kwam veel codeerwerk aan te pas en ik heb hier lang op vastgezeten. Ik leg hier kort uit wat ik heb moeten doen maar om een goed beeld te krijgen van de code en hoe deze werkt kun je beter in mijn Gitlab kijken. De comments bij de code leggen alles goed uit.

Zie: <https://git.fhict.nl/I437759/semester-3-media/-/tree/master/UnityScripts>

Als eerste maak ik een C# Script voor de DialogueManager. Hierin maak ik een queue variabele aan waar alle zinnen in worden bewaart die gezegd moeten worden. Omdat dit tekst is zeg ik erbij dat het type een string is.

```
private Queue<string> sentences; // Dit is de variabele die alle zinnen gaat bijhouden die gedisplated moeten worden. Dit is tekst dus een string, een queue zorgt
```

Hiernaast gaan we tegelijkertijd in een ander C#Script werken die alle informatie gaat bijhouden zoals de zinnen en de naam van de NPC's. Deze noem ik Dialogue. Ook moet er een script komen die ervoor zorgt dat de tekst geactiveerd wordt, door dit in een apart C# script te zetten kun je deze makkelijk op objecten slepen zoals een button of een npc. Dit script heet DialogueTrigger.

DialogueTrigger is een erg klein script die er alleen voor zorgt dat alle informatie uit de andere scripts wordt gehaald en wordt getriggerd. (Afbeelding hieronder)

```
0 references
public class DialogueTrigger : MonoBehaviour {

    public Dialogue dialogue; //Je roept hiermee de variabele op uit de Dialogue Script waardoor je de Name en Sentences in Unity kunt bewerken.

    0 references
    public void TriggerDialogue ()
    {
        FindObjectOfType<DialogueManager>().StartDialogue(dialogue); //Om de informatie die ingevoerd is door te geven aan de DialogueManager moet je deze eerst vinden met FindObjectOfTyp
    }
}
```

Door het DialogueTrigger script kan ik zelf zinnen en een naam in unity meegeven aan een knop of NPC. Deze worden in de DialogueManager in de Queue gezet en vervolgens worden de zinnen één voor één weergegeven. Hierdoor kan ik enorm makkelijk veel verschillende NPC's maken. Ik kan dan in Unity een naam meegeven, hoeveel zinnen ze zeggen en wat er in die zinnen komt te staan. Hierdoor bespaar ik uiteindelijk tijd op het einde van mijn game.

```
[System.Serializable] //Hiermee komen de variabele in de inspector waardoor je deze kunt aanpassen in Unity zelf.
2 references
public class Dialogue { //In dit script komt alle informatie van de dialogue's. Die worden doorgestuurd naar de DialogueManager script.
    public string name; //Je kunt ook de naam van de npc meegeven die praat zodat dit ook meegegeven wordt aan de DialogueManager

    [TextArea(3, 10)] //Hiermee geef je aan wat de minimale en maximale aantal lijnen zijn die de text area boxes worden.
    public string[] sentences; //Dit wordt een string array met alle zinnen die we naar de DialogueManager willen sturen
}
```

De DialogueManager is iets groter, hierin worden de belangrijke variabelen gemaakt en de UI elementen in Unity aangeroepen zodat deze aanpasbaar zijn. De dialogue tekst kan bijvoorbeeld aangepast worden naar de tekst die ik zelf heb opgegeven.

```
using UnityEngine;
using UnityEngine.UI; //Hiermee kunnen de UI elements aangepast worden in Unity

1 reference
public class DialogueManager : MonoBehaviour {

    public Text nameText; // De waarde in deze variabele wordt weergegeven als naam in de dialogue box
    public Text dialogueText; // De waarde in deze variabele wordt weergegeven als text in de dialogue box

    public Animator animator; // Hierdoor kun je de animator besturen, deze moet je wel in Unity eerst hierop slepen

    private Queue<string> sentences; // Dit is de variabele die alle zinnen gaat bijhouden die gedisplayed moeten worden. Dit is tekst dus een string, een queue zorgt ervoor dat steeds de v

    // Start is called before the first frame update
    0 references
    void Start() {
        sentences = new Queue<string>();
    }
}
```

De functie voor het starten van de dialogue zet eerst de parameter van de animator op true, hierdoor wordt de animatie van de dialoguebox_open geactiveerd en zal deze in het beeld verschijnen. Ik heb in de Animator al alle parameters en animaties goed gezet.

Vervolgens zorgt deze ervoor dat de nameText object in Unity verandert naar de naam waarde die is meegegeven en dat alle vorige sentences verwijderd worden. Vervolgens haalt hij de sentences uit de queue en speelt die deze één voor één af door de dialoguetext te veranderen.

```
1 reference
public void StartDialogue (Dialogue dialogue)
{
    //Debug.Log("Starting conversation with " + dialogue.name); // dit was even om te checken of de console de goede waardes weergaf

    animator.SetBool("IsOpen", true); // Hiermee zet je de parameter die ik in de animator heb aangemaakt op true, hierdoor kan de animatie van de dialogue box o

    nameText.text = dialogue.name; // De naam wordt in nameText weergegeven, dit is het UI element in de game en de naam wordt dus in de game weergegeven

    sentences.Clear(); //Om meerdere sentences goed weer te geven wil je eerst de sentences van een vorig gesprek verwijderen door sentences.clea

    foreach (string sentence in dialogue.sentences) //Je gaat door iedere sentence in de dialogue sentences array en queueed de volgende sentence
    {
        sentences.Enqueue(sentence); //Je zet hiermee dus alle sentences in de queue
    }

    DisplayNextSentence(); //Nanneer alle sentences zijn gequeueed wil je de volgende sentence oproepen met deze functie
}
```


Wel moet er gecheckt worden of er nog een zin in de queue is, daar kan ik makkelijk

```
if (sentences.Count == 0) {  
    EndDialogue();  
    return; }  
}
```

Als er dus geen zinnen meer zijn zal de dialogue gestopt worden door EndDialogue(); en de dialoguebox weggehaald worden. Als er wel nog zinnen zijn speelt hij de volgende zin af.

```
1 reference  
public void DisplayNextSentence() //Dit is de functie waarmee de volgende sentence wordt opgeroepen  
{  
    if (sentences.Count == 0) //Je checkt eerst of er wel nog sentences zijn in de queue, is dit niet het geval (dus .count == 0) dan eindigt de dialogue  
    {  
        EndDialogue(); //EndDialogue is nog een aparte functie die hier opgeroepen wordt  
        return;  
    }  
  
    string sentence = sentences.Dequeue(); //Is dit wel het geval, komt de volgende sentence in de sentence string  
    // Debug.Log(sentence); dit was om te testen of alles het deed  
    //dialogueText.text = sentence; //De volgende sentence wordt in dialogueText gestopt, en wordt vervolgens weergegeven in de game door het UI element genaamd  
    StopAllCoroutines(); //Je wilt niet dat er meerdere zinnen door elkaar komen als iemand snel door de tekst heen skipt dus dit zorgt ervoor dat es  
    StartCoroutine(TypeSentence(sentence)); //Je start hier de coroutine die ervoor zorgt dat alles letter voor letter weergegeven wordt met de aangegeven snelheid acht  
}
```

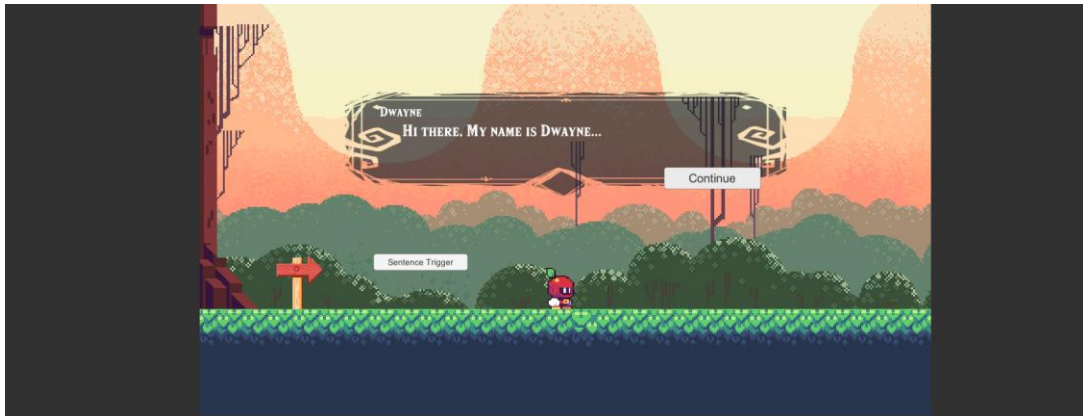
Wel moet ik nog aangeven wat de functie EndDialogue(); precies doet. Hieronder zie je dat ik ervoor zorg dat een parameter in de animator op false wordt gezet. Dit zorgt ervoor dat de animatie_close getriggerd wordt. Hierdoor zal de dialoguebox uit het beeld verdwijnen.

```
1 reference  
void EndDialogue() //Als er geen sentences meer in de queue zijn laat de log "End of conversation zien"  
{  
    //Debug.Log("End of conversation");  
    animator.SetBool("IsOpen", false); // Hiermee zet je de parameter die ik in de animator heb aangemaakt op false, hierdoor kan de animatie van de dialogue box  
}
```

Het geeft een leuk effect om elke letter één voor één weer te geven. Door foreach te gebruiken ga je door elke character heen en stop je die in een character array. Daarna wordt iedere letter in die array omstebeurt toegevoegd. Dit gebeurt eigenlijk elke frame (als je yield return null; gebruikt) maar op goede laptops die makkelijk 60fps halen zal dit effect te snel gaan om te zien. Daarom gebruik ik yield return new WaitForSeconds(0.02f) zodat er een paar milliseconde tussen elke letter zit.

```
1 reference  
IEnumerator TypeSentence (string sentence)  
{  
    dialogueText.text = "";  
    foreach (char letter in sentence.ToCharArray()) //hiermee loop je door iedere character in je sentences en zet deze in een character array  
    {  
        dialogueText.text += letter; //de letter waarde in letter wordt nu iedere keer erbij gestopt waardoor je een letter voor letter animatie krijgt  
        // yield return null; //Hierdoor wordt er iedere frame een nieuwe letter bij gedaan, dit kan té snel gaan als je game op 60fps runt waardoor je er nie  
        yield return new WaitForSeconds(0.02f); //Daarom heb ik hier een delay op gezet van 0.02 seconde zodat je het letter effect beter zult zien op elke computer  
    }  
}
```

Nu de code werkte kon ik nog een font toevoegen, uiteindelijk zag alles er zo uit (afbeelding hieronder). Nu moest ik alleen nog het trigger script op een ander character slepen en ervoor zorgen dat de dialogue getriggerd werd door op E te klikken en niet door een mouseclick.



Om ervoor te zorgen dat ik geen button hoeft in te klikken maar alleen E, kon ik `Input.GetKeyDown(KeyCode.E)` gebruiken. Echter kon dit nu overal in het scherm. Ik wou dat dit alleen kon zodra mijn character in de buurt van een NPC stond. Hiervoor moest ik `distance = Vector2.Distance(player.transform.position, gameObject.transform.position);` gebruiken en in de `if` statement als tweede eis invoeren dat de distance minder dan of gelijk aan 1.5 moest zijn voordat de dialogue box kon worden geopend.

```
0 references
private void Update()
{
    Debug.Log(distance);
    distance = Vector2.Distance(player.transform.position, gameObject.transform.position);
    if (Input.GetKeyDown(KeyCode.E) && distance <= 1.5f /*playerIsNear == true*/) //Als er op de E gedrukt wordt zal de functie TriggerDialogue worden uitgevoerd.
    {
        TriggerDialogue();
    }
}

//private void OnTriggerEnter2D(Collider2D collision)
//{
//    if (collision.gameObject.CompareTag("Player"))
//    {
//        playerIsNear = true;
//        //animatie parameter true
//    }
//}
```

Nu kan ik voor elke NPC een eigen tekst bedenken en kan de speler pas met hem praten zodra hij in de buurt staat en op E drukt.



Na nog een kleine suggestie van Bernd-Jan heb ik een kleine sound effect toegevoegd wanneer een NPC praat. Hiervoor moest ik een audio file toevoegen, deze aanroepen en vervolgens starten en stoppen in de IEnumerator. Nu speelt er een type geluidje af wanneer de tekst verschijnt.

```
IEnumerator TypeSentence (string sentence)
{
    TypeSound.Play(); //Dit speelt het type sound effect af
    dialogueText.text = "";
    foreach (char letter in sentence.ToCharArray()) //hiermee loop je door iedere character in je sentences en zet deze in een character array
    {
        dialogueText.text += letter; //de letter waarde in letter wordt nu iedere keer erbij gestopt waardoor je een letter voor letter animatie krijgt
        // yield return null; //Hierdoor wordt er iedere frame een nieuwe letter bij gedaan, dit kan té snel gaan als je game op 60fps runt waardoor je er ni
        yield return new WaitForSeconds(0.02f); //Daarom heb ik hier een delay op gezet van 0.02 seconde zodat je het letter effect beter zult zien op elke computer
    }
    TypeSound.Stop(); //Stopt het type sound effect na een zin
}
```

Switching Scenes/Titel Scherm

In de meeste games begin je niet meteen in een level. Je hebt vaak een startscherm. Ik ben erachter gekomen dat je in Unity in verschillende scenes kunt werken. Zo heb ik een scene gemaakt voor het startscherm. In dit startscherm heb ik een knop gezet en een titel. Door op de knop te drukken wordt de volgende scene ingeladen. Ik heb hier een klein scriptje voor moeten schrijven.

```
using UnityEngine.SceneManagement; //Dit moet je gebruiken voordat je scenes kunt oproepen

public class MenuManager : MonoBehaviour
{
    1 reference
    public void startGame()
    {
        SceneManager.LoadScene("TutorialLevel"); //De Scene genaamd TutorialLevel wordt ingeladen.
    }

    0 references
    private void Update()
    {
        if (Input.GetKeyDown(KeyCode.Space)) //Als er op de spatiebalk gedrukt wordt zal de functie startgame worden uitgevoerd. Deze laadt het tutoriallevel in.
        {
            startGame();
        }
    }
}
```

Je ziet een functie die ervoor zorgt dat de scene genaamd TutorialLevel wordt geladen. Daaronder heb ik ervoor gezorgd dat die functie geactiveerd wordt zodra je op spatie klikt. Je hoeft nu dus niet op de knop te drukken maar kan ook op spatie klikken om het level te starten.



Ik heb advies gevraagd over de kleur, titel en achtergrond. Als ik er maar voor zorgde dat alles duidelijk te lezen is en de achtergrond iets met de game zelf te maken heeft zat ik wel goed. Ik heb ervoor gekozen om een geanimeerde achtergrond te maken, deze achtergrond is dezelfde als het beginlevel maar dan geanimeerd, de code hiervoor staat bij Parallax Background. Hieronder zie je het resultaat. (Press Space To Start is ook geanimeerd)



Spikes /Respawn

In veel games heb je obstakels die je niet aan mag raken. Ik wil spikes in mijn game hebben, als de speler deze aanraakt gaat hij dood en respawnt hij op de laatste save point. Hiervoor moet ik dus 3 mechanics toevoegen, het opslaan van een save point, het doodgaan en het opnieuw respawnen bij het laatst opgeslagen save point. Gelukkig had ik net geleerd hoe ik een scene moet inladen. Ik zorg er dus voor dat een animatie afspeelt en aan het einde van de Death_animatie voeg ik een event toe die ervoor zorgt dat de scene opnieuw wordt geladen.

```
[using UnityEngine.SceneManagement; //Dit moet je gebruiken voordat je scenes kunt oproepen]

0 references
public class Spikes : MonoBehaviour {

    public Animator animator; // Hierdoor kun je de animator aanroepen door animator de typen (sleep de animator eerst nog op de player component)

    0 references
    private void OnTriggerEnter2D(Collider2D collision)
    {
        if (collision.gameObject.CompareTag("Player")) //Wanneer een collision met de speler wordt gedetecteerd word dit uitgevoerd. Dit script staat op de spikes dus wanneer de speler de sp
        {
            animator.SetBool("IsDeath", true);
        }
    }

    0 references
    void Die()
    {
        SceneManager.LoadScene("TutorialLevel"); //De Scene genaamd TutorialLevel wordt ingeladen.
    }
}
```

Dat event is de void Die(); functie, deze laad gewoon opnieuw de scene in nadat de dood animatie zich afspeelt. Deze Die(); functie wordt geactiveerd door de OnTriggerEnter2D(Collider2d collision) die checkt of de colliders van de player en de spikes in aanraking komen met elkaar. Zodra dit gebeurd speelt de death animatie zich af en wordt de scene opnieuw ingeladen.

Camera Transitions

Ik wou graag de camera's in elkaar laten overvloeien zodat ik verschillende shots in een level kan hebben. Hiervoor heb ik een tweede camera aangemaakt en deze een hogere prioriteit gegeven. Wel heb ik deze uitgeschakeld zodat ik deze in het script kan inschakelen wanneer ik een bepaalde collider raak in het level. Zo wordt er op bepaalde plekken in het level uitgezoomd en weer ingezoomd.

```
0 references
public class Camera : MonoBehaviour
{
    public GameObject vmCamTwo; // Een referentie naar de camera
    public bool beginning;

    0 references
    private void OnTriggerEnter2D(Collider2D col) // als een object de trigger in komt wordt dit uitgevoerd
    {
        if (col.gameObject.CompareTag("Player")) // check of het de speler is, zo ja voert die de code uit
        {
            // schakel de nieuwe camera (met hogere prioriteit) in/uit
            vmCamTwo.SetActive(beginning);
        }
    }
}
```

Attacking + Enemies

Ik wil dat mijn character kan aanvallen en damage doet tegen een enemy. Hiervoor maak ik een aantal scripts aan. Eerst zorg ik ervoor dat de attack animatie werkt door onderstaande code. Dit is niet veel anders dan de andere animaties.

```
void Update()
{
    if (Input.GetKeyDown(KeyCode.Space)) //Als deze knop wordt ingedrukt wordt de volgende code uitgevoerd
    {
        Attack(); //Zorgt dat de attack functie wordt uitgevoerd
    }
}

1 reference
void Attack()
{
    animator.SetTrigger("Attack"); //Zorgt ervoor dat de attack animatie getriggerd word
}
```

Ook maak ik wat variabele aan voor de levens van de vijand en zorg ik dat ook alle animaties van de vijand kloppen. Ook krijgt de vijand damage wanneer de speler zijn attack in de buurt van de vijand doet. Deze damage gaat van zijn currenHealth af.

```
public int maxHealth = 100; //Hier worden de variabele voor de enemy's levens en huidige levens gemaakt
int currentHealth;
public Animator animator;

// Start is called before the first frame update
0 references
void Start()
{
    currentHealth = maxHealth;
}

1 reference
public void TakeDamage(int damage)
{
    //Damage krijgen
    currentHealth -= damage;

    //Animatie triggeren
    animator.SetTrigger("Hurt"); //Zorgt ervoor dat de hurt animatie wordt afgespeeld wanneer de enemy damage krijgt

    //Doodgaan + animatie
    if(currentHealth <= 0) //Als de enemy's healt lager is dan 0 gaat deze dood en wordt de EnemyDie functie uitgevoerd
    {
        EnemyDie();
    }
}
```

De vijand moet ook doodgaan als zijn levens onder 0 komen. Wanneer dit gebeurt moeten ook sommige components van de vijand gedisable worden zodat je niet tegen een doorzichtige collider aanloopt of er een animatie blijft spelen. Dit wordt hieronder gedaan.

```
1 reference
void EnemyDie()
{
    Debug.Log("Enemy died");
    //Die animation
    animator.SetBool("IsDead", true); //Zet de parameter genaamd IsDead in de animator op true waardoor de IsDead animatie wordt afgespeeld

    //Disable Enemy
    GetComponent<Collider2D>().enabled = false; //Hiermee wordt de box collider uitgeschakelt zodat je over de dode enemy heen kunt lopen
    this.enabled = false; //Hiermee wordt het script op de enemy gedisable zodat je er verder geen last meer van hebt
}
```

Ik update het script van de player combat zodat deze ook damage doet en checkt of er een vijand in de buurt van de attackpoint is. Ook gebruik ik Gizmos.DrawWireSphere om een sphere vanuit de positie van de attackpoint te tekenen met de grootte van attackRange. Hierdoor kan ik in de inspector kijken hoe groot de attackRange precies is en dit aanpassen.

```
1 reference
void Attack()
{
    //Animatie afspelen
    animator.SetTrigger("Attack"); //Zorgt ervoor dat de attack animatie getriggerd word

    //Geluid afspelen
    //PunchSound.Play(); //Dit speelt het type sound effect af

    //Checken of er een enemy in range is
    Collider2D[] hitEnemies = Physics2D.OverlapCircleAll(attackPoint.position, attackRange, enemyLayers); //Het maakt een circle (met range van attackrange) vanuit de attackpoint en e

    //Damage doen
    foreach(Collider2D enemy in hitEnemies)
    {
        //Debug.Log("We hit the enemy");
        enemy.GetComponent<Enemy>().TakeDamage(attackDamage);
    }

0 references
private void OnDrawGizmosSelected() //Dit is alleen om even in de editor te kunnen zien wat de range van de attackpoint precies is, hierdoor is het afstellen van deze range wat makkelijk
{
    if (attackPoint == null)
        return;
    Gizmos.DrawWireSphere(attackPoint.position, attackRange); //Hiermee wordt er een sphere getekent vanuit de positie van attackpoint met de range van attackrange, dit voegt du
}
}
```

Nu kan de speler nog enorm snel slaan, dit is een beetje te sterk. Daarom zorg ik er met onderstaande code voor dat de player niet zo snel kan slaan en eerst moet wachten tot er een bepaalde tijd voorbij is na een slag. Ook kan ik een punch sound toevoegen elke keer als de speler slaat maar dit heb ik er uiteindelijk uit gelaten omdat ik het geen fijn geluidje vond.

```
0 references
private void Start()
{
    //PunchSound = GetComponent<AudioSource>(); // Haalt de audiosource die ik in Unity hierop heb gesleept
}

// Update is called once per frame
0 references
void Update()
{
    if(Time.time >= nextAttackTime) //Checkt of je al kunt aanvallen
    {
        if (Input.GetKeyDown(KeyCode.Mouse0)) //Als deze knop wordt ingedrukt wordt de volgende code uitgevoerd
        {
            Attack(); //Zorgt dat de attack functie wordt uitgevoerd
            nextAttackTime = Time.time + 1f / attackRate; //je voegt tijd toe bovenop je huidige tijd en dat wordt de tijd waarna je weer kunt aanvallen (wordt bij Time.time >= nextA
        }
    }
}
```

Muziek + Don't Destroy

Ik wil nog wat achtergrond muziek toevoegen. Hiervoor moet ik een audiosource toevoegen aan de scene, maar omdat er een nieuwe scene in wordt geladen met nieuwe objecten wordt de muziek opnieuw afgespeeld of helemaal niet. Je moet ervoor zorgen dat hetzelfde object uit de eerste scene mee wordt genomen naar de tweede scene. Het moet dus niet vernietigd worden zoals de rest. Hiervoor heb ik het kleine scriptje hieronder geschreven. Deze vernietigt wel alle andere objecten met de Music tag voordat hij hem meeneemt omdat je anders dubbele muziek hoort als je terug zou gaan naar het startscherm.

```
public class DontDestroy : MonoBehaviour
{
    // references
    private void Awake()
    {
        GameObject[] objs = GameObject.FindGameObjectsWithTag("Music"); //Pakt de gameobjects met de music tag en zet die in de array
        if(objs.Length > 1) //als die er meer dan 1 vind vernietigd die de andere muziekjes (dit is alleen als je terug zou gaan naar de scene waar de oorspronkelijke muziek was)
        {
            Destroy(this.gameObject);
        }
        DontDestroyOnLoad(this.gameObject); //Zorg ervoor dat dit gameobject blijft bestaan als je de nieuwe scene laadt, de muziek blijft zo doorgaan
    }
}
```

Feedback

Elke keer dat ik een nieuwe mechanic had toegevoegd, liet ik dit aan Bernd-Jan of een medestudent zien. Hierdoor kwam ik snel achter bugs of kreeg ik tips over hoe ik iets beter kon doen.

Bernd-Jan vertelde me dat als je wat meer tijd in de code stopt, je later veel tijd bespaart. Dit kan bijvoorbeeld door niet alle dialogue tekst in je code te zetten, maar dit via Unity zelf in je code te laten zetten. Hierdoor kan je makkelijk een NPC dupliceren en deze zijn eigen tekst geven. Zo kon je ook het bouwen van een level efficiënter maken.

Omdat ik weinig ervaring had met Unity en C# gaf Bernd-Jan mij wat voorbeeld projecten, hierin kon ik zien hoe structuur van Unity eruit zag en hoe sommige mechanics werkten. Ik heb hier enorm veel van geleerd en dit uiteindelijk toegepast in mijn project. Bernd-Jan was enorm tevreden met het eindresultaat en het proces.

Reflectie

Ik had amper ervaring met Unity voordat ik hieraan begon, wel had ik in semester een klein beetje met C# gewerkt maar hier was ik al veel van vergeten. Het was erg fijn om dit op te kunnen frissen voordat ik de specialisatie Game Design ga doen.

In het begin had ik veel code die wel werkte, maar niet echt efficiënt was. Eerst zette ik bijvoorbeeld de tekst die de NPC's moesten zeggen in de code zelf. Een tip van Bernd-Jan was om iets meer tijd in je code te stoppen en het efficiënt te maken zodat je later veel minder werk hebt. Laterna had ik de code zo aangepast dat je de tekst in Unity kon zetten bij de bijbehorende NPC. Dit gold ook voor de grond en spikes. Door hier aparte layers en tilegrids van te maken, hoefde ik niet steeds colliders op de grond te slepen met de juiste scripts. Ik kon uiteindelijk gewoon de laag selecteren, blokjes tekenen op het level en alles werkte meteen. Voor de volgende keer wil ik dit vaker gaan gebruiken zodat ik op het einde weinig tijd kwijt ben met level design.

Voor de volgende keer zou ik beter mijn objecten ordenen. Nu had ik veel losse objecten waardoor colliders en animaties soms niet goed samen werkten. Het was veel makkelijker geweest als ik alle objecten die samen horen in een parent object had gezet. Ook had dit me veel chaos en tijd bespaart op het einde van het proces.

Ik heb erg lang vastgezeten op de dialogue system. Achteraf had ik eerder hulp moeten vragen aan Tom. Wel vond ik het goed dat ik elke keer als ik iets nieuws had, dit meteen liet zien aan Bernd-Jan of een mede student om te testen. Zo kwam ik meteen achter de bugs en kon ik deze oplossen voordat ik aan iets anders begon.

Ook weet ik nu dat ik het erg leuk vind om mijn eigen ideeën in een game te kunnen verwerken, ik ben nu erg positief over mijn keuze om Game Design te gaan doen.