2D Endless Runner

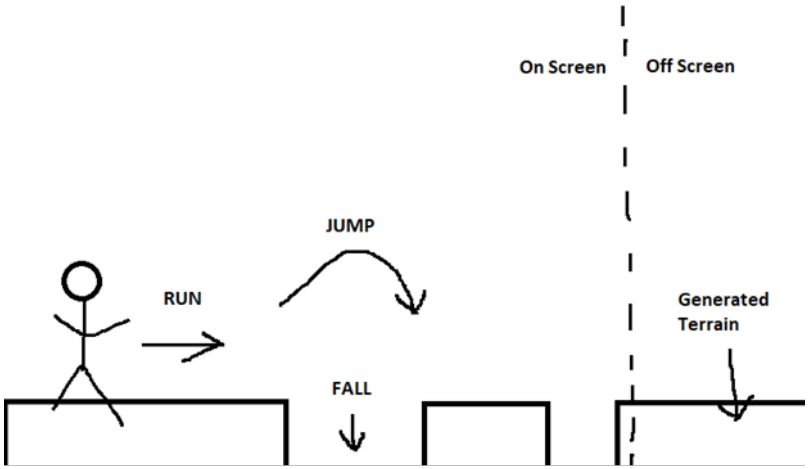
Site: DILo Game Academy
Course: Game Programming Studi Independen
Book: 2D Endless Runner
Printed by: 408 Dewa Sinar Surya
Date: Monday, 4 October 2021, 7:54 PM

Table of contents

1. Pengantar

Pada materi kali ini kita akan membahas cara membuat game 2D Endless Runner, yaitu sebuah game dimana si player akan bergerak ke kanan secara terus menerus melewati rintangan tanpa ada batasan. Semakin jauh si player bergerak, maka semakin besar skor yang didapat. Game ini bisa sudah cukup banyak dipasaran dengan fitur2 nya yang keren-keren, tapi kali ini kita akan fokus untuk membuat sistem utama dari game 2D Endless Runner nya saja.

Untuk gambaran awal, seperti inilah game yang akan kita buat



Dari gambar, kita bisa bayangkan pada game ini akan ada karakter yang berlari ke kanan dan harus melompati lubang, jika gagal maka kita bisa jatuh dan game over. Selain itu terrain yang muncul akan di generate secara otomatis di luar layar kamera. Dari informasi ini, kita bisa bagikan dua sistem yang nantinya akan dibuat, yaitu pergerakan karakternya dan proses generate terrain. Selain itu kita juga akan tambahkan sistem skoring dan game over agar permainan lebih menarik. Jadi kalau kita simpulkan, kita akan membuat fitur-fitur berikut

- Character Movement
- Terrain Generation
- Score & Game Over

Sebelum mulai membuat gamenya, silakan download asset di link-link berikut:

https://drive.google.com/file/d/1swfdl714P5FQxKnUDGxnbp4J1wHthnXs/view?usp=sharing

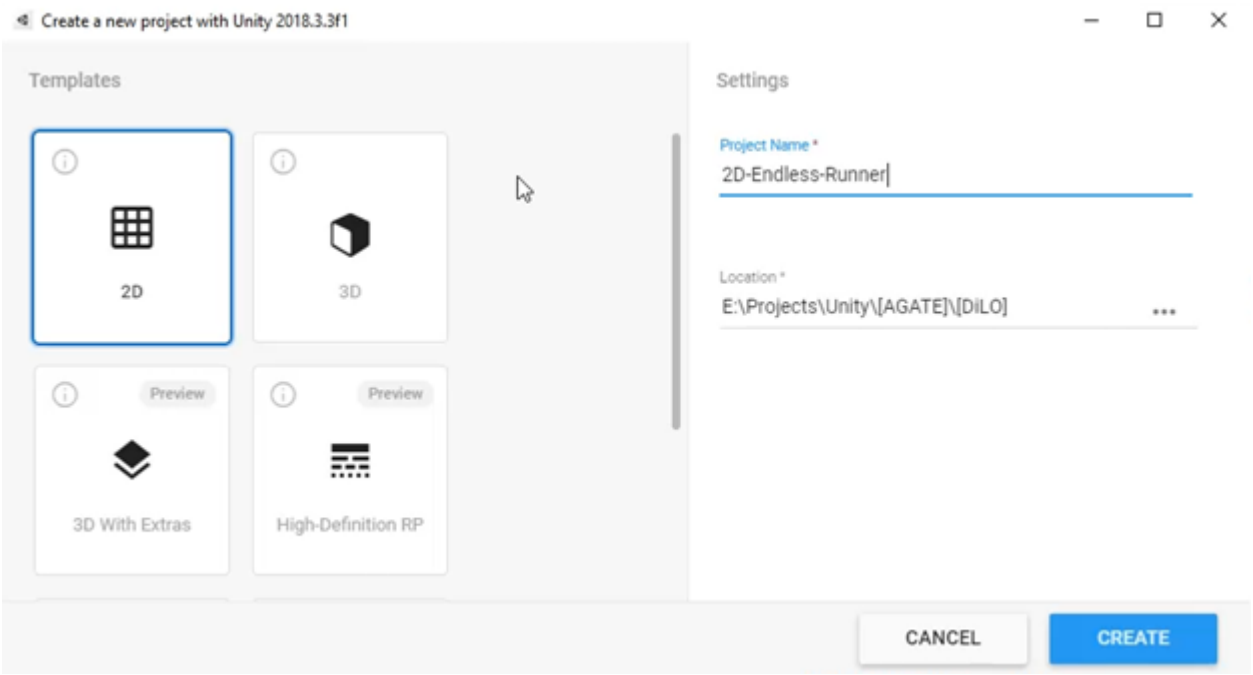
Versi Unity yang digunakan dalam penulisan materi ini : 2018.3.3f1

Kamu juga bisa melihat rekaman praktik pembuatan project ini di link berikut:

2. Chapter 1 - Project & Asset Setup

New Project

Pertama, kita buat project baru dulu, di pilihan project nya kita bisa pilih 2D project agar unity mengatur project kita supaya langsung bisa dimainkan secara 2D.



Import Asset

Bila sampai tahap ini kamu belum mendownload file asset untuk project ini, silakan download terlebih dulu dari link yang diberikan pada page sebelumnya. Filenya dalam bentuk zip jadi kalian perlu untuk mengekstrak nya terlebih dahulu, kemudian masukan file nya ke dalam project unity kita dengan cara drag n drop ke tab project di unity.

Set Up Asset

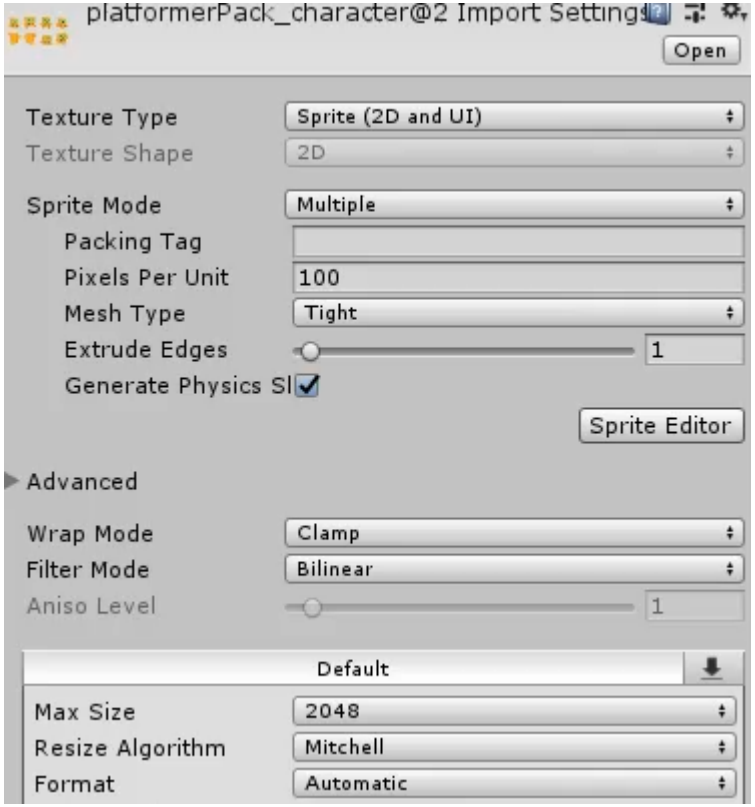
Setelah kita masukan asset-nya, kita rapikan foldernya agar tidak berantakan.

- Masukan asset gambar (png) ke dalam folder Sprites
- Masukan asset suara (ogg) ke dalam folder Sounds

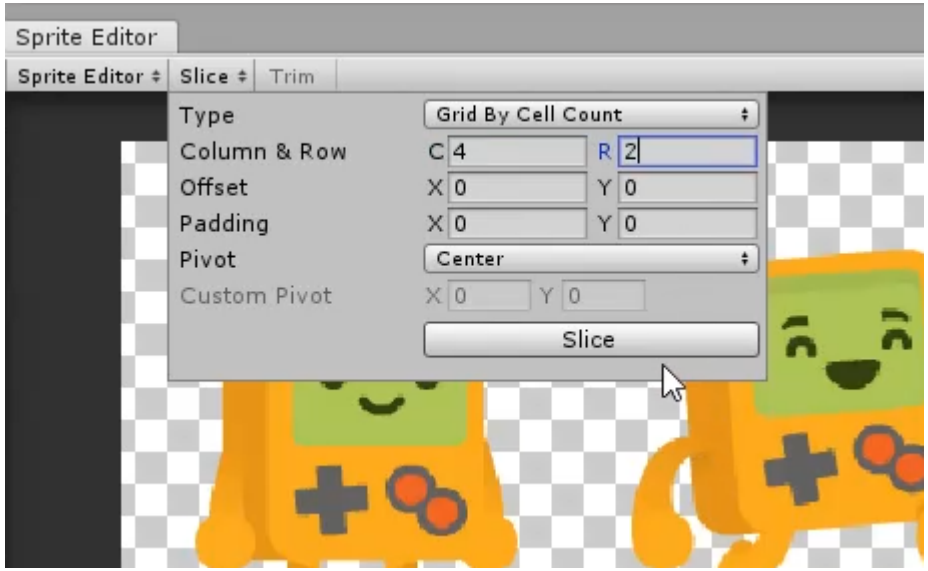
Hal ini penting dilakukan agar kita tidak sulit untuk mencari asset dalam project kita, kalian bisa atur folder sesuai keinginan kalian juga kok, yang penting jangan sampai berantakan file nya karena akan menyulitkan kalian sendiri nanti saat project game kalian semakin membesar

Kita juga perlu mengatur asset gambar kita karena ada dalam bentuk spritesheet, yang kita inginkan adalah bentuk sprite satu-satu yang merupakan potongan dari spritesheet tersebut, untuk membuatnya berubah menjadi sprite, kita bisa lakukan ini:

- Select salah satu asset spritesheet
- Ubah *Sprite Mode* nya menjadi *Multiple*

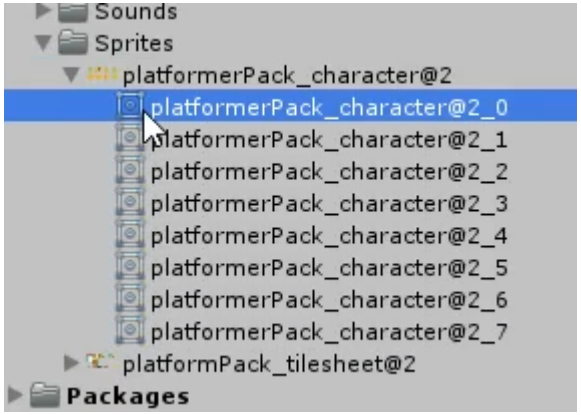


- Klik *Apply* untuk menyimpan perubahannya
- Klik *Sprite Editor*, akan muncul window *Sprite Editor* di layar
- Klik *Slice* untuk membuka menu *Slice*
- Untuk melakukan slice dengan mudah pada kalian bisa menggunakan tipe *Grid by Cell Count*
- Menggunakan tipe *Automatic* terlalu risky karena bisa menghasilkan ukuran sprite yang berbeda2 sehingga sprite kita tidak bisa di animasikan



- Untuk spritesheet *character* kalian bisa gunakan
- *Grid by Cell Count*
- Column : 4
- Row : 2
- Untuk spritesheet *tilesheet* kalian bisa gunakan
- *Grid by Cell Count*
- Column : 14
- Row : 7
-

Kalian bisa mengecek sprite yang sudah dipotong pada bagian preview dengan cara mengklik sprite yang ingin kalian cek. Cara lainnya adalah kalian klik dua kali tanda panah pada file spritenya.

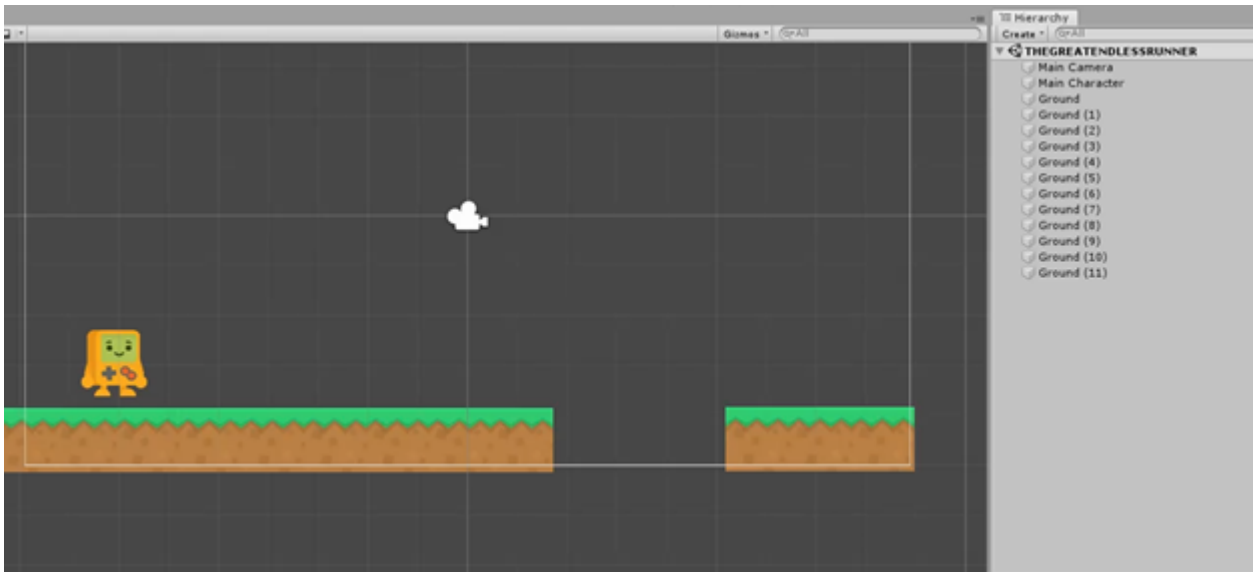

Sounds
Sprites
platformerPack_character@2
platformerPack_character@2_0
platformerPack_character@2_1
platformerPack_character@2_2
platformerPack_character@2_3
platformerPack_character@2_4
platformerPack_character@2_5
platformerPack_character@2_6
platformerPack_character@2_7
platformPack_tilesheet@2
Packages

3. Chapter 1 - Set Up Scene

Set Up Scene

Sekarang, agar kita bisa melihat seperti apa game yang akan kita buat, kita bisa langsung memasukan sprite-sprite yang sudah kita potong tadi ke dalam scene, masukan karakter game dengan cara drag sprite karakter yang kalian inginkan ke dalam hierarchy, lalu posisikan sprite nya di tempat yang kalian inginkan. Ingat untuk me-rename nama sprite tersebut pada hierarchy menjadi "main character"

Kemudian untuk pijakan si playernya, kalian bisa ambil salah satu bentuk terrain atau tanah dari tilesheet kemudian atur sehingga menjadi terrain yang menggambarkan game yang akan kalian buat. Disini juga kita akan buat lubang sebagai contoh kalau di game ini karakter akan menemui sebuah lubang dan harus melompati lubang tersebut. Tips: kalian bisa menduplicate objek tersebut dengan shortcut CTRL + D



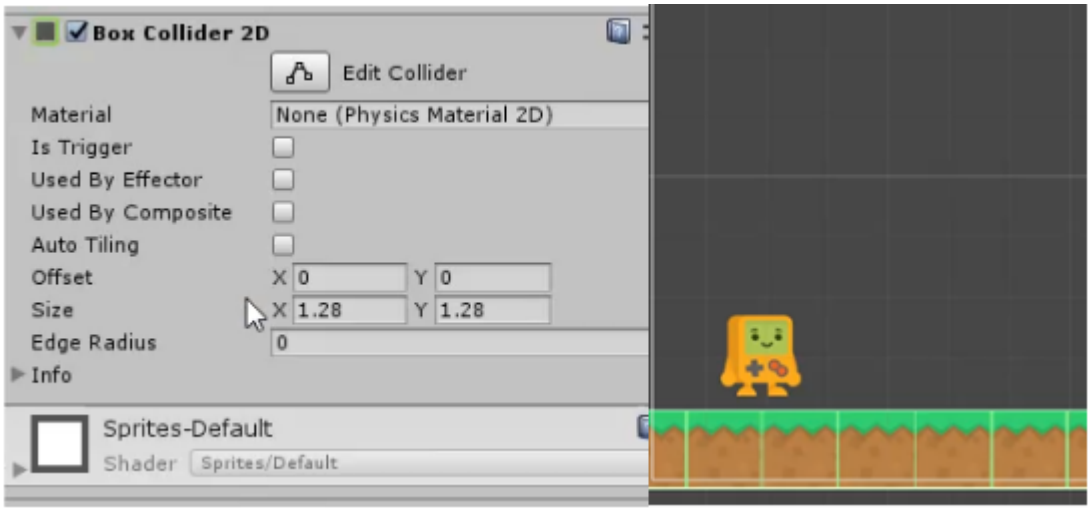
Step ini sebenarnya memang terlihat tidak terlalu penting bagi proses pembuatan game, akan tetapi ini akan membantu kita untuk membayangkan gameplay dari game yang akan kita buat.

Collider and Rigidbody

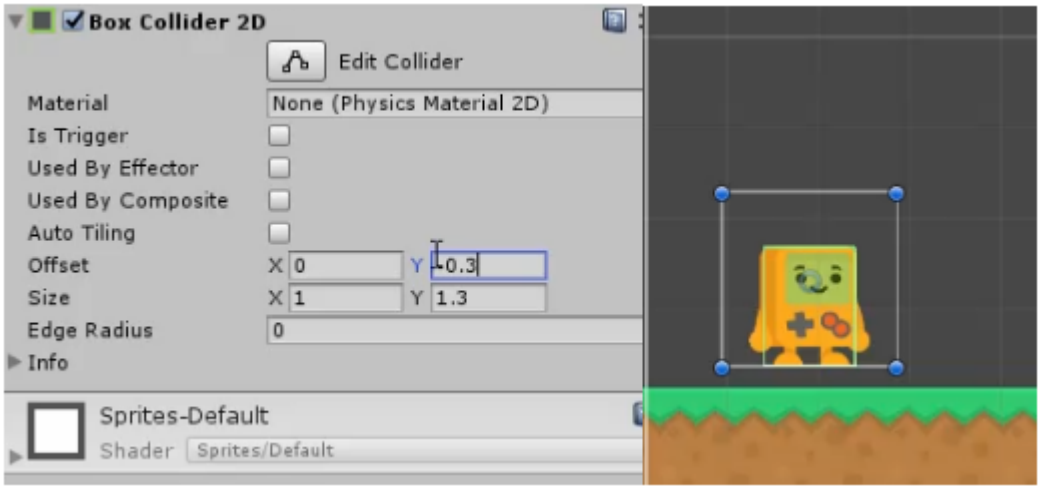
Setelah kalian mengatur tampilan scenenya, kita tambahkan collider pada terrain kita dengan cara:

- Select sprite yang ingin ditambahkan collider, untuk terrain kita bisa langsung select semua terrain yang ada
- Klik Add Component

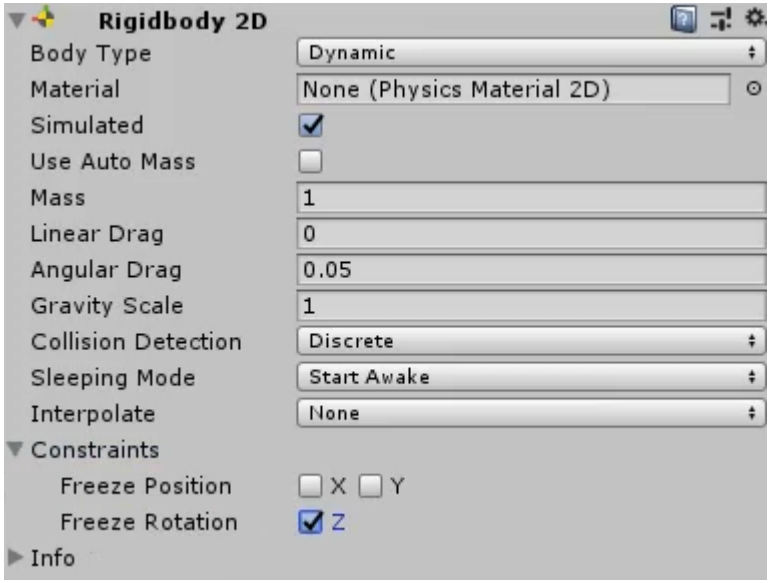
- Ketik component yang diinginkan pada search bar, disini kita ketik Collider 2D
- Klik Collider yang diinginkan, disini kita akan menggunakan Box Collider 2D karena bentuk terrain kita adalah kotak



Lakukan hal yang sama untuk karakter kita, tapi untuk karakter perlu sedikit adjustment agar collidernya pas, kita bisa atur size dan offsetnya



Kemudian kita tambahkan Rigidbody 2D pada karakter kita dengan cara add component, lalu cari Rigidbody 2D. Ceklist Freeze Rotation Z pada karakter karena kita tidak ingin karakter berputar2

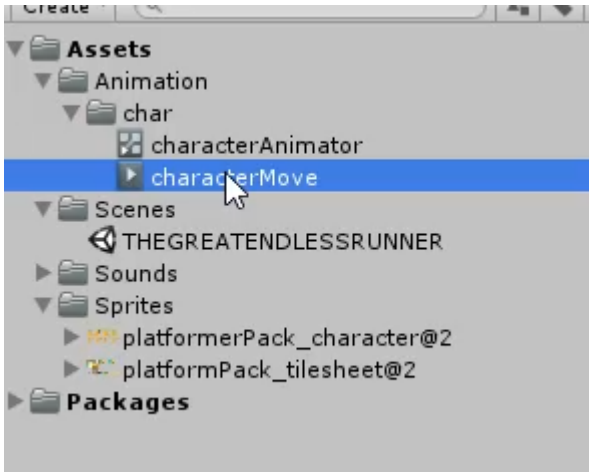


Selesai! Kalian bisa coba play scene nya, jika karakter kalian jauh ke terrain, maka kalian berhasil membuatnya. pastikan player lebih tinggi dari terrain agar player tidak menempel pada collider.

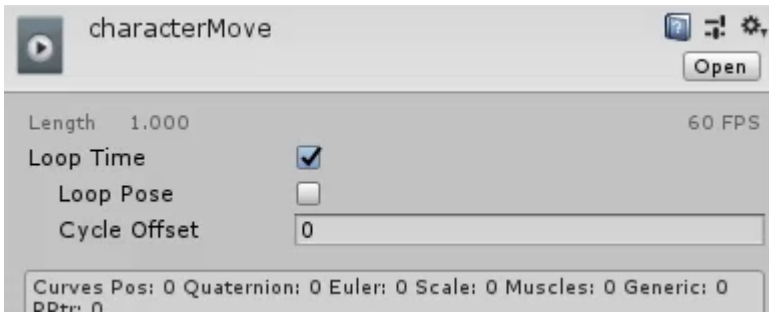
4. Chapter 1 - Using Animation

Using Animation

Sekarang kita akan tambahkan animasi supaya karakter kita terasa lebih hidup, kalian bisa buat asset animasi dengan cara klik kanan pada window project, pilih create, lalu pilih Animation Controller. Tulis nama yang sesuai dengan keperluannya seperti “characterAnimator” (Animator adalah singkatan dari Animation Controller). Animator ini adalah asset yang nantinya akan mengatur animasi yang dijalankan.



Lakukan hal yang sama, tapi sekarang pilih Animation. Untuk sekarang kita akan mencoba membuat animasi berjalan jadi kita bisa namakan “characterMove”. Asset Animation ini lah yang berisi animasi sebenarnya yang dijalankan nanti. Setelah ditambahkan, kita pilih asset animasinya dan kita ceklist Loop Time agar animasi nya di-set sebagai animasi yang berulang

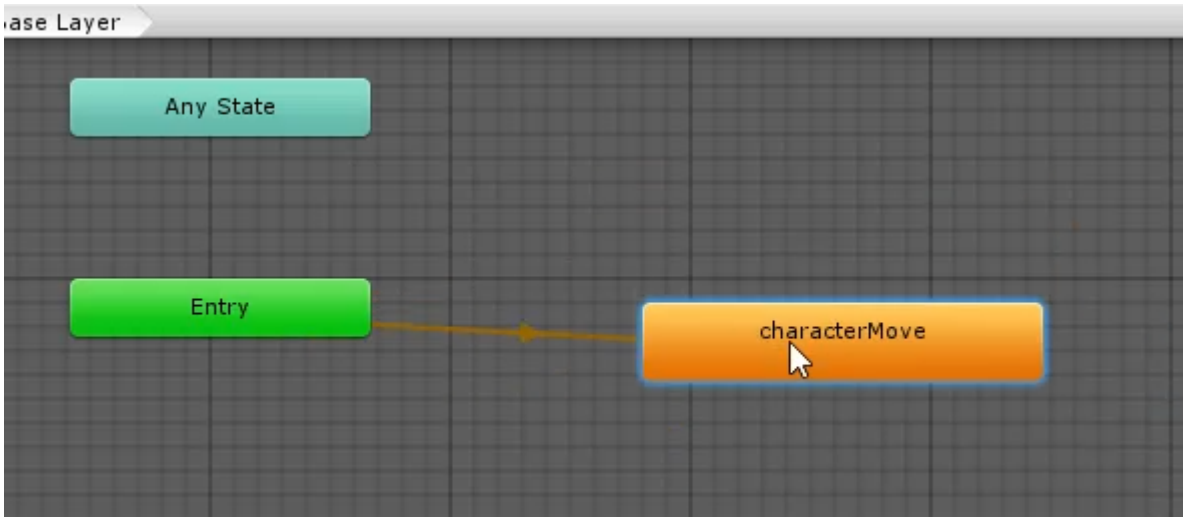


Sekarang kita bisa tambahkan animator yang kita sudah buat tadi ke karakter kita dengan cara drag asset animatornya ke dalam inspector karakter

kita, pastikan pada hierarchy karakter kita yang sedang dipilih.

Jika kalian kesulitan lihat kursor mouse kalian saat drag animatornya, posisi kan kursor mouse di pembatas antar component, saat icon berubah tandanya anda bisa menambahkan component di situ. lepaskan drag dan component akan muncul di inspector

Sekarang kita buka animatornya dengan cara double klik. Disini kita hanya perlu drag asset animasi characterMove kita tadi ke dalam animatornya, Unity akan secara default membuat animasi ini menjadi animasi default.



Sekarang kita buat animasinya, dengan cara membukanya dari Window -> Animation -> Animation, Setelah window Animation terbuka, klik object pada hierarchy yang memiliki animasi yang ingin kita ubah, yaitu karakter kita.

Masukan sprite karakter ke dalam tab animation dengan cara drag n drop sprite karakter dari tab project ke tab animation.

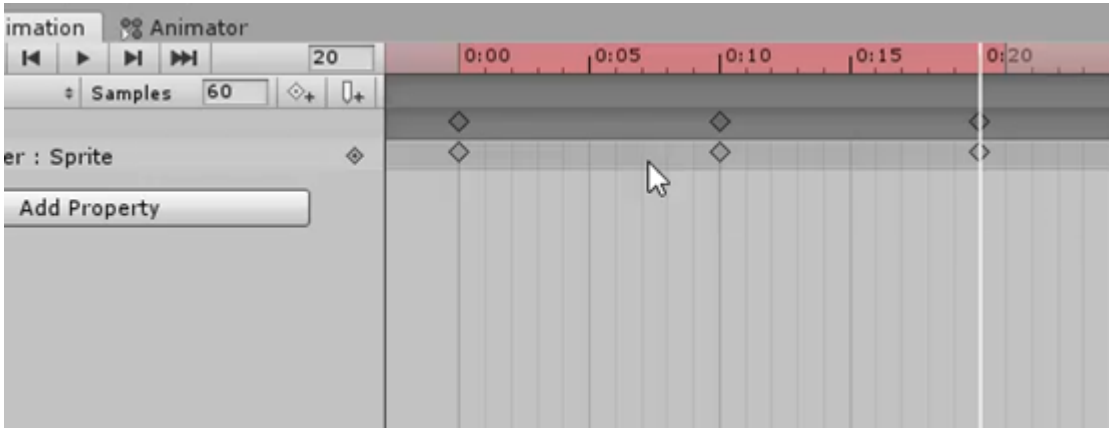
Untuk sekarang, kita hanya akan membuat animasi yang mengubah sprite dari object karakter kita agar terlihat seperti berjalan, untuk membuat animasi dengan mudah kita bisa lakukan hal ini:

- Klik record, maka animasi akan berubah menjadi warna merah yang artinya unity sedang menyimpan perubahan yang kita buat sebagai animasi
- Klik waktu ingin kita buat keyframe pada timeline animasi
- Ubah value-value komponen pada object yang ingin kita animasikan, untuk sekarang kita ganti sprite nya agar terlihat bergerak
- Kalian bisa mengikuti timeline ini jika bingung
- 0:00 = “_3”

- 0:10 = “_2”
- 0.20 = “_3”

tips: cara mengganti sprite pada timeline animasi adalah drag n drop sprite dari tab project ke keyframe pada tab animasi

- Klik tombol record lagi jika kalian sudah puas dengan animasinya



Dan selesai! Kalian bisa coba play scene nya lagi untuk melihat animasi yang telah kalian buat

5. Chapter 2 - Moving Character & Using Physic

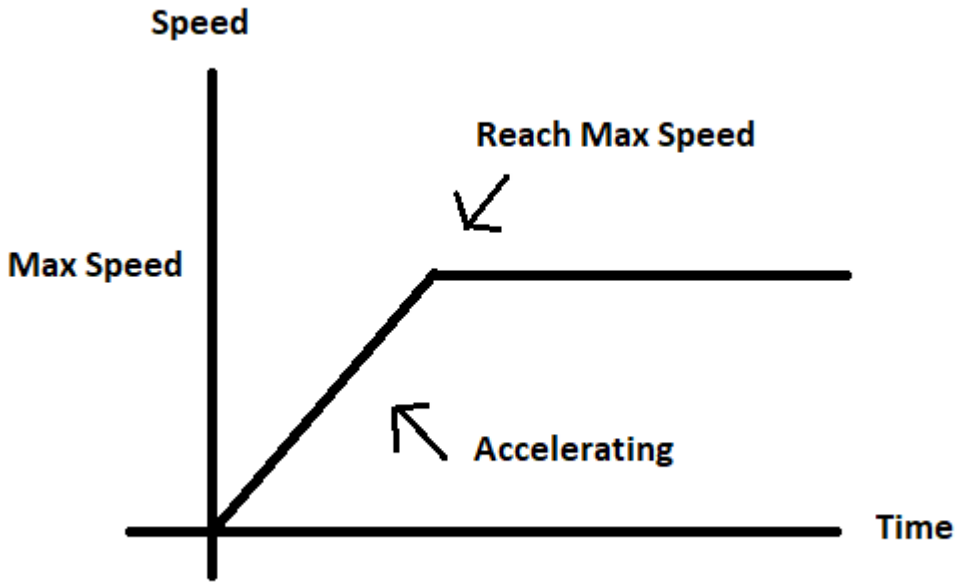
Moving Character

Sekarang kita akan membuat karakter kita berjalan ke kanan, kedengarannya mudah kan? Untuk menggerakan player kita ke kanan kita hanya perlu memindahkan posisinya ke kanan saja setiap frame. Tapi bagi programmer ternyata tidak semudah itu. Sebelum kita langsung membuat karakter kita bergerak, kita perlu mendefinisikan bagaimana dia bergerak terlebih dahulu.

Pada umumnya game memiliki 2 jenis tipe gerakan, yaitu

- gerakan dengan akselerasi, seperti pada game balapan
- gerakan tanpa akselerasi, seperti pada game 2D RPG

Pada game ini, kita akan membuat gerakan dengan akselerasi sehingga player kita akan secara perlahan menambah kecepatannya hingga mencapai kecepatan maksimal. Mari kita lihat grafik di bawah untuk memahami desain pergerakan karakter di game kita



Buat folder untuk menyimpan seluruh script untuk project ini. Caranya adalah klik kanan pada tab project > create > folder. Beri nama Scripts. Ok sekarang kita buat script baru di unity dengan cara klik kanan pada project window, pilih create, lalu pilih c# script. Kita namakan script kita CharacterMoveController yang nantinya bertugas untuk mengendalikan seluruh gerakan player kita. Kemudiian kita pasang pada player kita.

Pada scriptnya, kita tambahkan dua variabel, yaitu moveAccel untuk mengatur akselerasi dan maxSpeed untuk mengatur kecepatan maksimum. Kita buat keduanya public agar bisa diubah-ubah langsung di inspector. Script ini dimasukkan dalam "public class CharacterMoveController : MonoBehaviour"

```
[Header("Movement")]
public float moveAccel;
public float maxSpeed;
```

Agar script bisa dibuka di inspector untuk karakter, cara yang bisa kamu lakukan adalah memasukkan script tersebut pada game object tersebut. Caranya: klik objek karakter pada tab hierarchy. Lalu pada bagian inspector, klik Add component > search "Character Move Controller"

Untuk membuat karakter kita bisa bergerak, kita bisa menggunakan rigidbody yang telah kita tambahkan di chapter sebelumnya untuk mengubah kecepatan karakter kita. Pada script buat fungsi baru bernama FixedUpdate () setelah fungsi Update(). Lalu kita ambil rigidbody kita pada Start kemudian pada FixedUpdate kita gerakan player kita ke kanan, pastikan perubahan yang menggunakan physics engine ada di dalam FixedUpdate supaya physics nya tidak ngebug.

```
private Rigidbody2D rig;

private void Start()
{
   rig = GetComponent<Rigidbody2D>();
}

private void FixedUpdate()
{
   Vector2 velocityVector = rig.velocity;
   velocityVector.x = Mathf.Clamp(velocityVector.x + moveAccel * Time.deltaTime, 0.0f,
maxSpeed);

   rig.velocity = velocityVector;
}
```

Using Physics Material

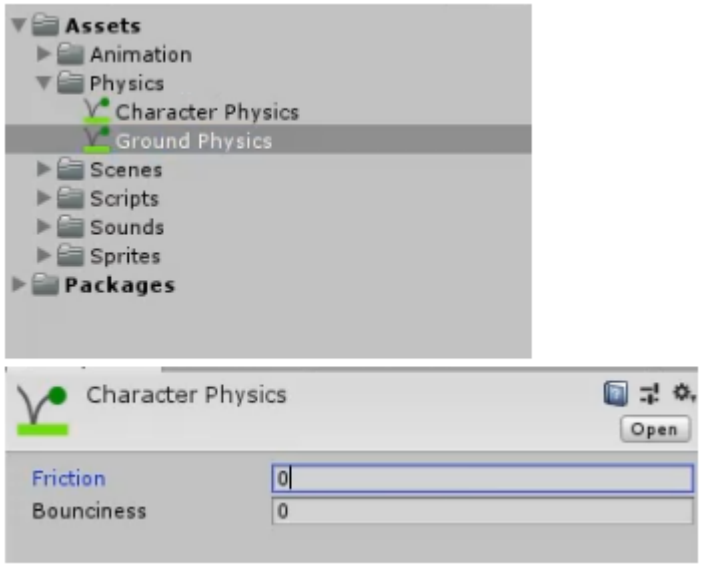
Secara default Unity akan memperlambat kecepatan rigidbody jika bergesekan dengan collider. Hal ini tentu tidak cocok untuk game kita dimana player akan selalu bergesekan dengan collider saat dia berlari. Untuk menanggulangi hal ini kita harus menambahkan Physics material dengan cara:

- Pada Project Window buat folder Physics

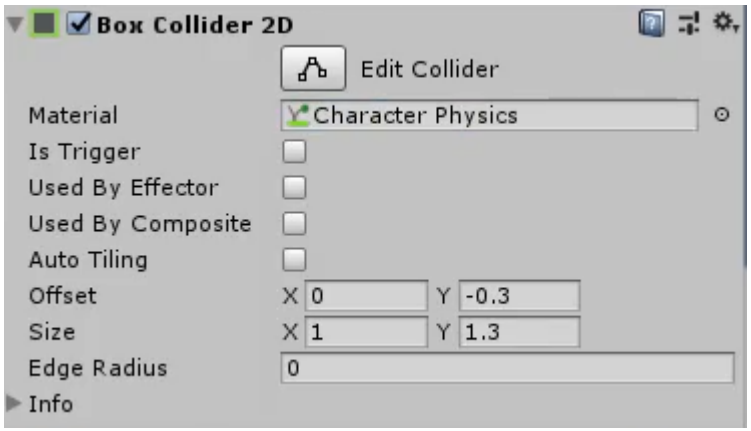
- Klik kanan di dalam folder tersebut

- Pilih Create, lalu pilih Physics Material 2D

Untuk kasus ini kita akan membuat dua Physics Material, satu untuk player dan satu lagi untuk Terrain. Untuk menghapus perlambatan saat bergesekan kita cukup set friction pada Physics Material kita menjadi nol



Kemudian kita pasang Physics Material tadi pada rigidbody dan collider masing-masing object yang kita buat.



Setelah itu, jika kalian mencoba untuk play projectnya, karakter akan berjalan terus ke kanan

6. Chapter 2 - Camera Follow

Camera Follow

Karena player sudah bisa bergerak, sekarang kita perlu menggerakan kamera ke agar mengikuti gerakan player, untuk melakukan hal ini kita tinggal mengambil posisi player dan mengkalkulasikannya dengan offset posisi kamera yang kita inginkan.

Kita buat script baru, kita namakan script kita CameraMoveController yang nantinya bertugas untuk menggerakan kamera kita. kita pasang script tersebut pada kamera kita, lalu kita ubah posisi sumbu x kamera kita dengan kalkulasi antara posisi player dan offset yang kita inginkan.

```
[Header("Position")]
public Transform player;
public float horizontalOffset;

private void Update()
{
   Vector3 newPosition = transform.position;
   newPosition.x = player.position.x + horizontalOffset;
   transform.position = newPosition;
}
```

Setelah kita buat script nya kita langsung drag saja pada object kamera pada scene kita. Kemudian kita isi value nya pada inspector. Untuk tipe beberapa tipe custom seperti Transform ini kita bisa langsung drag object yang memiliki komponen dengan tipe tersebut pada field di inspector atau pilih dari list dengan cara klik icon lingkaran pada sebelah kanan field

Setelah itu, jika kalian mencoba untuk play projectnya, kamera akan bergerak secara horizontal mengikuti gerakan karakter

7. Chapter 2 - Jumping

Jumping

Game endless runner platformer seperti ini tentu saja perlu ada hambatan yang perlu dilewati, nah pada game seperti ini biasanya karakter akan melompati hambatan tersebut. Pada scene yang kita buat pun ada sebuah lubang yang mesti dilompati oleh karakter. Kita akan klik kiri pada mouse yang juga bekerja sebagai tap pada android sebagai input perintah karakter untuk melompat.

Kita tambahkan fungsi Update pada script CharacterMoveController, dan kita tambahkan pembacaan input di dalam fungsi tersebut.

```
[Header("Jump")]
public float jumpAccel;

private bool isJumping;
private bool isOnGround;

private void Update()
{
    // read input
    if (Input.GetMouseButtonDown(0))
    {
        if (isOnGround)
        {
            isJumping = true;
        }
    }
}
```

Karena kita menambahkan input pada Update, kita hanya akan memberikan status isJumping pada karakter kita. Input harus selalu dimasukan pada fungsi Update agar dapat dideteksi oleh game kita. Kalo masih bingung coba lihat penjelasan berikut:

Update :

Fungsi ini dijalankan tiap frame sehingga waktu antar pemanggilannya akan sesuai waktu pemanggilan frame, waktu frame dipanggil tergantung

pada kemampuan hardware yang dipakai atau bisa juga di set fixed di awal seperti pada 60 fps atau 30 fps. Waktu pemanggilan bisa tidak sama tergantung lamanya proses yang dilakukan pada frame tersebut, contoh: pada awal hanya ada 1 musuh sehingga framerate stabil di 60 fps, akan tetapi setelah 10 menit berlalu mulai bermunculan naga dan para pengikutnya mencapai 100 musuh, banyak VFX dan texture berbeda-beda yang di load menyebabkan game nya mulai ngelag sehingga fps tidak lagi stabil. Disini pemanggilan Update juga akan semakin melambat.

Fixed Update :

Fungsi ini dijalankan tiap waktu yang fixed. Waktu ini diset sebagai dari physics engine (biasanya pada 0.02 detik). Pemanggilan fungsi ini tidak terpengaruh oleh frame-rate dan akan selalu stabil. Akan tetapi fungsi ini hanya dianjurkan untuk kalkulasi physics saja, dikarenakan fungsi ini akan tetap lambat saat terlalu banyak kalkulasi lain yang dilakukan.

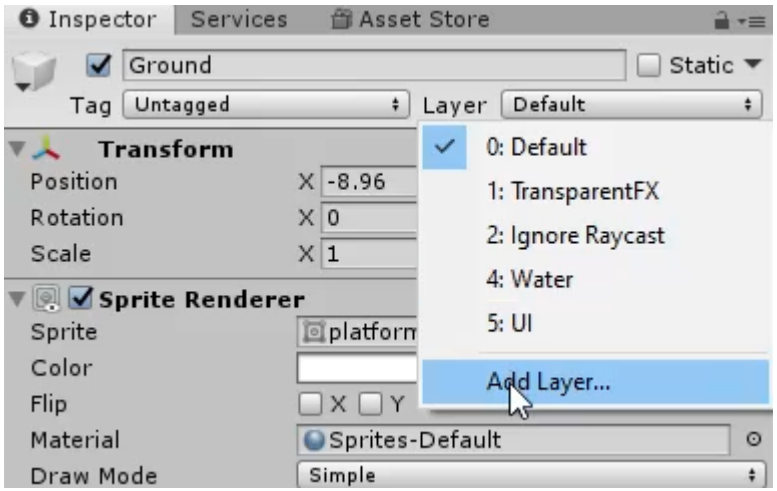
8. Chapter 2 - Using Raycast & Layers

Using Raycast and Layer

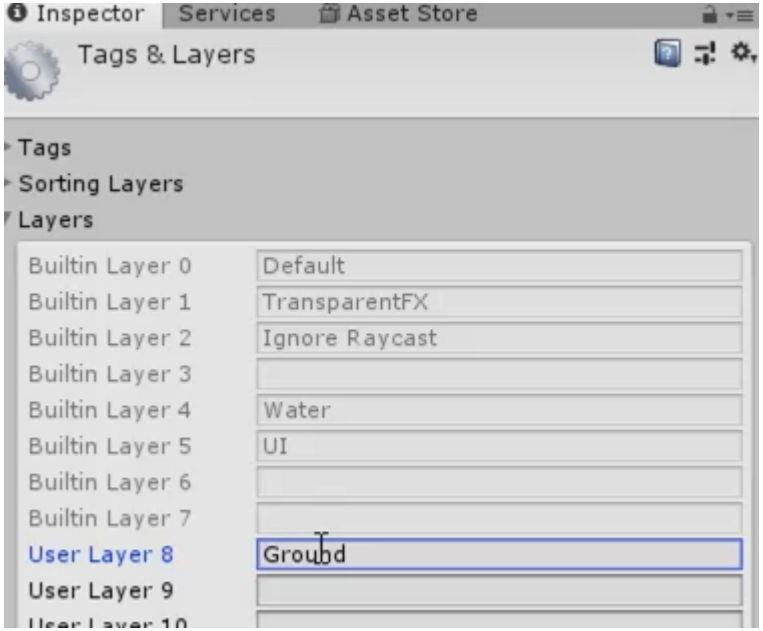
Kembali ke game kita, sekarang kita perlu merealisasikan status isJumping yang kita buat dengan cara menambahkahkan kalkulasi tambahan pada sumbu y untuk perhitungan velocityVector di FixedUpdate. Selain itu untuk membuat karakter kita hanya dapat lompat saat menyentuh terrain, maka kita akan menggunakan status isOnGround. Status ini akan diubah dengan mengecek saat karakter kita menyentuh tanah atau tidak, disini kita akan menggunakan Raycast.

Untuk mempermudah proses raycast kita bisa mendefinisikan layer pada object terrain kita agar raycast hanya memproses collision dengan object pada layer terrain saja. Kita bisa menambahkan layer dengan cara berikut:

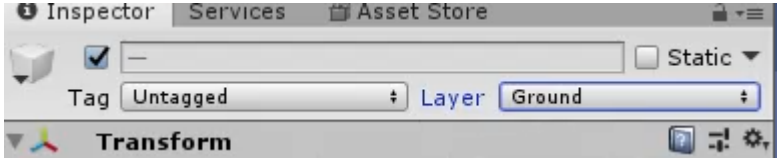
- Klik Object pada scene, kemudian klik Layer
- Pada pilihan layer klik Add Layer…



- Kita isikan Layer baru pada kolom yang kosong, kita namakan Ground yang nantinya akan dipakai oleh semua Terrain yang telah kita buat pada scene



- Lalu kita tinggal memilih object yang ingin kita ubah layernya, yaitu semua ground, lalu kita pilih Layer ground pada menu pemilihan layer



Sekarang kita hanya tinggal membuat script nya. Kita tambahkan jarak raycast dan layer raycast pada inspector agar bisa kita ubah-ubah sesuai keinginan kita. Selain itu, raycast termasuk proses yang menggunakan physics, jadi kita akan menambahkannya pada FixedUpdate. Pada script CharacterMoveController masukan script berikut. Pengingat: perhatikan bahwa di bawah ini terdapat beberapa code yang sudah kita masukkan sebelumnya.

```
[Header("Ground Raycast")]
public float groundRaycastDistance;
public LayerMask groundLayerMask;


private bool isOnGround;


private void FixedUpdate()
{
   // raycast ground
   RaycastHit2D hit = Physics2D.Raycast(transform.position, Vector2.down,
groundRaycastDistance, groundLayerMask);
   if (hit)
   {
      if (!isOnGround && rig.velocity.y <= 0)
      {
         isOnGround = true;
      }
   }
   else
   {
      isOnGround = false;
   }

   // calculate velocity vector
   Vector2 velocityVector = rig.velocity;

   if (isJumping)
   {
      velocityVector.y += jumpAccel;
      isJumping = false;
   }

   velocityVector.x = Mathf.Clamp(velocityVector.x + moveAccel * Time.deltaTime, 0.0f,
maxSpeed);

   rig.velocity = velocityVector;
```

```
}
```

Kita juga akan menambahkan Debug pada Raycast ini supaya mempermudah pengerjaan. Untuk debug berupa visual kita bisa menambahkannya pada fungsi OnDrawGizmos

```
private void OnDrawGizmos()
{
   Debug.DrawLine(transform.position, transform.position + (Vector3.down *
groundRaycastDistance), Color.white);
}
```

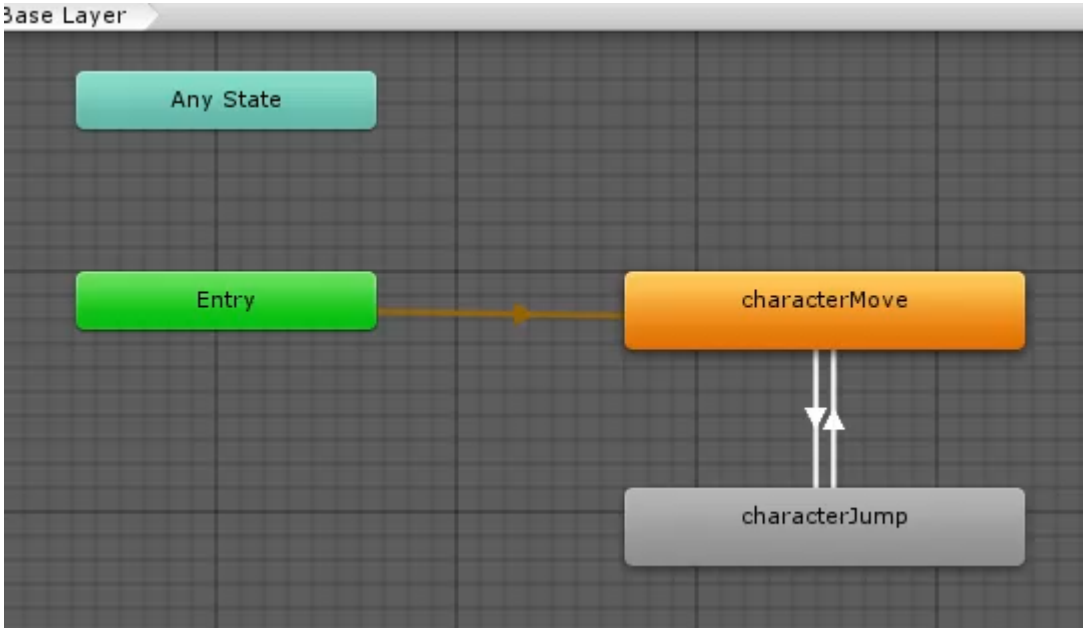
9. Chapter 2 - Adding Jumping Animation

Adding Jumping Animation

Kita bisa menambahkan animasi untuk loncat. Sama seperti sebelumnya, kita tinggal tambahkan animasi baru untuk animasi loncat, lalu kita tambahkan pada animator. Sekarang kita harus menghubungkan animasi loncat dengan animasi move agar proses animasi bisa berpindah dengan cara:

- Klik kanan pada animasi pertama
- Pilih Make Transition
- Klik kiri pada animasi kedua

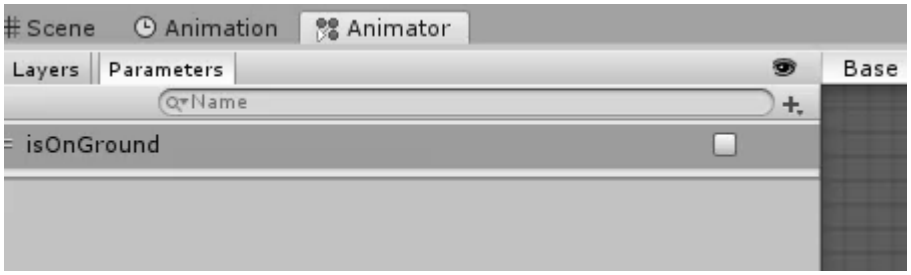
Lakukan untuk proses transisi dari animasi move ke animasi loncat, dan dari animasi loncat ke animasi move sehingga menjadi seperti ini



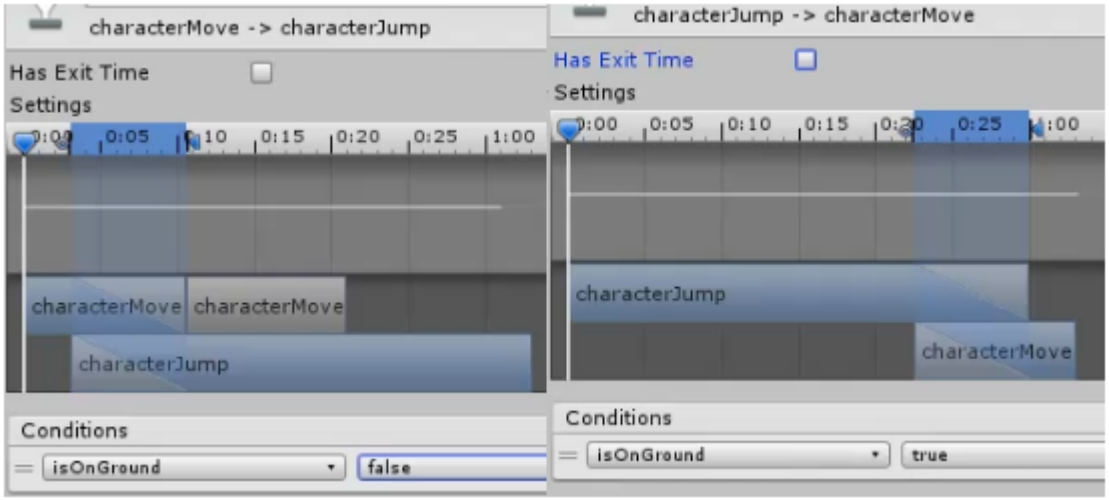
Pada kasus ini kita ingin mengubah animasi dari move ke jump saat karakter tidak menyentuh tanah dan juga sebaliknya, kita ingin mengubah animasi jump ke move saat karakter menyentuh tanah. Kita bisa menggunakan variabel pada animator untuk mengatur proses transisinya dengan cara:

- Klik parameter di pilihan tab pada animator

- Klik tanda plus (+) di sebelah kanan lalu pilih tipe parameter yang dibutuhkan, untuk kasus ini kita memerlukan parameter bool
- Namakan parameter sesuai kebutuhan



Sekarang kita klik garis transisi yang telah kita buat, di inspector akan muncul menu-menu yang bisa kita utak-atik pada transisi ini, untuk sekarang kita hanya akan mematikan exit time agar animasi kita tidak berpindah saat animasinya selesai, lalu kita tambahkan kondisi isOnGround true untuk transisi dari jump ke move dan isOnGround false untuk transisi move ke jump



Untuk animasi Jump nya sendiri kita hanya akan menggunakan 1 sprite, akan tetapi tetap memerlukan 2 keyframe dikarenakan sebuah animasi perlu ada minimal dua keyframe yaitu awal dan akhir. Disini kita akan menggunakan “_1” untuk animasinya.

Sekarang kita hanya perlu menambahkan perintah untuk mengubah parameter animator kita pada script CharacterMoveController. Pertama, kita tambahkan dulu Animator pada script kita, lalu kita bisa menggunakan fungsi pada animator tersebut untuk mengubah value dari parameternya, pada kasus ini kita akan menggunakan Animator.SetBool pada parameter isOnGround.

```
private Animator anim;

private void Start()
{
   rig = GetComponent<Rigidbody2D>();
   anim = GetComponent<Animator>();
}

private void Update()
{
   // read input
   if (Input.GetMouseButtonDown(0))
   {
      if (isOnGround)
      {
         isJumping = true;
      }
   }

   // change animation
   anim.SetBool("isOnGround", isOnGround);
}
```

10. Chapter 2 - Adding Jumping Sound

Adding Jumping Sound

Kita juga akan menambahkan suara pada fitur loncat ini agar terasa lebih menyenangkan. Hal pertama yang harus kita lakukan adalah menambahkan komponen Audio Source pada player kita agar player kita dapat mengeluarkan suara. Kita juga perlu untuk mematikan opsi Play On Awake agar Audio Source dimulai dalam keadaan mati. Audio Source ini akan memutar suara pada dunia game kita, dan kita akan dapat mendengarnya dengan menggunakan Audio Listener yang ada pada kamera kita. Tambahkan komponen audio source pada objek karakter. Caranya klik tab hierarchy > klik objek karakter > lihat inspector > klik add component > search "audio source"

Sekarang saatnya membuat script. Agar lebih terstruktur, kita akan membuat script baru yang bertugas untuk mengatur suara player, kita namakan CharacterSoundController lalu kita pasang pada karakter kita. Isi dari script ini cukup simpel, kita hanya menambahkan variabel pada inspector untuk menyimpan suara loncat. Kemudian kita tambahkan audioSource kita dan kita tambahkan fungsi public yang akan memutar suara loncat tersebut

```
public AudioClip jump;

private AudioSource audioPlayer;

private void Start()
{
   audioPlayer = GetComponent<AudioSource>();
}

public void PlayJump()
{
   audioPlayer.PlayOneShot(jump);
}
```

Lalu pada fungsi Update berbarengan dengan perubahan status isJumping kita tambahkan fungsi untuk memutar suara loncat. Pada script Character Move Controller, tambahkan script berikut:

```
private CharacterSoundController sound;


private void Start()
{
   rig = GetComponent<Rigidbody2D>();
   anim = GetComponent<Animator>();
   sound = GetComponent<CharacterSoundController>();
}

private void Update()
{
   // read input
   if (Input.GetMouseButtonDown(0))
   {
      if (isOnGround)
      {
         isJumping = true;

         sound.PlayJump();
      }
   }

   // change animation
   anim.SetBool("isOnGround", isOnGround);
}
```

Tambahkan komponen script Character Sound Controller pada objek karakter lewat inspector. Lalu, ubah parameter Jump pada komponen Character Sound Controller dengan file audio phaserUp2 dari folder Audio.

Lalu, jangan lupa untuk mengubah parameter Jump Accel, Ground Raycast di inspector objek karakter

Sekarang kalian dapat mencoba hasil penambahan fitur gerak dan loncat yang kalian sudah buat. Keren bukan!!

11. Chapter 3 - Making Terrain Template

Game Endless runner tentu perlu adanya Terrain yang tidak akan habis sampai karakter kalah. Untuk melakukan hal ini kita akan membuat fitur yang akan kita namakan Infinite Terrain. Fitur ini kan secara otomatis memunculkan terrain yang baru saat player mendekat pada ujung terrain terakhir dan juga akan otomatis menghapus terrain yang sudah dilewati karakter.

Objek ground yang kita gunakan pada materi sejauh ini berfungsi untuk mengetes kontrol terhadap karakternya. Sekarang, sebelum masuk ke dalam pembuatan infinite terrain, hapus terlebih dulu seluruh objek ground pada tab hierarchy.

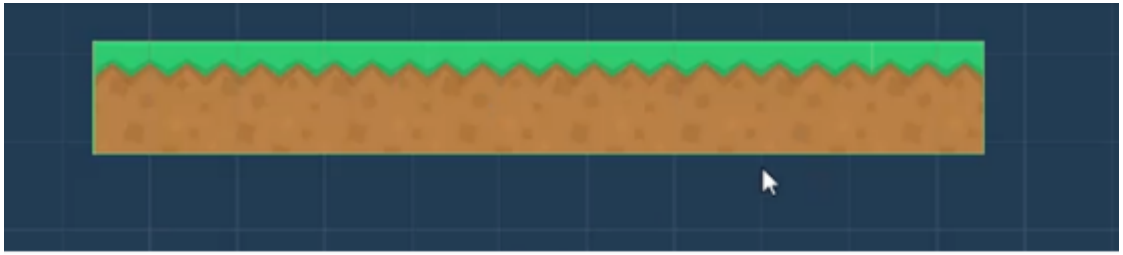
Making Terrain template (prefab)

Hal pertama yang kita lakukan adalah membuat prefab sebagai Terrain Template yang nantinya akan dimunculkan pada posisi berikutnya. Hal ini bertujuan untuk lebih mengontrol terrain yang muncul agar tidak memiliki lubang yang terlalu panjang yang membuat player tidak dapat menghindarinya.

Disini kita hanya akan membuat tiga tipe terrain, yaitu tanpa lubang, dengan lubang kecil, dan dengan lubang besar.

Untuk melakukannya, pertama-tama buat folder prefabs pada tab project. Untuk membuat prefabs berikut caranya: masukkan 1 sprite dari tab project ke scene. Lalu, dari tab hierarchy tarik objek tersebut ke dalam folder prefab di tab project. Setelah itu, icon objek tersebut akan menjadi warna biru. Itu adalah tanda bahwa objek tersebut sudah menjadi prefab. Untuk mengedit prefab tersebut, klik tanda panah yang ada di bagian kanan kolom prefab tersebut.

Untuk membuat varian prefab, berikut caranya: pada tab project, masuk ke dalam folder prefab. Klik kanan prefab yang ingin kamu buat variasinya > creates > prefab variant



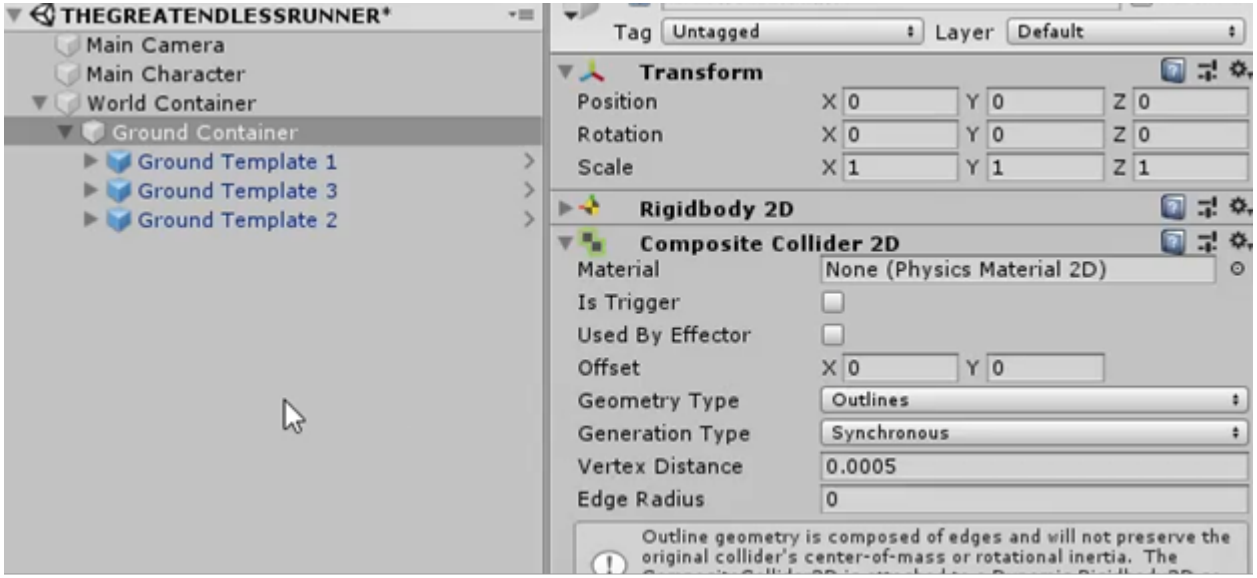
Prefab ini nantinya akan di spawn secara random sehingga area di depan karakter tidak monoton dan tidak membosankan.

12. Chapter 3 - Using Composite Collider

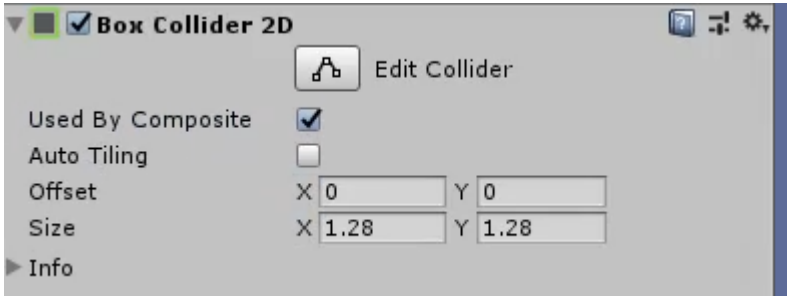
Using Composite Collider

Penggunakan collider standar untuk tiap tile memang sebenarnya cukup untuk menahan player agar tidak menembus terrain, akan tetapi karena tiap tile memiliki collider yang berbeda ada kemungkinan dapat mengakibatkan kalkulasi yang tidak akurat karena pada kasus ini beberapa tile tersebut adalah satu kesatuan sehingga seharusnya hanya memiliki satu collider untuk seluruh tile

Untuk mencapai hal ini kita bisa menggunakan Composite Collider. Composite Collider akan secara otomatis menyatukan semua collider yang ada pada child nya. Kita hanya tinggal membuat kontainer terrain yang nantinya akan ditambahkan komponen Composite Collider ini dan juga bekerja sebagai parent dari semua terrain yang di generate. Jangan lupa untuk mengubah layer menjadi Ground agar terdeteksi raycast isOnGround (jangan ikuti gambar)



Kemudian pada tiap tile di template yang telah kita buat, pada collidernya kita aktifkan Used By Composite agar nantinya dapat dikonversi menjadi Composite Collider


Box Collider 2D
Edit Collider
Used By Composite
Auto Tiling
Offset
X 0
Y 0
Size
X 1.28
Y 1.28
Info

13. Chapter 3 - Using area for auto terrain generator

Using area for auto terrain generator

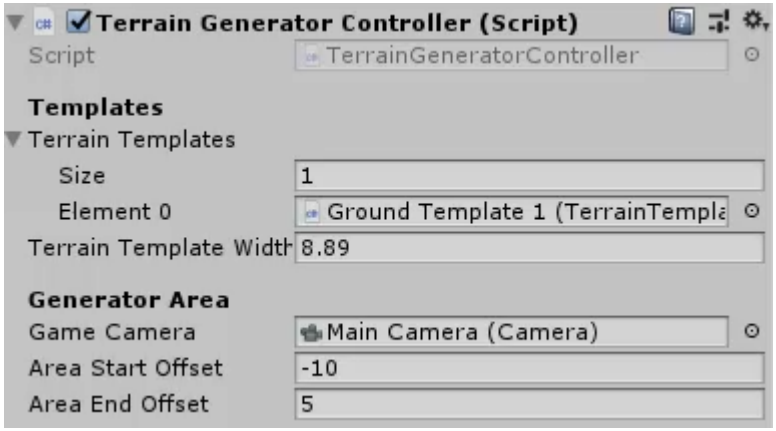
Ada setidaknya dua cara yang bisa digunakan untuk membuat terrain dalam game ini, yakni auto terrain generator dan object pooling. Pada bagian ini kita akan mencoba terlebih dahulu cara auto terrain generator. Barulah setelah itu kita akan menggunakan metode yang lebih efisien, yakni object pooling. Hal ini dilakukan agar pembaca dapat melihat perbedaan kedua metode ini dan efisiensinya.

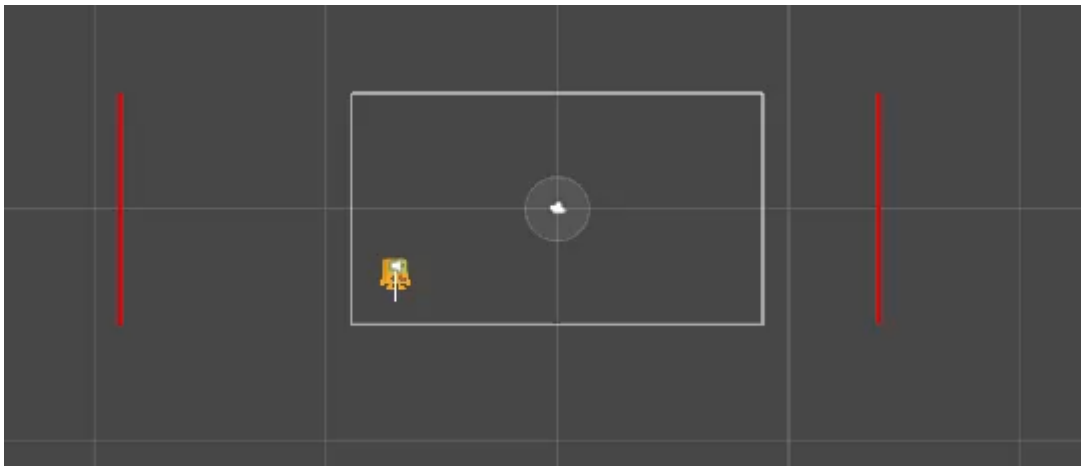
Nah sekarang kita akan membuat proses auto generator untuk template yang telah kita buat. Disini kita akan membuat dua script, yaitu TerrainTemplateController yang kita pasang pada Ground Template dan TerrainGeneratorController yang kita pasang pada Ground Container.

Pada TerrainTemplateController kita hanya akan melakukan proses debug visual pada editor untuk menandai batas dari terrain. Berikut isi debug pada TerrainTemplateController

```
private const float debugLineHeight = 10.0f;

private void OnDrawGizmos()
{
   Debug.DrawLine(transform.position + Vector3.up * debugLineHeight / 2, transform.position
+ Vector3.down * debugLineHeight / 2, Color.green);
}
```

Kemudian pada TerrainGeneratorController kita akan menambahkan menu pada editor untuk list template kita dan juga lebar dari template tersebut. Kita juga akan menggunakan kamera sebagai generator area kita agar lebih mudah dengan ditambahkan offset baik di depan maupun di belakang sehingga menjadi seperti ini


Terrain Generator Controller (Script)
Script
TerrainGeneratorController
Templates
Terrain Templates
Size
1
Element 0
Ground Template 1 (TerrainTempla
Terrain Template Width
8.89
Generator Area
Game Camera
Main Camera (Camera)
Area Start Offset
-10
Area End Offset
5



```
[Header("Templates")]
public List<TerrainTemplateController> terrainTemplates;
public float terrainTemplateWidth;

[Header("Generator Area")]
public Camera gameCamera;
public float areaStartOffset;
public float areaEndOffset;

private const float debugLineHeight = 10.0f;

private float GetHorizontalPositionStart()
{
   return gameCamera.ViewportToWorldPoint(new Vector2(0f, 0f)).x + areaStartOffset;
}

private float GetHorizontalPositionEnd()
{
   return gameCamera.ViewportToWorldPoint(new Vector2(1f, 0f)).x + areaEndOffset;
}

// debug
private void OnDrawGizmos()
{
   Vector3 areaStartPosition = transform.position;
   Vector3 areaEndPosition = transform.position;

   areaStartPosition.x = GetHorizontalPositionStart();
   areaEndPosition.x = GetHorizontalPositionEnd();

   Debug.DrawLine(areaStartPosition + Vector3.up * debugLineHeight / 2, areaStartPosition +
Vector3.down * debugLineHeight / 2, Color.red);
   Debug.DrawLine(areaEndPosition + Vector3.up * debugLineHeight / 2, areaEndPosition +
Vector3.down * debugLineHeight / 2, Color.red);
}
```

14. Chapter 3 - Generate & Delete Object

Generate object

Sekarang kita akan mulai untuk memunculkan Terrain Template pada game kita, untuk di awal game, kita perlu memunculkan terrain dari batas awal sampai batas akhir. Jadi kita akan menambahkan pemanggilan fungsi GenerateTerrain pada Start. Fungsi GenerateTerrain sendiri hanya memunculkan terrain secara acak pada posisi sumbu x tertentu menjadi child dari object yang memunculkannya.

Kita juga perlu menyimpan posisi sumbu x terrain terakhir yang di generate sebagai penanda tempat muncul terrain berikutnya dengan cara menambahkannya dengan lebar terrain, juga sebagai pembatas terrain yang digenerate agar tidak melebihi batas akhir area.

Pada skrip Terrain Generator Controller masukkan script berikut:

```
private List<GameObject> spawnedTerrain;

private float lastGeneratedPositionX;

private void Start()
{
   spawnedTerrain = new List<GameObject>();

   lastGeneratedPositionX = GetHorizontalPositionStart();

   while (lastGeneratedPositionX < GetHorizontalPositionEnd())
   {
      GenerateTerrain(lastGeneratedPositionX, terrain);
      lastGeneratedPositionX += terrainTemplateWidth;
   }
}

private void GenerateTerrain(float posX, TerrainTemplateController forceterrain = null)
{
   GameObject newTerrain = Instantiate(terrainTemplates[Random.Range(0,
terrainTemplates.Count)].gameObject, transform);

   newTerrain.transform.position = new Vector2(posX, 0f);

   spawnedTerrain.Add(newTerrain);
}
```

Sekarang kita juga akan melakukan generate pada fungsi Update dengan cara yang sama pada Start agar terrain muncul tiap kali posisi x terrain terakhir yang digenerate kurang dari batas akhir area

```
private void Update()
{
   while (lastGeneratedPositionX < GetHorizontalPositionEnd())
   {
      GenerateTerrain(lastGeneratedPositionX);
      lastGeneratedPositionX += terrainTemplateWidth;
   }
}
```

Kalau kita mencoba untuk play sekarang, kita mungkin akan menemukan kasus dimana langsung muncul lubang permainan baru dimulai. Untuk mengatasi hal ini kita perlu menambahkan fitur untuk menspawn template spesifik yang muncul di awal permainan.

```
[Header("Force Early Template")]
public List<TerrainTemplateController> earlyTerrainTemplates;
```

```
private void Start()
{
   spawnedTerrain = new List<GameObject>();

   lastGeneratedPositionX = GetHorizontalPositionStart();

   foreach (TerrainTemplateController terrain in earlyTerrainTemplates)
   {
      GenerateTerrain(lastGeneratedPositionX, terrain);
      lastGeneratedPositionX += terrainTemplateWidth;
   }

   while (lastGeneratedPositionX < GetHorizontalPositionEnd())
   {
      GenerateTerrain(lastGeneratedPositionX);
      lastGeneratedPositionX += terrainTemplateWidth;
   }
}
```

Delete object

Kita sudah bisa menambahkan terrain, sekarang kita perlu untuk menghapus terrain yang telah melewati batas awal area. Caranya mirip dengan proses generate, kita memerlukan posisi sumbu x terrain terakhir yang dihapus sebagai penanda lalu kita tambahkan RemoveTerrain pada Update jika posisi sumbu x terrain sudah melewati batas awal area.

Kemudian untuk RemoveTerrain sendiri, cari dulu terrain yang ada pada batas awal area. Lalu kita hapus terrain tersebut

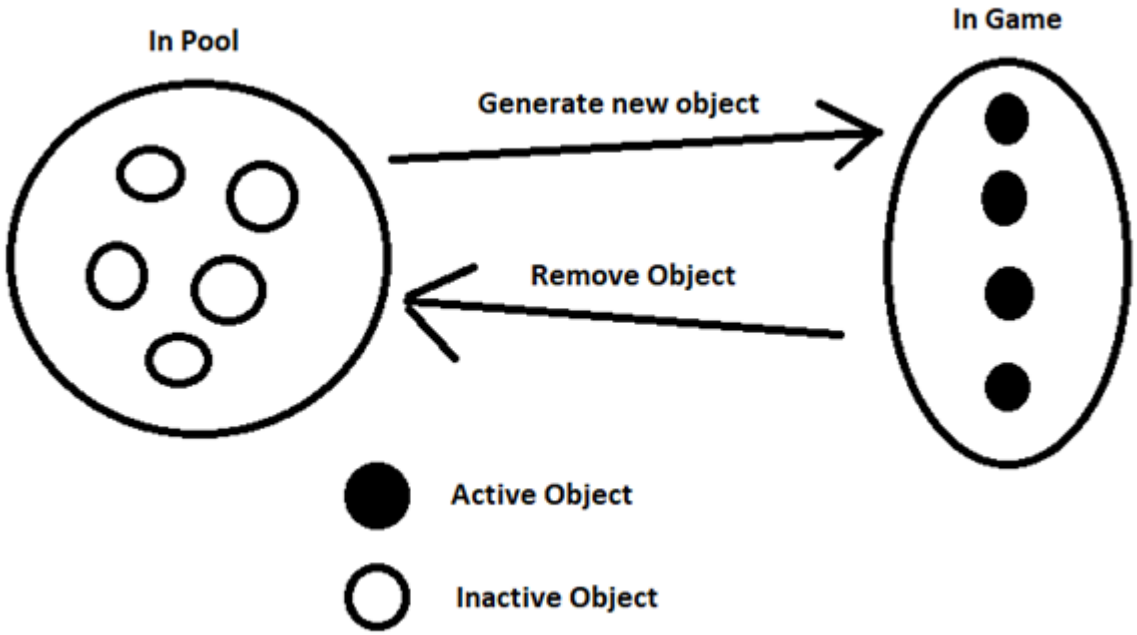
```
    private float lastRemovedPositionX;

    private void Start()
    {
        spawnedTerrain = new List<GameObject>();

        lastGeneratedPositionX = GetHorizontalPositionStart();
        lastRemovedPositionX = lastGeneratedPositionX - terrainTemplateWidth;


        foreach (TerrainTemplateController terrain in earlyTerrainTemplates)
        {
            GenerateTerrain(lastGeneratedPositionX, terrain);
            lastGeneratedPositionX += terrainTemplateWidth;
        }


        while (lastGeneratedPositionX < GetHorizontalPositionEnd())
        {
            GenerateTerrain(lastGeneratedPositionX);
            lastGeneratedPositionX += terrainTemplateWidth;
        }
    }

    private void Update()
    {
        while (lastGeneratedPositionX < GetHorizontalPositionEnd())
        {
            GenerateTerrain(lastGeneratedPositionX);
            lastGeneratedPositionX += terrainTemplateWidth;
        }

        while (lastRemovedPositionX + terrainTemplateWidth < GetHorizontalPositionStart())
        {
            lastRemovedPositionX += terrainTemplateWidth;
            RemoveTerrain(lastRemovedPositionX);
        }
    }
```

```
private void RemoveTerrain(float posX)
{
    GameObject terrainToRemove = null;

    // find terrain at posX
    foreach (GameObject item in spawnedTerrain)
    {
        if (item.transform.position.x == posX)
        {
            terrainToRemove = item;
            break;
        }
    }



    // after found;
    if (terrainToRemove != null)
    {
        spawnedTerrain.Remove(terrainToRemove);
        Destroy(terrainToRemove);
    }
}
```

15. Chapter 3 - Better Method: Object Pooling

Better method: Using object pooling

Penggunaan Instantiate dan Destroy secara berlebihan dapat mengurangi performa game yang dibuat. Oleh karena itu, kita perlu menggunakan teknik yang dinamakan Object Pooling. Object pooling akan mematikan game object daripada menghancurkannya, lalu akan menyalakannya kembali dengan properties yang baru jika kita membutuhkannya. Berikut skema object pooling agar lebih mudah dipahami



Untuk membuat object pooling kita hanya perlu menambahkan Dictionary yang menyimpan pool masing2 object. Untuk proses generate kita akan mengambil object yang ada pada pool. Jika tidak tersedia, maka pool akan membuatkan object baru untuk kita dan pool akan mengaktifkannya agar muncul pada game. Untuk proses remove kita hanya akan mengembalikan object kita pada pool dan menonaktifkannya.

Code dimasukkan pada script Template Generator Controller:

```
// pool list
private Dictionary<string, List<GameObject>> pool;

private void Start()
{
    // init pool
    pool = new Dictionary<string, List<GameObject>>();

    spawnedTerrain = new List<GameObject>();

    lastGeneratedPositionX = GetHorizontalPositionStart();
    lastRemovedPositionX = lastGeneratedPositionX - terrainTemplateWidth;


    foreach (TerrainTemplateController terrain in earlyTerrainTemplates)
    {
        GenerateTerrain(lastGeneratedPositionX, terrain);
        lastGeneratedPositionX += terrainTemplateWidth;
    }


    while (lastGeneratedPositionX < GetHorizontalPositionEnd())
    {
        GenerateTerrain(lastGeneratedPositionX);
        lastGeneratedPositionX += terrainTemplateWidth;
    }
}

// pool function
private GameObject GenerateFromPool(GameObject item, Transform parent)
{
    if (pool.ContainsKey(item.name))
    {
        // if item available in pool
        if (pool[item.name].Count > 0)
        {
            GameObject newItemFromPool = pool[item.name][0];
            pool[item.name].Remove(newItemFromPool);
            newItemFromPool.SetActive(true);
```

```
            return newItemFromPool;
        }
    }
    else
    {
        // if item list not defined, create new one
        pool.Add(item.name, new List<GameObject>());
    }


    // create new one if no item available in pool
    GameObject newItem = Instantiate(item, parent);
    newItem.name = item.name;
    return newItem;
}

private void ReturnToPool(GameObject item)
{
    if (!pool.ContainsKey(item.name))
    {
        Debug.LogError("INVALID POOL ITEM!!");
    }


    pool[item.name].Add(item);
    item.SetActive(false);
}
```

Sekarang kita coba mainkan game kita. Yay, sekarang game kita sudah menjadi Endless Runner beneran. Sudah tidak ada lagi area hampa pada game kita ini

16. Chapter 4 - Create Score Data & Score Controller

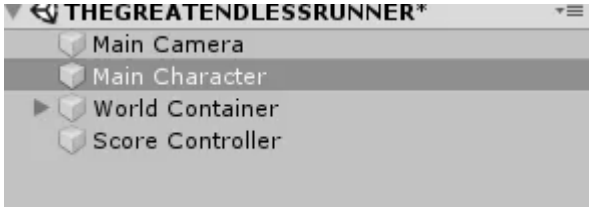
Bukan game namanya kalau kita tidak mendapatkan skor atau achievement saat kita bermain. Maka dari itu, sekarang kita akan menambahkan fitur untuk Scoring dan juga menambahkan fitur untuk GameOver setiap kali player jatuh ke lubang.

Create Score Data and Score Controller

Untuk menyimpan skor tertinggi, kita hanya akan membuat static variable saja untuk memvisualisasikannya. Kita akan simpan skor tertinggi itu pada script baru yang kita namakan ScoreData. Kita ubah script ini menjadi static dan juga menghapus extend dari Monobehaviour karena hanya bertugas untuk menyimpan data dan juga agar bisa diakses oleh semua class. Berikut adalah isi dari script ScoreData:

```
public static class ScoreData
{
   public static int highScore;
}
```

Kemudian kita tambahkan object baru pada scene kita yang bertugas untuk mengatur sistem Scoring, lalu kita buat script baru ScoreController dan kita pasangkan pada object tersebut. ScoreController ini akan bertugas untuk menyimpan skor game yang sedang berjalan, mengatur perubahan skor, dan juga bertugas untuk mengubah highscore pada saat game over.



Ini adalah isi dari script ScoreController:

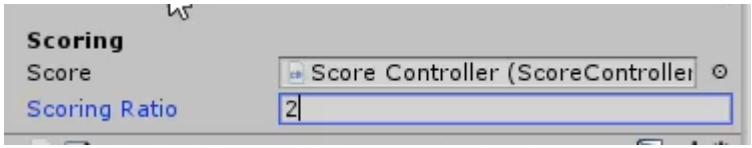
```
    private int currentScore = 0;

    private void Start()
    {
        // reset
        currentScore = 0;
    }

    public float GetCurrentScore()
    {
        return currentScore;
    }

    public void IncreaseCurrentScore(int increment)
    {
        currentScore += increment;
    }

    public void FinishScoring()
    {
        // set high score
        if (currentScore > ScoreData.highScore)
        {
            ScoreData.highScore = currentScore;
        }
    }
```

17. Chapter 4 - Calculating Score

Calculating score

Untuk menambahkan skor kita akan membuat kalkulasi skor berdasarkan jarak yang di tempuh karakter. Kita akan menambahkan proses kalkulasi ini pada fungsi update karena bukan merupakan perhitungan physics.

Kita juga perlu mengambil ScoreController sebelum memproses skor. Karena ScoreController berada di luar karakter, maka kita perlu memasangnya lewat inspector. Selain itu, kita akan menggunakan scoring ratio agar skor yang dihasilkan bisa dikecilkan atau dinaikan sesuai kebutuhan

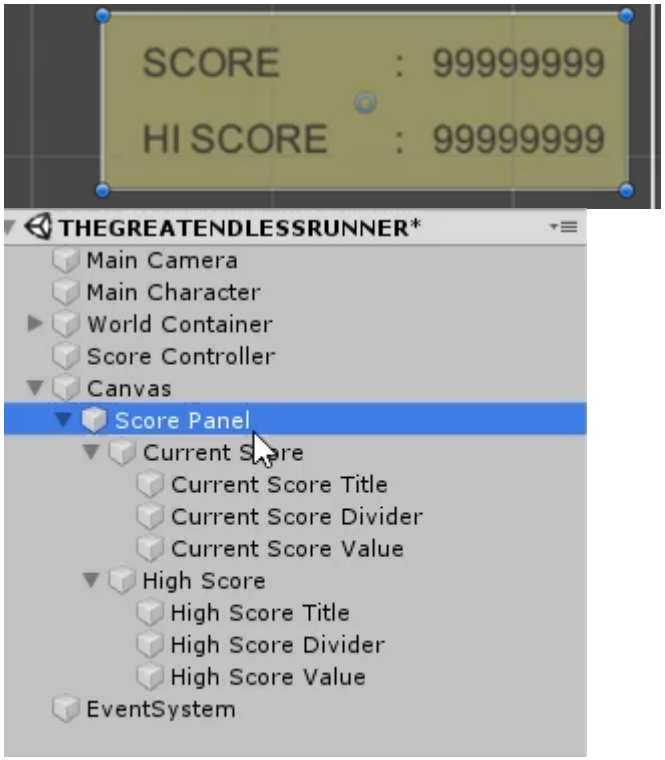


```
[Header("Scoring")]
public ScoreController score;
public float scoringRatio;
private float lastPositionX;

private void Update()
{
    // read input
    if (Input.GetMouseButtonDown(0))
    {
        if (isOnGround)
        {
            isJumping = true;

            sound.PlayJump();
        }
    }

    // change animation
    anim.SetBool("isOnGround", isOnGround);

    // calculate score
    int distancePassed = Mathf.FloorToInt(transform.position.x - lastPositionX);
    int scoreIncrement = Mathf.FloorToInt(distancePassed / scoringRatio);

    if (scoreIncrement > 0)
    {
        score.IncreaseCurrentScore(scoreIncrement);
        lastPositionX += distancePassed;
    }
}
```

18. Create UI to Show Score

Create UI to show Score

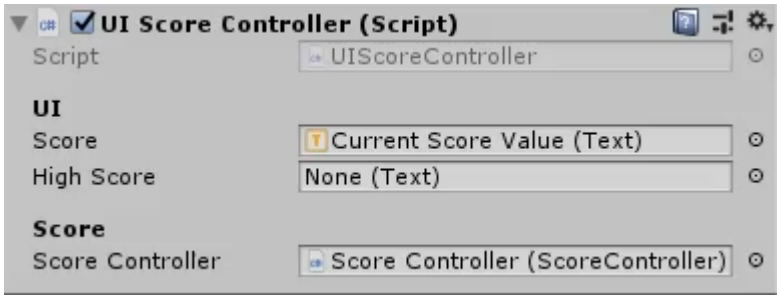
Sistem Scoring sudah selesai, sekarang kita perlu menampilkan skor nya saja. Disini kita akan menggunakan UI Text untuk menampilkan skor dan highscore nya menjadi seperti ini



Sekarang kita akan membuat script baru untuk mengontrol UI skor yang telah kita buat ini dengan nama UIScoreController.

```
[Header("UI")]
public Text score;
public Text highScore;

[Header("Score")]
public ScoreController scoreController;

private void Update()
{
   score.text = scoreController.GetCurrentScore().ToString();
   highScore.text = ScoreData.highScore.ToString();
}
```

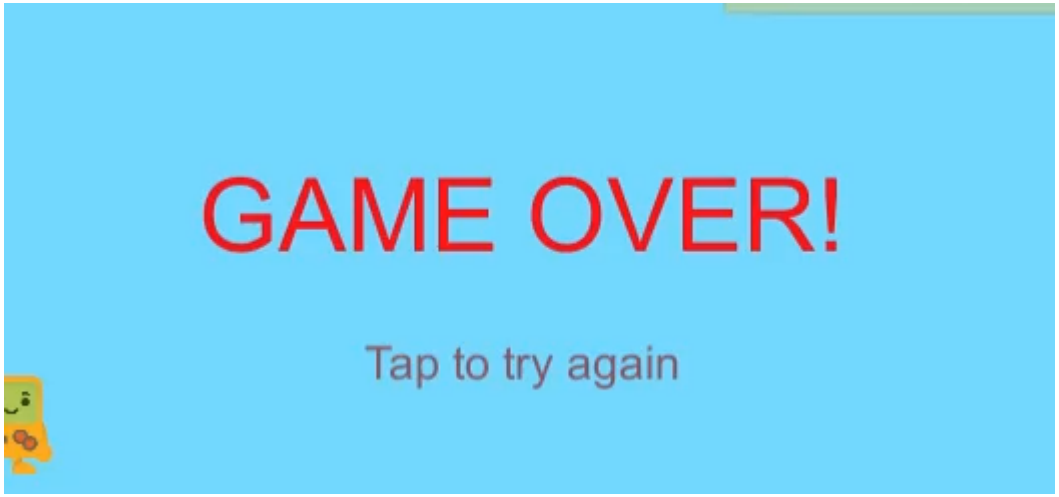
Kita pasangkan pada Score Panel, lalu kita masukan object score dan highscore kita dan masukan juga object score controller.

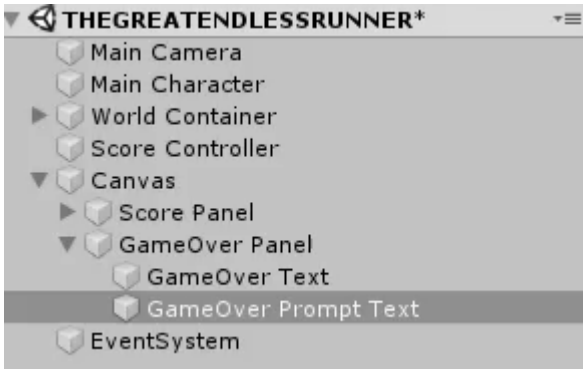


19. Chapter 4 - Adding Game Over Behaviour

Adding Game Over behaviour

Sebuah game perlu ada ending, yah minimal ada game over nya. Sebelum kita membuat fungsi game over, terlebih dahulu kita buat UI game overnya. Kalian bisa menambahkan animasi juga untuk scene game over ini.





Lalu kita buat script UIGameOverController ini yang sekaligus akan mengatur proses restart game saat kita klik atau tap. Nanti script UIGameOverController dimasukkan ke dalam objek GameOver Panel

```
private void Update()
{
   if (Input.GetMouseButtonDown(0))
   {
      // reload
      SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex);
   }
}
```

Untuk game endless runner ini kita akan membuat game over saat karakter nya jatuh ke lubang. Kita bisa membuat ini dengan menambahkan fungsi baru pada CharacterMoveController yang akan berjalan saat karakter kita jatuh ke bawah pada posisi tertentu yang berada dibawah terrain

```
[Header("GameOver")]
public GameObject gameOverScreen;
public float fallPositionY;

[Header("Camera")]
public CameraMoveController gameCamera;


private void Update()
{
    // read input
    if (Input.GetMouseButtonDown(0))
    {
        if (isOnGround)
        {
            isJumping = true;

            sound.PlayJump();
        }
    }

    // change animation
    anim.SetBool("isOnGround", isOnGround);

    // calculate score
    int distancePassed = Mathf.FloorToInt(transform.position.x - lastPositionX);
    int scoreIncrement = Mathf.FloorToInt(distancePassed / scoringRatio);

    if (scoreIncrement > 0)
    {
        score.IncreaseCurrentScore(scoreIncrement);
        lastPositionX += distancePassed;
    }

    // game over
    if (transform.position.y < fallPositionY)
    {
        GameOver();
```

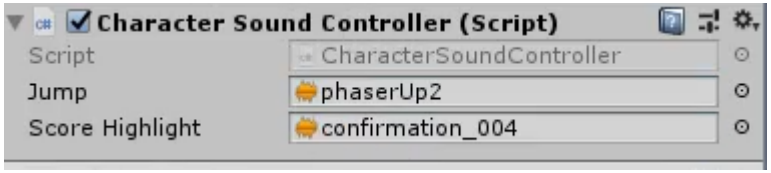
```
    }
}

private void GameOver()
{
    // set high score
    score.FinishScoring();

    // stop camera movement
    gameCamera.enabled = false;

    // show gameover
    gameOverScreen.SetActive(true);

    // disable this too
    this.enabled = false;
}
```

Selesailah proses pembuatan game ini, cobalah kalian mainkan dan rasakan sendiri game yang kalian buat.

20. Chapter 5 - Adding Sound

Adding Sound each x unit range

Pertama, kita tambahkan dulu sound yang akan diputar ke dalam folder sound, lalu kita tambahkan sound kita pada script CharacterSoundController




```
public AudioClip scoreHighlight;

public void PlayScoreHighlight()
{
   audioPlayer.PlayOneShot(scoreHighlight);
}
```

Sekarang kita tambahkan CharacterSoundController kita pada ScoreController karena nantinya ScoreController lah yang akan memutar sound ini. Kita juga tetapkan range yang akan kita gunakan untuk memutar sound.

```
[Header("Score Highlight")]
public int scoreHighlightRange;
public CharacterSoundController sound;

private int lastScoreHighlight = 0;

private void Start()
{
   // reset
   currentScore = 0;
   lastScoreHighlight = 0;
}

public void IncreaseCurrentScore(int increment)
{
   currentScore += increment;

   if (currentScore - lastScoreHighlight > scoreHighlightRange)
   {
      sound.PlayScoreHighlight();
      lastScoreHighlight += scoreHighlightRange;
   }
 }
```