Inleiding tot "Unity 2D Arcade Project"

Unity Game Development voor Beginners met Basis Programmeerkennis voor Kandidaten bij Helden In IT.

Welkom bij Unity 2D Arcade Project! Deze cursus is speciaal ontworpen om jou vertrouwd te maken met de fundamentele principes van het ontwikkelen van games met hulp van de Unity-game-engine. We verwachten dat je bekend bent met basis programmeerconcepten, zoals loops en variabelen, hoewel je niet specifiek bekend hoeft te zijn met C#.

Inleiding tot "Unity 2D Arcade Project" 1

- Over Unity 3
- Voor wie is deze cursus bedoeld? 3
- Benodigdheden 3
- Hoe is de cursus opgezet? 3
- Waarom Unity: 3
- Leerdoelen: 4
- Vooraf wat begrippen: 4
  - Sprites: 4
  - Game Mechanics: 4
  - Unity UI (User Interface): 4
  - Placeholders: 4
  - Scènes: 4
  - Backend: 4
  - Prefabs: 4
  - PlayerPrefs: 4
  - Velocity: 4
  - Rigidbody: 4
  - Collider: 5
  - Quaternion.identity: 5
  - Public en Private in Unity C#: 5
- Basic Movement in Unity Editor: 5
- Game Concept: 5
- Hoofdstuk 1: Unity, Github en Visual Code 6
- Quick Start 6
- Hoofdstuk 2: De Speler 6

Stap 1: Creëer de Speler 6
Stap 2: Creëer de Grond/Plafond 7
Stap 3: Begin met het Script 8
Hoofdstuk 3: De Obstakels 9
Stap 1: Creëer de Prefab 9
Stap 2: maak het 2de script. 10
Stap 3: maak een manager die de objecten aanmaakt. 11
Hoofdstuk 4: Punten Systeem 11
Stap 1: de punten 11
Stap 2: de High Scores 12
Hoofdstuk 5 UI: 13
Stap 1 Reset Knop: 13
Stap 2 HighScores: 14
Stap 3 Game pauzeren: 16
Hoofdstuk 6 De Build: 16
Project Notes 15
Project "Could-Have" 16

Over Unity

Unity is een van de meest populaire en krachtige game-engines in de industrie. Het biedt een uitgebreide set tools en functies die het mogelijk maken om games te ontwikkelen voor verschillende platforms, zoals pc, mobiel, consoles en meer. Met Unity kun je zowel 2D- als 3D-games maken en heeft het een grote community van ontwikkelaars die samenwerken en kennis delen.
Wat ga je leren? In deze cursus zullen we ons richten op het ontwikkelen van een 2D arcadegame met Unity. De nadruk ligt op het bouwen van leuke, Interactieve spelervaringen zonder de complexiteit van het maken van een volledige spelwereld.

We zullen onder andere behandelen:

- Het opzetten van een nieuw 2D project en de Interface van Unity verkennen.
- Werken met 2D-assets, waaronder Sprites en Prefab 's.
- Het schrijven van code om functionaliteit aan je arcadespel toe te voegen (basis programmeerkennis vereist).

Voor wie is deze cursus bedoeld?

Deze cursus is geschikt voor beginners in Unity die al enige basis programmeerkennis hebben en specifiek gericht is op kandidaten bij Helden In IT. Als je bijvoorbeeld bekend bent met concepten zoals loops, variabelen en conditionele statements, ben je goed op weg! Je hoeft niet per se bekend te zijn met C#, aan de hand van deze cursus leer je de basis.

Benodigdheden

Om deze cursus te volgen, heb je het volgende nodig:

- Een computer met toegang tot Internet.
- Een installatie van Unity (we zullen laten zien hoe je dit installeert in de eerste les).
- Enthousiasme en een verlangen om te leren!

Hoe is de cursus opgezet?

De cursus is opgedeeld in verschillende modules die elk een aspect van de game zal behandelen. Elke module bevat praktische opdrachten om je te helpen de geleerde concepten toe te passen.

Waarom Unity:

We hebben gekozen voor Unity vanwege verschillende voordelen. Allereerst is het een zeer populaire cross-platform engine, wat betekent dat Unity kan worden gebruikt op verschillende consoles en besturingssystemen. Dit biedt veel flexibiliteit in de ontwikkeling van onze game. Daarnaast heeft Unity een gebruiksvriendelijke Interface en maakt het gebruik van de duidelijke programmeertaal C#. Dit maakt het ideaal voor beginners, zeker omdat het gratis is voor non-profit gebruik. Dit maakt het leren van Unity toegankelijk voor een breder publiek.

Een ander sterk punt van Unity is de grote community eromheen. Met een overvloed aan forums en andere bronnen is er altijd veel ondersteuning beschikbaar om te helpen bij eventuele uitdagingen die we tegenkomen tijdens de ontwikkeling van onze game.

Leerdoelen:

- Aan het einde van deze cursus zul je in staat zijn om een 2D game in Unity te creëren met basisgame mechanics, zoals speler beweging en Interactie met objecten
- Je zult bekwaam worden in het gebruik van Unity's UI-systeem om Intuïtieve gebruikersinterfaces te ontwerpen en implementeren voor een vloeiende game-ervaring
- Je zult leren hoe je code schrijft in C# om game-logica te implementeren, zoals scoreberekening, levensbeheer en overgangen tussen scènes.
- Na afloop van de cursus zul je in staat zijn om PlayerPrefs te gebruiken voor het beheren van gamegegevens, waardoor je een efficiënte manier hebt om gegevens op te slaan.
- Aan het einde van deze cursus zul je in staat zijn om je kennis en vaardigheden toe te passen op een tweede gameproject, waarbij informatie wordt overgenomen en verbeterd.

Vooraf wat begrippen:

Sprites: Grafische afbeeldingen die worden gebruikt om GameObjecten vorm te geven in een 2D-spel. Dit kunnen personages, voorwerpen, achtergronden, enzovoort zijn.

Game Mechanics: De regels en Interacties die bepalen hoe het spel werkt. Dit omvat bijvoorbeeld beweging, Interactie met objecten, en het afhandelen van speler invoer

Unity UI (User Interface): Het systeem binnen Unity waarmee je knoppen, menu's en andere elementen kunt maken en beheren om een gebruikersvriendelijke ervaring te creëren.

Placeholders: Tijdelijke grafische of geluidsbestanden die worden gebruikt om de functionaliteit van een game te testen voordat de definitieve kunst en geluidseffecten zijn gemaakt.

Scènes: In Unity, de verschillende omgevingen of niveaus van een game. Elke scène kan verschillende GameObjecten, achtergronden en logica bevatten.

Backend: Het deel van een game dat verantwoordelijk is voor de logica en functionaliteit die niet direct zichtbaar is voor de speler. Dit omvat zaken als game mechanics, scoreberekening, en gegevensbeheer.

Prefabs: Prefabs zijn GameObjecten die je vooraf maakt en kunt hergebruiken in je project. Ze bevatten vaak visuele, gedrags- en andere componenten die je hebt ingesteld en geconfigureerd. Prefabs maken het gemakkelijk om dezelfde elementen in verschillende scènes of op verschillende locaties in je spel te gebruiken.

PlayerPrefs: PlayerPrefs in Unity laat je eenvoudig gegevens bewaren tussen speelsessies. Het is handig voor kleine stukjes informatie zoals instellingen, gamevoortgang en keuzes van de speler. PlayerPrefs.SetInt(naam, Int). Naam is de naam waarin je het opslaat en Int is de waarde die je op de naam wilt opslaan.

Velocity: Verwijst naar de snelheid van een GameObject. Stel je voor dat je een gamepersonage hebt, de snelheid waarmee het beweegt en de richting waarin je beweegt, samen bepalen de Velocity

Rigidbody:

In Unity verwijst een **Rigidbody** naar een component die fysische eigenschappen aan GameObjecten geeft. Met een **Rigidbody** kunnen objecten reageren op zwaartekracht, krachten en impulsen. Dit stelt

ontwikkelaars in staat om realistische bewegingen en Interacties in hun games te simuleren. Het gebruik van een **Rigidbody** is essentieel voor objecten die fysische reacties moeten vertonen, zoals personages, ballen of voertuigen.

Collider:

Een **Collider** in Unity is een component die wordt gebruikt om de vorm van een GameObject te definiëren, waardoor het reageert op fysische Interacties zoals botsingen. **Collider**-componenten creëren de grenzen van een object en worden gebruikt door het fysicasysteem van Unity om te bepalen wanneer objecten elkaar raken. Er zijn verschillende soorten **Collider**-componenten, zoals **BoxCollider**, **SphereCollider**, en **MeshCollider**, die elk geschikt zijn voor verschillende vormen en situaties.

Het combineren van een **Rigidbody** en een **Collider** maakt een GameObject fysisch Interactief, waardoor het realistisch kan bewegen en reageren op de omgeving. Dit duo is van cruciaal belang voor het implementeren van overtuigende fysica in Unity-games.

Quaternion.identity:

Quaternion wordt in Unity gebruikt om de rotatie van een gameobject aan te geven. En de .identity wordt gebruikt om aan te geven dat er geen extra rotatie is maar dat de rotatie hetzelfde blijft.

Public en Private in Unity C#:

In C# geven Private en Public toegangsniveaus aan hoe andere delen van je programma toegang hebben tot elementen zoals variabelen. Toegangsniveaus regelen hoe de elementen van een klasse (bijvoorbeeld variabelen, methoden) toegankelijk zijn vanuit andere delen van het programma. Een Private element is alleen toegankelijk binnen dezelfde klasse, terwijl een Public element toegankelijk is vanuit elk deel van het programma.

In Unity wordt [SerializeField] gebruikt om een privévariabele zichtbaar en aanpasbaar te maken in de Unity Editor, zonder deze openbaar te maken. Dit is handig om “Encapsulatie” (of inkapseling) te behouden terwijl je nog steeds de variabelen kunt inspecteren en aanpassen in de editor.

Basic Movement in Unity Editor:

Q: is de View tool.

W: is om GameObjects te verplaatsen.

E: is om GameObjects te roteren.

R: is om dimensies van een GameObject te schalen.

T: is om de dimensies aan te passen.

Y: is de Multitool met alles.

Game Concept:

In deze cursus gaan we een spel maken, vergelijkbaar met Flappy Bird. Het doel is om als een vogeltje tussen pijpen door te vliegen en punten te verdienen. We beginnen met het implementeren van de

beweging van het vogeltje en voegen zwaartekracht toe voor realistische beweging. Vervolgens richten we ons op het instellen van de beweging van de pijpen, zodat ze naar links schuiven en steeds opnieuw verschijnen.

Om een oneindige speelwereld te creëren, implementeren we een recycling-systeem voor de pijpen. We leren hoe we botsingen kunnen detecteren wanneer het vogeltje tegen een pijp botst, en we voegen een highscoresysteem toe zodat spelers hun prestaties kunnen meten.

Daarnaast voegen we een leeg GameObject toe dat een belangrijke rol speelt bij het maken van de pijpen en het handhaven van de juiste afstand tussen de pijpen. Dit GameObject wordt ook gebruikt om een zogenaamde 'Prefab' te creëren, een handige manier om game-elementen snel te dupliceren en te hergebruiken. Wanneer de speler succesvol tussen de pijpen door vliegt, zal dit GameObject ook worden gebruikt om punten toe te kennen.

Hoofdstuk 1: Unity, Github en Visual Code

In dit hoofdstuk zetten we ons gereedschap klaar om te gebruiken. We hebben een aantal accounts en applicaties nodig om de game te ontwikkelen.

Voor stap-voor-stap instructies zie: [Unity, Github & Visual Code]

Voor een Quick start, zie de instructie hieronder.

Quick Start

1. Maak een **Github Account**
2. **Installeer Github**
3. Maak een nieuwe repository aan op "*%USERPROFILE%/Github/Unity2D_ArcadeProject/*"

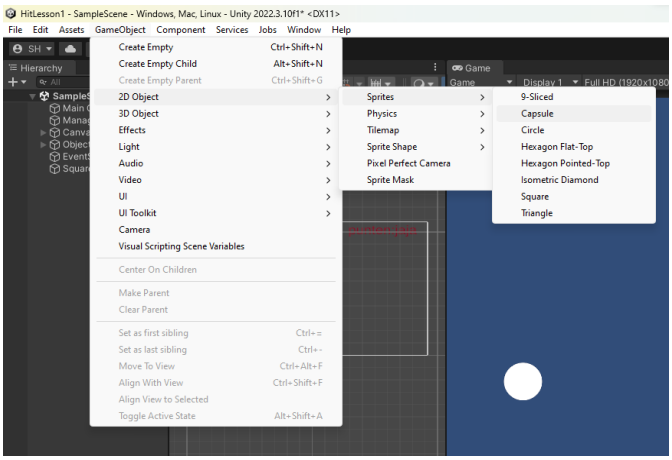
4. Maak een **Unity Account**
5. Installeer **Unity Hub** onder: "*C:/Development/Unity Hub/*"
6. Installeer de **Unity Editor** onder: "*C:/Development/Unity Editor/*"
7. Activeer een **persoonlijke licentie**
8. Creëer een **nieuw 2D project** op locatie: "*%USERPROFILE%/Github/Unity2D_ArcadeProject/*"

9. Installeer Visual Studio Code onder: "*C:/Development/Visual Code/*"
10. Installeer de **Unity Extensie** in VS Code
11. Installeer het **Visual Studio Editor** Pakket in Unity, onder Package Manager
12. Stel VS Code in als de **externe editor** van Unity

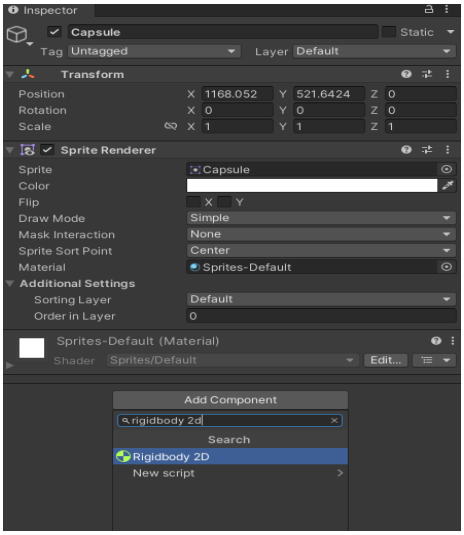
Hoofdstuk 2: De Speler

Stap 1: Creëer de Speler

- Maak een nieuw 2D GameObject door in de Unity-editor naar GameObject te gaan, dan 2D Object, Sprites, en kies Circle. Plaats hem dan aan de linkerkant van je scherm en verander de naam naar “Player”.



- Voeg een [Rigidbody2D] component toe, door op het PlayerObject te klikken komt er een inspector menu aan de rechterkant. Klik dan op Add Component en zoek/selecteer Rigidbody2D. Dit geeft ons PlayerObject de nodige Physics om te bewegen.

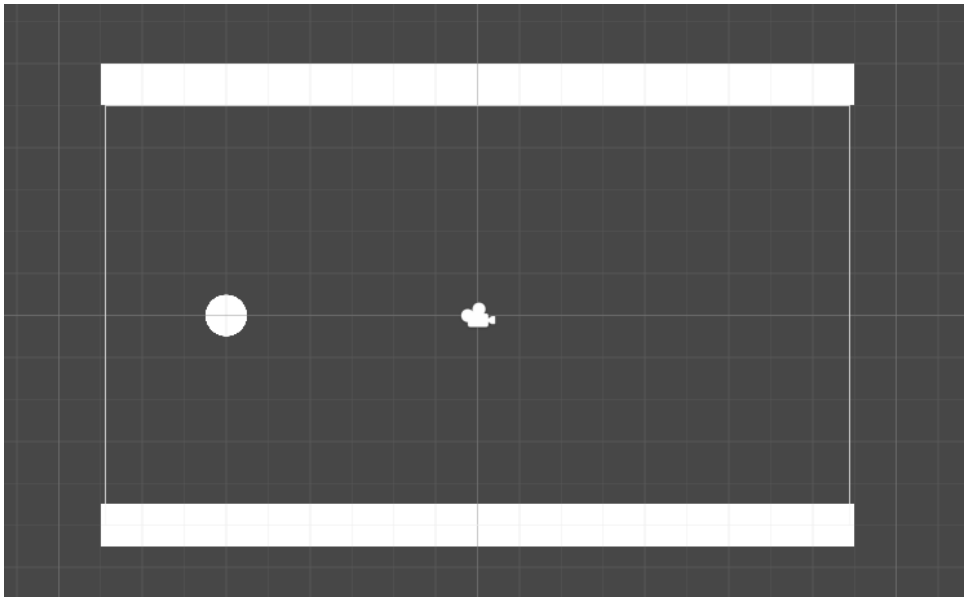


- Voeg een [CircleCollider2D] toe aan het PlayerObject, door weer naar Add Component te gaan en dan de CircleCollider2D. (Alles wat we toevoegen moet 2D zijn.)

Stap 2: Creëer de Grond/Plafond

- Maak nu nog een GameObject maar dan een kubus. Rek die uit en geef het een BoxCollider2D. Deze dient als de grond. En noemen we Ground.

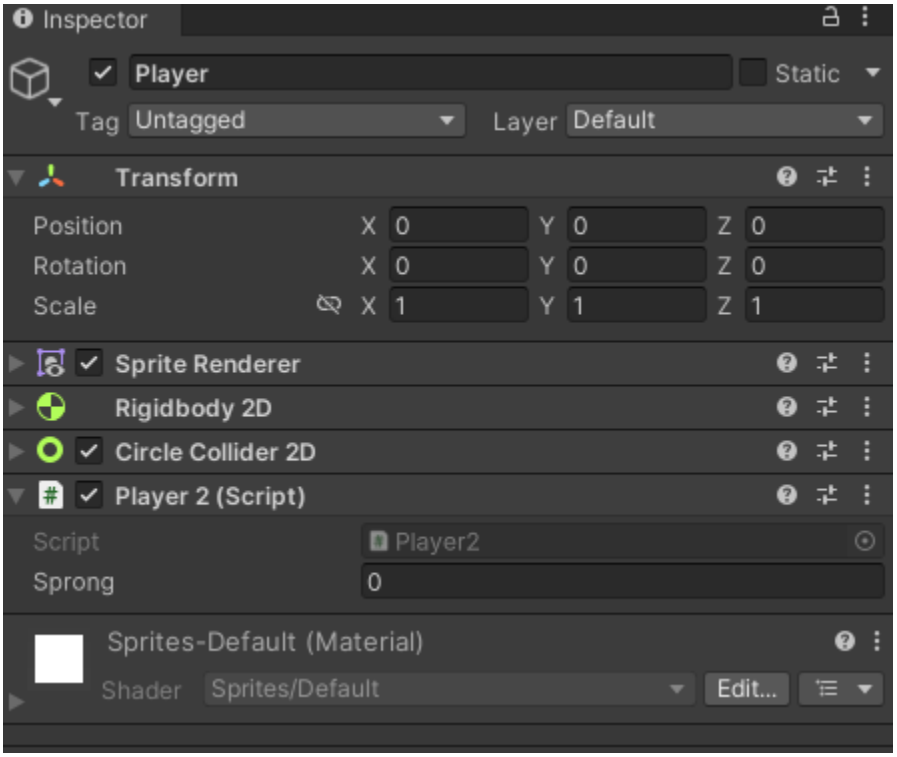
- Doe dit nogmaals maar dan voor het plafond. End deze noemen we Ceiling.



Stap 3: Begin met het Script

- Als je nu op start drukt zul je zien dat de Speler naar beneden valt en op de grond blijft liggen. Maar wij willen dat onze Speler kan vliegen. Om dit voor elkaar te krijgen gaan we het PlayerObject een force omhoog geven, dat doen we met Velocity.
- Maak een nieuw script genaamd 'PlayerController'. Klik met de rechtermuisknop in je Assets-map, ga naar Create, dan C# Script en noem het 'PlayerController'.
- Om bij de Velocity van ons PlayerObject te komen moeten we eerst naar de Rigidbody toe gaan. Dus binnen het 'PlayerController' script maken we een Rigidbody2D-variabel aan. Om dit te doen zeggen we private Rigidbody2D met een naam, *en definiëren die in onze start functie met “GetComponent<Rigidbody2D>();”. Dit doen we in start omdat start 1 keer wordt aangeroepen in het begin en daarna niet meer.
- Al onze variabelen worden Private gemaakt zodat ze niet opeens aangepast kunnen worden en ze blijven doen wat we willen.
- Als je dan de Rigidbody aanroept en er een punt achter zet kun je Velocity zoeken. Velocity is een Vector2 wat betekend dat ie 2 waardes heeft. 1 voor de X en 1 voor de Y as. Dus we maken een nieuwe Vector2 aan waar we de waarde X nul laten en de Y wordt onze sprong. Waar we een variabel voor maken. De ingebouwde Gravity trekt de Velocity naar beneden waar door we een realistische sprong creëren.
- De Variabel die we maken kunnen we Jump noemen. We maken deze Private met [SerializeField] ervoor. Dit zorgt ervoor dat we de waarde kunnen aanpassen in de Editor zonder het script aan te passen.
- Nu alleen nog zorgen dat het gebeurt wanneer wij willen, dus we zetten een If-statement in onze Update. We zetten de if in onze Update omdat Update elke frame wordt aangeroepen. In de If staat “Input.GetKeyDown(KeyCode.Space)”. En nu elke keer als we spatiebalk indrukken zou onze Speler omhoog moeten gaan.

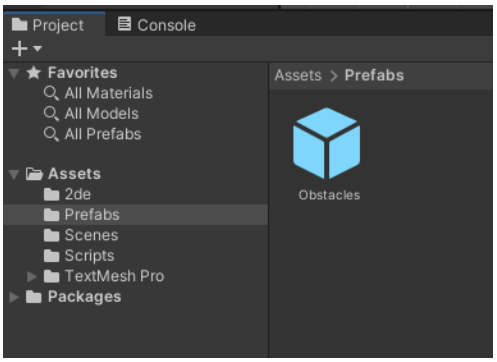
- Als je nu het PlayerController script naar je PlayerObject sleept, zal die er automatische opgaan en zou die het moeten doen. Nadat je de Jump waarde hebt aangepast.

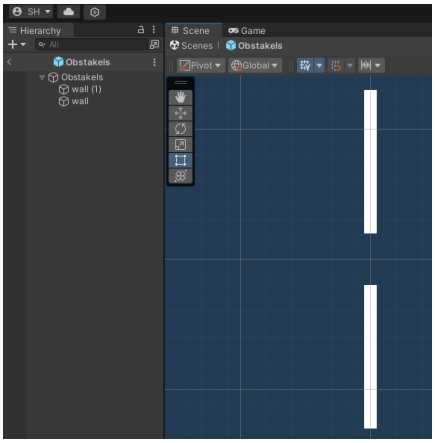


Hoofdstuk 3: De Obstakels

Stap 1: Creëer de Prefab

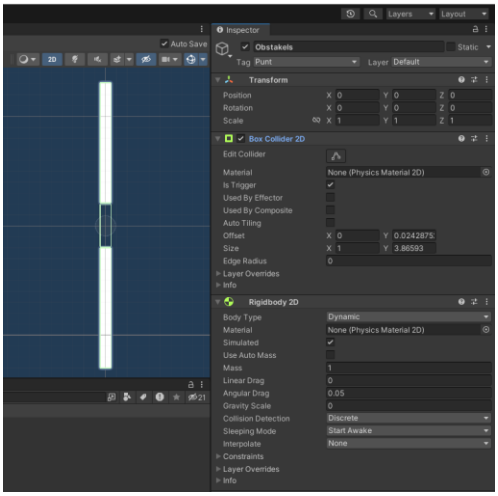
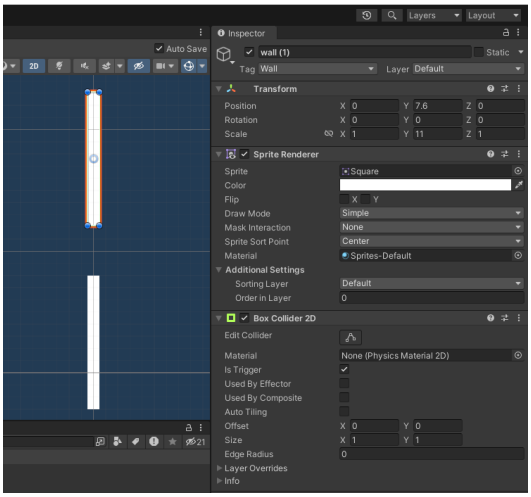
- Maak een Empty GameObject noem hem Obstacles en sleep het naar je assetsmap. Hiermee heb je zojuist een Prefab gemaakt. Open de Prefab met dubbelklik en plaats twee blokjes erin genaamd Wall met wat ruimte ertussen, zodat je Speler erdoorheen kan bewegen. Rek de Wall's uit zodat het echte palen worden. In de Scene komen de Wall's nu tevoorschijn en dat kun je gebruiken om aanpassingen te maken. Totdat je tevreden bent met de afmetingen.





- Voeg aan beiden Wall's een BoxCollider2D met Trigger toe.

- Voeg ook aan Obstacles een BoxCollider2D toe en maak deze tot een Trigger. Gebruik Edit Collider om hem te vormen naar het gat tussen de Wall's. Voeg ook een Rigidbody toe aan het Obstacles en zet de Gravity-scale op 0, omdat we niet willen dat het naar beneden valt.
- Nu gaan we Tags aanmaken voor later gebruik om te kijken met welk object we colliden in de toekomst. Klik op een Wall, druk op 'Tag', en selecteer 'Add Tag'. Maak twee Tags aan: een voor 'Wall' en een voor 'Point'. Maak van de Wall's ('Wall') en van Obstacles een('Point').


Obstacles

1 Wall

Stap 2: maak het 2de script.

- In dit 2de script doen we hetzelfde als in ons 1ste script. Maar eerst noemen we dit script Obstacles. We gaan de Velocity aanpassen. Maar nu stoppen we het in start en gaan we uit van Newton's eerste natuurwet. "*Something that is in motion, wil stay in motion.*" En in plaats van naar boven naar links dit keer. (Let op de X en Y as en denk aan negatief)
- Als we deze op de Prefab stoppen en de Scene runnen zal onze Prefab naar links gaan en nu moeten we nog zorgen dat ze opnieuw rechts verschijnen wanneer we dat willen.
- Dat doen we met een Timer als de Timer op nul staat dan gaat ie naar rechts. Een Timer in Unity is makkelijk te maken. Alles wat je nodig hebt is een Float en time.deltatime. Time.deltatime houdt namelijk bij hoeveel tijd er voorbij is gegaan tussen de stappen van update, als we dat van onze Float af halen kunnen we bij houden hoelang onze Prefab al bezig is.
- Dan kunnen we met een If bijhouden, of onze Float onder 0 is gekomen. Als die namelijk onder 0 is kunnen we met Transform.Position de positie van onze Prefab verplaatsen naar de rechterkant van ons scherm. Dat doen we door weer een Vector2 te maken en de positie van de X ver rechts zetten.
- Om het spel uitdagend te maken moeten we alleen nog de hoogte van de Prefabs veranderen doormiddel van een random. Je kan een random aanmaken door Random.Range(Int a, Int b). Nu wordt er een random getal getrokken tussen 2 getallen die we kunnen gebruiken voor de hoogte.
- Stop dit script nu op je Prefab.
- Maak van de variabelen [SerializeField] om rond te spelen met de waardes en te kijken welke het beste werken voor jou game. Vergeet niet een 2de Float te maken 1 om de tijd bij te houden en 1 om je Timer te resetten.

Stap 3: maak een manager die de objecten aanmaakt.

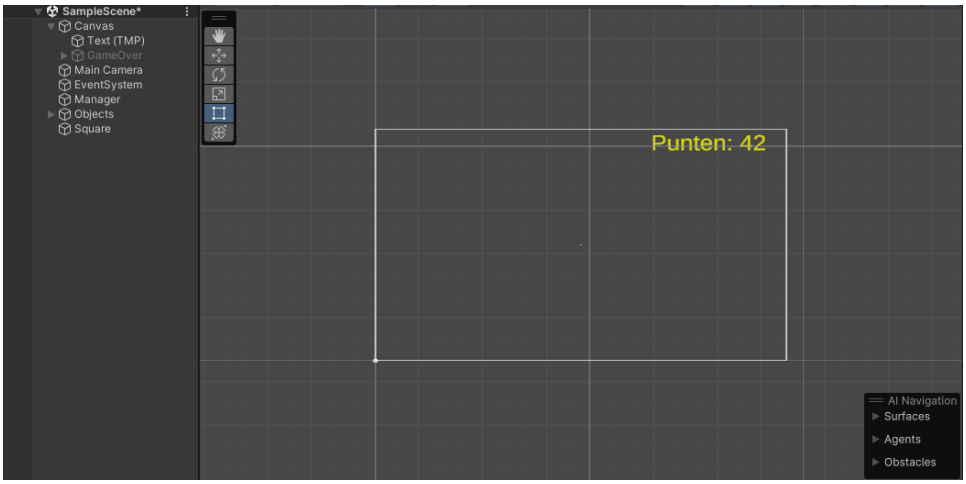
- Maak nog een script aan genaamd Manager en maak van je Prefab een variabel aan. Dat doe je door het script aan te roepen dat op de Prefab staat. Bijvoorbeeld "[SerializeField] Private Obstacles obstakel" Maak ook een Timer en een Counter variabel aan waar Counter een Int is. Counter gaat bijhouden hoeveel Obstacle Prefabs er al gespawned zijn.
- Nu gaan we een Private functie maken, die onze objecten de eerste keer in de wereld zet, zodat we niet alles hoeven uit te meten in onze Scene. Om een functie te maken typ je Private void "naam"(). De reden dat er void tussen staat is omdat we niks return-en in onze functie. Zouden we dat wel doen kun je daar bijvoorbeeld een Int (Integer) of Bool (Boolean) neerzetten.
- In de Functie gooien we onze Counter 1 omhoog en instantiëren we onze Prefab. Dat doen we met Instantiate(Prefab, Position, Rotation) voor Rotation gebruiken we Queternion.identity omdat het GameObject niet gedraaid hoeft te worden. En dan resetten we de Timer. Roep de functie aan in de Update, zorg dat je niet te veel objecten maakt met behulp van Counter en dat het werkt op een Timer.
- Omdat we in het manager script meerdere Libraries gaan gebruiken, met meerdere versies van random, moeten we boven in het script met de Libraries aangeven welke random we gebruiken. Dat doen we met: using Random = UnityEngine.Random;.
- Maak dan een nieuw empty GameObject aan, noem deze Manager en sleep het ManagerScript erop.
- (TIP: zorg dat de ruimte tussen de Obstacles even groot blijft voor een soepele loop). Prefab respawnTime / Count = Timer tussen Obstacles. Oftewel de tijd dat de Obstacles bestaan, deel je door het aantal Obstacles, en die tijd moet er tussen het aanmaken van de Obstacles zitten.
- Om bij de respawntijd van de Obstacles te komen maken we een public function GetTimer() in de Obstacles Script. Om de waarde terug te sturen returnen we inplaats van een void een float en dan kunnen we return Timer erin zetten. En als we de Obstacles dan aanroepen kunnen we GetTimer() ook gelijk aanroepen.

Hoofdstuk 4: Punten Systeem

Stap 1: de punten

- We gaan beginnen bij de Speler en zorgen dat die weet hoeveel punten die heeft en hebben het PlayerScript nodig. Daarvoor maken we de variabel aan. We gebruiken de “private void OnTriggerExit2D(Collider2D col)” functie. Deze ingebouwde Unity functie wordt automatische aangeroepen als de Speler een Collider verlaat. De parameter in dit geval is de Collider waarmee we in contact zijn. In ons geval is de Collider die we willen de Collider van Obstacles.
- Om te zorgen dat de Obstacles-Collider ook gebruikt wordt hebben we een Tag aangemaakt. En die kunnen we checken met een If statement If(col.GameObject.CompareTag("Point")). En dan voegen we een punt erbij.
- Maar we moeten ook punten kunnen kwijtraken als we de Wall raken. Maak met je huidige kennis die functie aan. Om ons werk te controleren kunnen we Print(punten) gebruiken als we een punt halen, deze wordt dan uitgeprint in de console.
  (TIP: Vergeet niet 2D toe te voegen aan de functie als het niet werkt).
- Nu zou het ook leuk zijn als we onze score konden zien. Dus maak nog een GameObject, ga naar UI, text TextMeshPro. Nu komt er een melding om UI te importeren dat is allemaal oke en daarna komt het Canvas tevoorschijn.

- Plaats de TextBox op de gewenste locatie en verander de kleur wat donkerder bij de extra instellingen zodat ie goed zichtbaar blijft. En verander de text naar "Punten: 42"



- Maak in het PlayerController script een variabel van TMP_Text aan en maak hem [SerializeField], plaats deze variabel in de update met ".text" erachter. En maak hem gelijk aan de punten. (Je kan ook extra tekst toevoegen door " Punten: " + punten; te gebruiken. Alles tussen " " wordt letterlijk erin gezet).
- Nu alleen nog ons TextObject linken aan het PlayerObject en je hebt een werkende GameLoop.

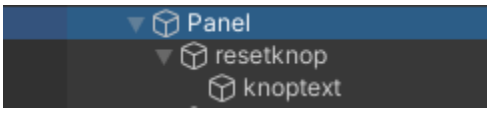
Stap 2: de High Scores

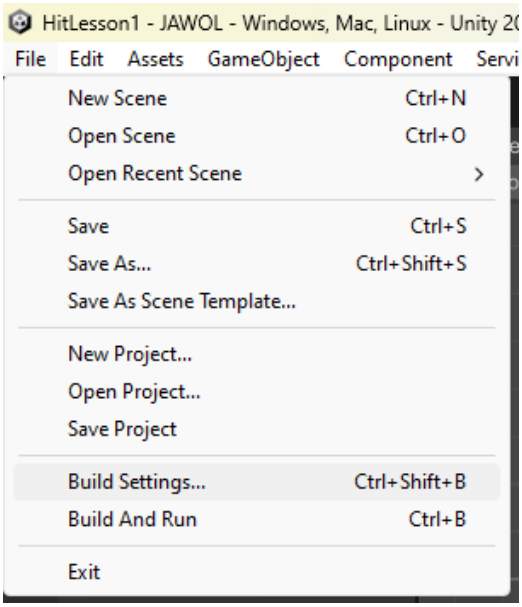
- Om onze punten op te slaan gaan we gebruik maken van onze Manager en PlayerPrefs. Aangezien PlayerPrefs per naam worden opgeslagen, gaan we voor al onze highscores een naam maken, deze namen slaan we dan op in een String Array. Als we het een [SerializeField] maken kunnen we in de editor de scorehoeveelheid aanpassen.
  Dan kunnen we met PlayerPrefs.GetInt(Name); de score krijgen en met PlayerPrefs.SetInt(Name, punten) kunnen we de score opslaan.
- Om de Highscore op te slaan maken we een functie die we aanroepen als de Player een muur raakt, en geven de punten mee als parameter.
- In plaats van de punten op 0 zetten, kunnen we de HitWall functie aanroepen bij PlayerController als de Player de Wall hit. We moeten alleen de manager linken en dat doen we door een [SerializeField] manager te maken en onze manager te linken.
- En we moeten weten of de behaalde score hoger is dan de huidige high score. We doen dit door te zorgen dat de laagste score altijd op positie 0 zit van de Array. Dan kijken we of de huidige punten hoger zijn. Als die hoger zijn schrijven we de nieuwe score over deze positie heen.
- Maar om te zorgen dat onze lijst oplopend is maken we een Private functie SortArray. In deze functie vullen we een nieuwe Array met alle scores van de String Array op dezelfde plekken doormiddel van een For-loop en Playerprefs.GetInt(NameArray[i]). Als je de tijdelijke Array definieert zorg dan dat deze even groot is als de vaste Array.
- Vervolgens zorgen we met "Array.Sort(Nieuwe Array, oude)", dat onze oude Array met de juiste scores gesorteerd wordt. Van klein naar groot.
- Nu kunnen we kijken of de nieuwe score hoger is dan positie 0 van onze lijst, als die hoger is zetten we de nieuwe score met PlayerPrefs.SetInt(naam, Punten).

- En dan maken we een For-loop die start met i = 1, dit doen we zodat we met i en met i – 1 kunnen werken zodat we niet boven/onder onze array kunnen komen.
- Vervolgens kijken we of de punten ook hoger zijn dan de opvolgende. Als die hoger is switchen we die posities. En dat kan heel makkelijk met (lijst[i - 1], lijst[i]) = (lijst[i], lijst[i - 1]);.
  Dit zorgt dat de i – 1 en de i met elkaar verwisseld wordt.
- Vergeet ook niet een reset functie te maken, voor het geval dat je de scores wilt resetten, en laat het activeren met een “KeyPress” bijvoorbeeld.

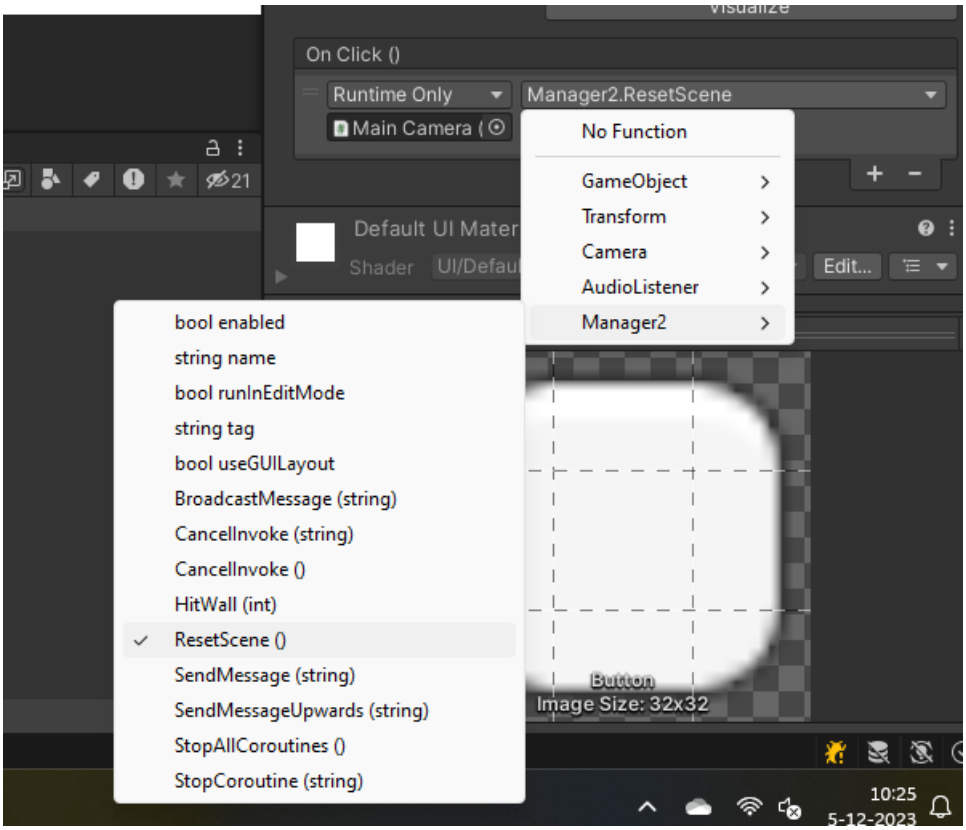
Hoofdstuk 5 UI:

Stap 1 Reset Knop:

- Nu gaan we aan ons GameOver scherm werken en we beginnen met een panel aan te maken bij UI en de Form aan te passen naar wat we willen. Daarna maken we een knop en die zetten we onder het panel. Dat doen we door de knop te slepen naar het panel. En dan gaan we naar Text onder de button en veranderen we de tekst naar GameOver.
- 

- Knoppen in Unity werken heel makkelijk. Om een knop te gebruiken druk je op de plus bij “onClick()”. Dan komt er een blokje tevoorschijn en hierin kun je 1 van je objecten slepen met het werkende script. En dan kun je alle functies aanroepen zolang ze Public zijn.
- Als de knop op de juiste plek staat gaan we de manager aan de knop toevoegen. En dan gaan we een functie schrijven in de manager die de knop uit gaat voeren. Om te zorgen dat deze functie uitgevoerd kan worden moet de functie Public omdat de knop erbij moet kunnen komen.
- De functie die we gaan maken wordt “ResetScene()” deze functie gaat de huidige scene opnieuw aanroepen waardoor alles wordt gereset en we een verse start hebben. Op het Internet is te vinden hoe we van scene switchen.
  (Om te weten waar we heen moeten switchen kijken we naar File/BuildSettings daar kunnen we al onze scenes zien. En op welk nummer)
- 

- Als deze functie is gemaakt kunnen we hem aanroepen bij onze knop. Als alles goed gegaan is kun je nu op de knop drukken en dan zal de scene opnieuw laden

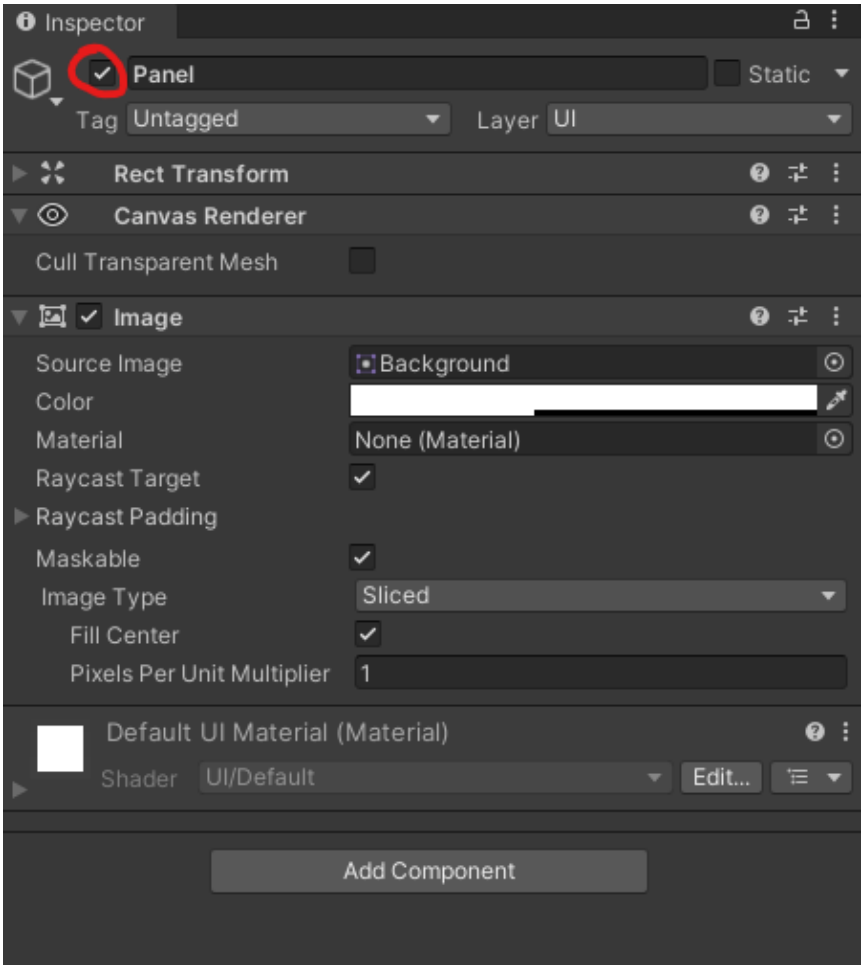


Stap 2 HighScores:

- Nu zou het mooi zijn als onze highscores geshowd worden als we GameOver zijn.
- Om dit te doen maken we een nieuw tekst object, stoppen het text object in onze panel en dan gaan we in onze Manager een functie maken die ons text object gaat vullen. Het mooie hieraan is dat we het text object kunnen linken aan een String, dus als we 1 String maken met alle tekst kunnen we op die manier alles in 1 keer over zetten. Dus we maken in onze functie een String met "GameOver\nHighscores\n". We gebruiken de \n om aan te geven dat we aan het einde van onze regel zijn. En dan met een For-loop alle scores toevoegen.
  (TIP: For-loop van scores.length naar 0 brengen zodat de scores de goeie volgorde hebben.)
- En dan kunnen we voor al onze scores String += de nieuwe score doen + "\n". Zodat alle scores mooi onder elkaar komen te staan. En dan is onze nieuwe tekst.text = String. En die roepen we

aan als onze highscore gecheckt wordt.

- Om te zorgen dat ons menu alleen te zien is als die nodig is gaan we in de editor het panel uit zetten. (Zie illustratie hieronder)
  In onze code gaan we hem aan zetten wanneer die nodig is. Voor dit hoeven we alleen ons panel als GameObject aan te roepen. En .SetActive(true) op het moment dat we ons panel willen aanroepen.

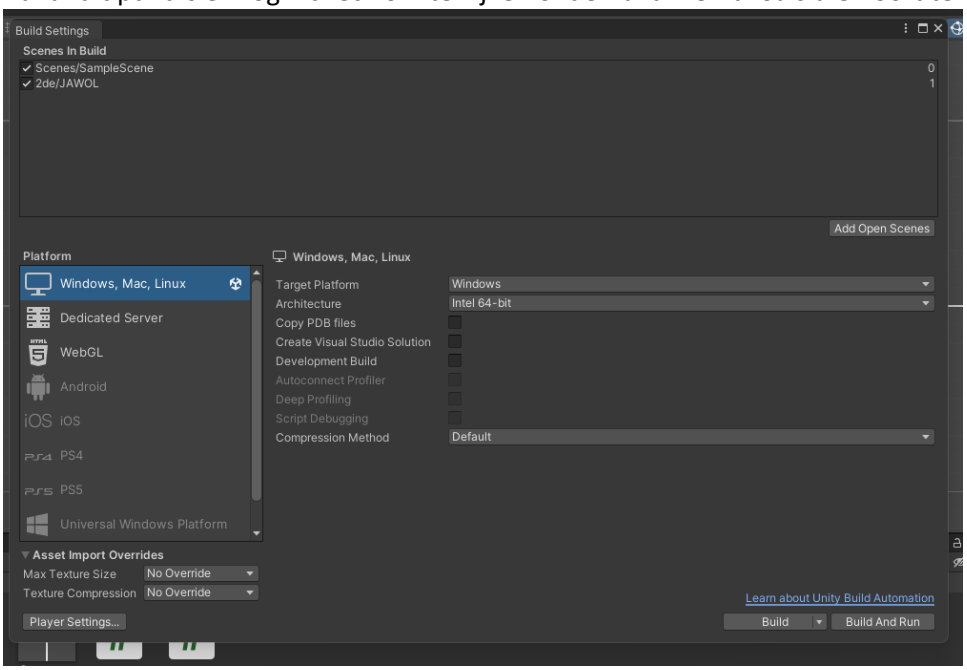


Stap 3 Game pauzeren:

- Om te zorgen dat ons spel stilstaat (of pauzeert) terwijl het GameOver scherm zichtbaar is, kunnen we de TimeScale uit zetten door te zeggen “Time.TimeScale = 0;”
  Daardoor gaan al onze objecten stilstaan terwijl ons menu tonen. Vergeet niet om de TimeScale weer op 1 te zetten, als we van scene switchen omdat de TimeScale niet gereset wordt als we een nieuwe scène laden.

Hoofdstuk 6 De Build:

- Nu we de core-features hebben van onze game, gaan we de Alpha-Build maken (of versie 1.0.0). Een Build is een versie van de game die draait zonder de Unity Editor omgeving, maar een losstaande executabel. Om te zorgen dat we ons spel kunnen afsluiten moeten we in onze manager nog 1 button input toevoegen waar we Application.Quit(); aan toevoegen. Dit zorgt ervoor dat ons spel afgesloten kan worden.
- Nu komt de Test-fase, in deze fase gaan we testen of er bugs/errors in onze game zitten en of de game functioneert naar behoren. Hier houden we ons ook bezig met het balanceren van onze game om te zorgen dat het niet te moeilijk en niet te makkelijk is.
- Om een Build te maken gaan we naar File/BuildSettings en Build. Maak een mapje in je Githubmap genaamd "AlphaBuild", sla hier de Alpha Build op.
  “%USERPROFILE%/Github/Unity2D_ArcadeProject/AlphaBuild/”
- Vanaf dit punt is er nog 1 check om te kijken of de Build werkt zoals die hoort te werken.



Project "Could-Have"

- 3 Letters input voor naam Highscore (Als oude arcade kasten)
- Extern "PlayerPrevs" om Highscores te kunnen exporteren naar andere machines
- Bounce mechanics voor een minder statisch gevoel.

- Extra keybind voor het resetten van de scene zodat je niet de knop hoeft in te klikken.
- Parents uitleggen en linken voor de Obstacles.
- Break toevoegen bij het controleren voor codes voor efficiëntere code.
- Cursor uitzetten en extra ingaan op build specIfieks en Interface
- Sound design implementeren.
- Mensen leren commenten.