ALIENSHOOT

Progetto Sistemi di realtà virtuale

8 GIUGNO 2020

ALESSANDRO GIONANGELI

MATRICOLA: 305082

ALIENSHOOT

Progetto realizzato in Unity 3D

Per il progetto d'esame di "Sistemi di realtà virtuale" ho deciso di sviluppare un gioco interattivo in 3D interamente costruito con la piattaforma Unity. Il linguaggio utilizzato per il funzionamento è il C#, proprietario del colosso digitale Microsoft.

Il gioco è strutturato secondo lo schema FPS (First Person Shooter) che ci consente di navigare all'interno della mappa e l'obbiettivo è quello di eliminare tutti i nemici, sparando dei proiettili , ed attraversare il portale in fondo alla mappa. Questo ci teletrasporterà al livello successivo. L'intero progetto quindi è strutturato secondo una progressione di scene che va dal livello 1 al livello 5. La difficoltà e il numero di nemici al cambiare della scena sarà superiore. Non è semplice arrivare in fondo e completare il gioco in quanto nel momento in cui si perde si dovrà ricominciare dal primo livello.

"QUANDO SEI AD UN PASSO DALLA VITTORIA ECCO CHE UN PICCOLO ERRORE PUO' RISULTARE FATALE, TANTO DA FARTI TORNARE ALL'INIZIO E RICOMINCIARE DA ZERO"

Detto questo, buon game!

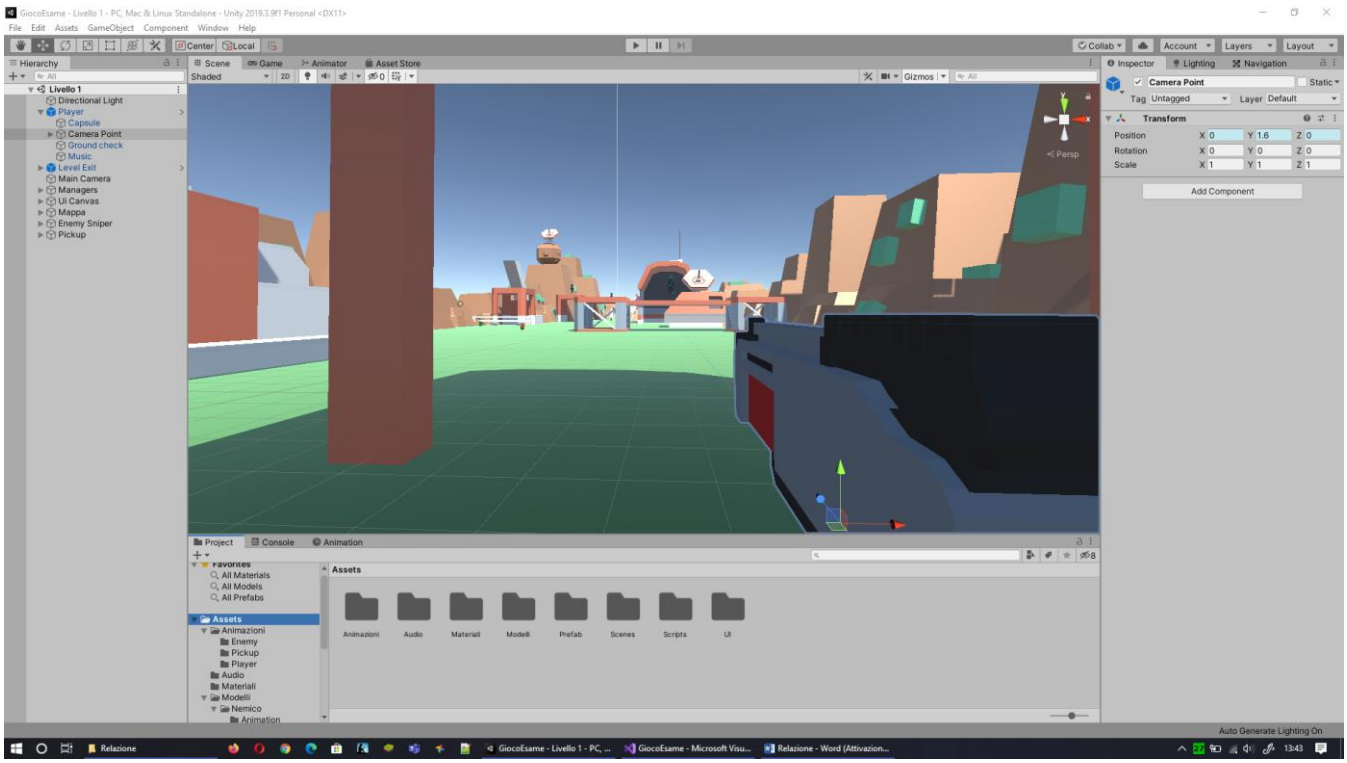
PROCESSI DI SVILUPPO DEL SOFTWARE ALIENSHOOT

Per lo sviluppo del gioco possiamo suddividere il progetto in categorie ed andremo ad analizzarle una per uno. Queste possono essere riassunte in questo modo:

- **SETTAGGI VISUALE CAMERA**
- **PLAYER E MOVIMENTO**
- **ENEMY**
- **ASSET IMPORTATI**
- **PROIETTILI ED EFFETTI**
- **RACCOLTA OGGETTI**
- **ANIMAZIONI**
- **AUDIO**
- **SCENE E STRUTTURA A LIVELLI**
- **BUILD ESEGUIBILE E ALTRE APPLICAZIONI**

- # SETTAGGI VISUALE CAMERA

Come anticipato il progetto è strutturato secondo uno schema FPS dove la visuale di gioco ruota intorno al giocatore in prima persona e quindi per rendere possibile questo è necessario che la camera segua sempre la visuale del player. In questo caso l'unica parte visibile del personaggio è la pistola.



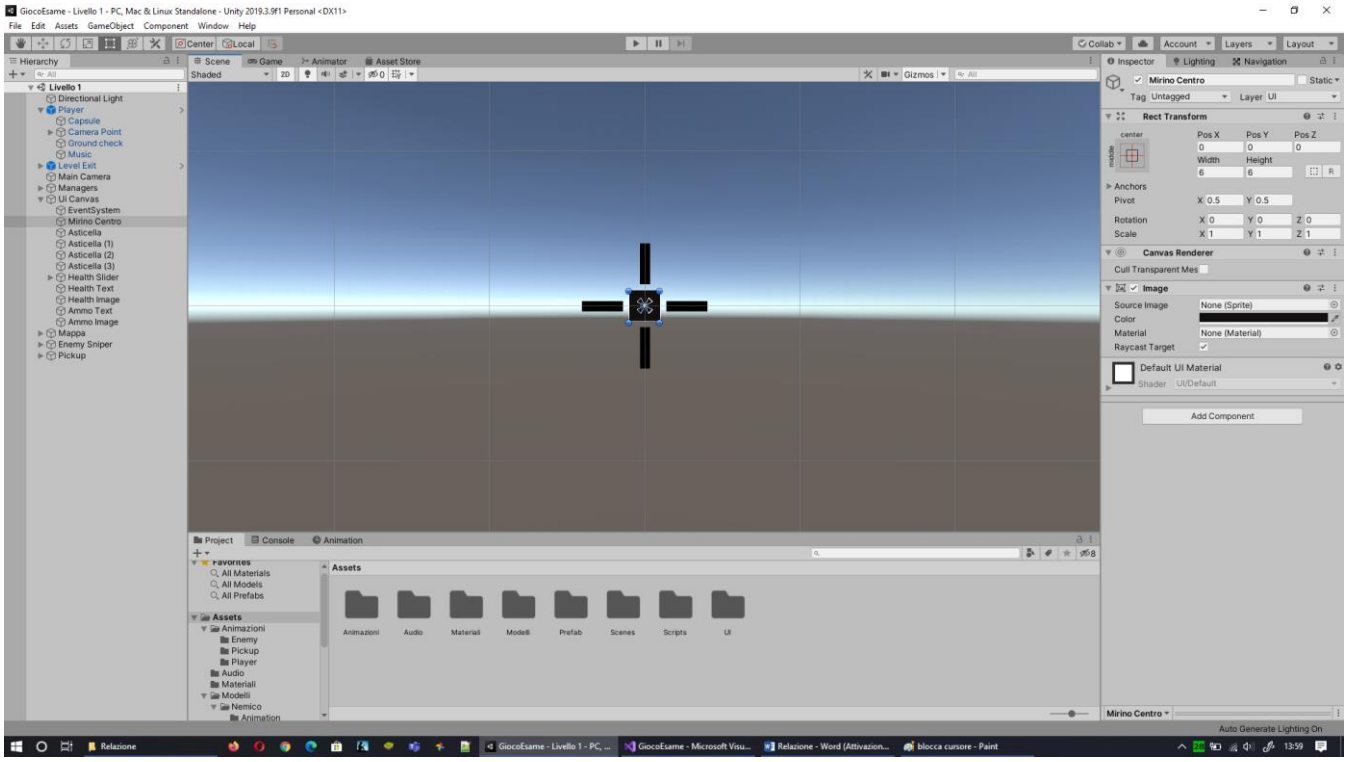
Per rendere funzionale uno gioco in prima persona è necessario che il cursore del mouse punti sempre al centro e che quindi la visuale al movimento ruoti con esso.

Per far questo è necessario impostare uno script che con poche righe di codice lo permette di fare.

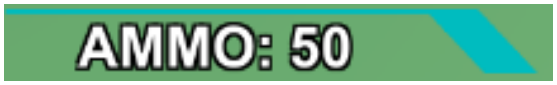
```
Cursor.lockState = CursorLockMode.Locked;
Cursor.visible = false;
```

Una volta fissato però occorre che nel menu principale e nelle varie schermate di passaggio al livello successivo il cursore sia sbloccato. Pertanto esso in questi casi sarà presente.

Al centro dello schermo per rendere più pratica e funzionale la mira sono presenti delle piccole immagini fisse che fungono da mirino.



Oltre a questi sempre fissati nella schermata di gioco, sono presenti anche due Label che permettono di vedere, in maniera aggiornata con un' oppurtuna variabile, la vita e il numero di proiettili disponibili.



Di default ho impostato una vita di 100 punti e 50 munizioni iniziali.

- # PLAYER E MOVIMENTO

Al personaggio è consentito muoversi, grazie alla frecce direzionali e al movimento del mouse, navigando all'interno della mappa in 3D sugli assi X, Y, Z.

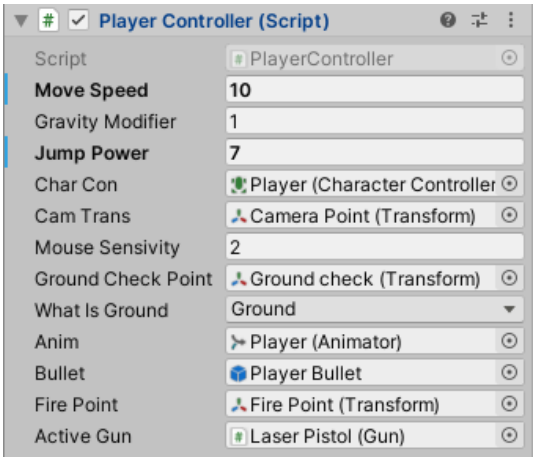
Per la realizzazione di questo è necessario impostare uno script che andrà inserito dentro l'oggetto "player" e il codice in questione è questo:

```
Vector3 vertMove = transform.forward * Input.GetAxis("Vertical");
Vector3 horiMove = transform.right * Input.GetAxis("Horizontal");

moveInput = horiMove + vertMove;
moveInput.Normalize();
moveInput = moveInput * moveSpeed;

moveInput.y = yStore;
moveInput.y += Physics.gravity.y * gravityModifier * Time.deltaTime;

if(CharCon.isGrounded)
{
    moveInput.y = Physics.gravity.y * gravityModifier * Time.deltaTime;
}

canJump = Physics.OverlapSphere(groundCheckPoint.position, 0.25f, whatIsGround).Length > 0;

//jump
if (Input.GetKeyDown(KeyCode.Space) && canJump)
{
    moveInput.y = jumpPower;
}

CharCon.Move(moveInput * Time.deltaTime);

//controllo camera
Vector2 mouseInput = new Vector2(Input.GetAxisRaw("Mouse X"), Input.GetAxisRaw("Mouse Y")) * mouseSensivity;

transform.rotation = Quaternion.Euler(transform.rotation.eulerAngles.x, transform.rotation.eulerAngles.y + mouseInput.x, transform.rotation.eulerAngles.z);
camTrans.rotation = Quaternion.Euler(camTrans.rotation.eulerAngles + new Vector3(-mouseInput.y, 0f, 0f));
```

Per rendere più realistica l'esperienza utente, la navigazione nella mappa è permessa anche solamente premendo la freccia avanti e spostando il cursore per indirizzare.

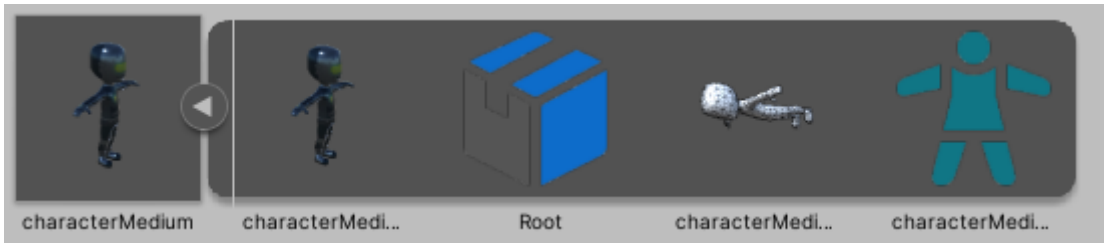
All'interno di queste righe è presente pure un modulo che consente di poter far fare un salto al personaggio usando la barra spaziatrice, che per evitare di creare eccezioni indesiderate questa funzione funziona solamente se si trova sopra un oggetto che ha come layout "ground" che caratterizza tutti gli oggetti posizionati in basse posizioni.

Inoltre in questro script grazie a delle variabili pubbliche opportumente impostate è possibile settare senza intervenire sul codice alcuni parametri importanti di gameplay, come modificare la velocità del personaggio, la sensibilità del mouse.



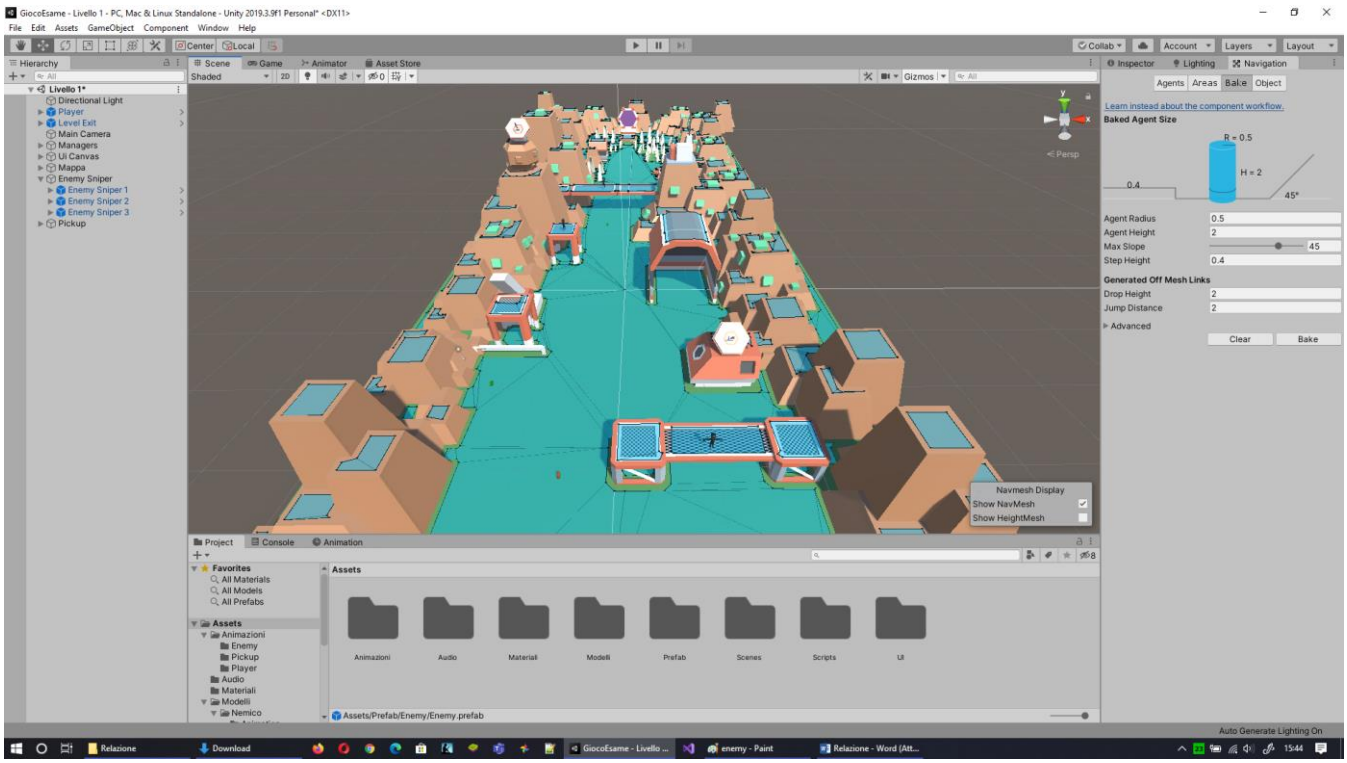
- **PERSONAGGIO ENEMY**

Per completare il gioco è necessasio uccidere tutti i nemici presenti sulla mappa. Per il primo ed il secondo livello è presente solamente un tipo di nemico, che ho chiamato "enemy sniper", esso rimane sopra una struttura e spara da lontano. Dal terzo livello in poi invece vediamo la presenza anche un personaggio che oltre a sparare dalla distanza, gli è consentito di avvicinarsi al player e di inseguirlo.



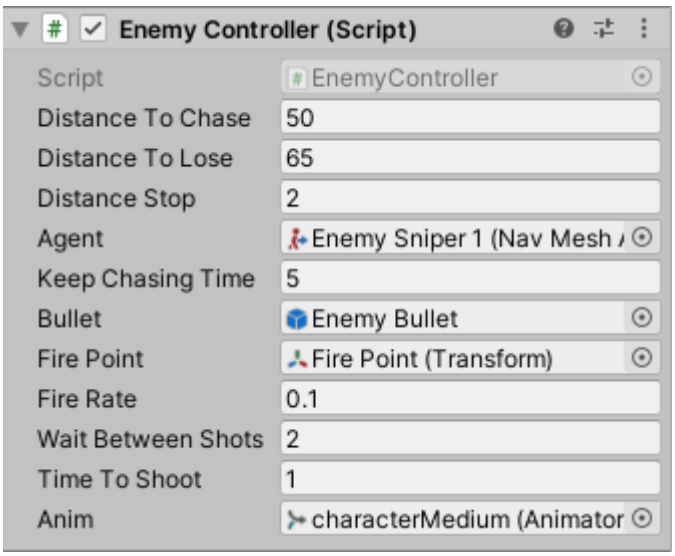
Il personaggio è stato importato da un Asset Store su internet, nella prossima categoria parleremo approfonditamente di questo aspetto.

Il tipo di nemico che è in grado di inseguire il player, per far questo in maniera corretta necessita di un modulo di AI (intelligenza artificiale) che gli permette a seguito di una mappatura della mappa di evitare tutti gli ostacoli e quindi rimanere bloccato, ed eventualmente salire anche sopra oggetti per avvicinarsi il più possibile al target impostato che in questo caso è il player.



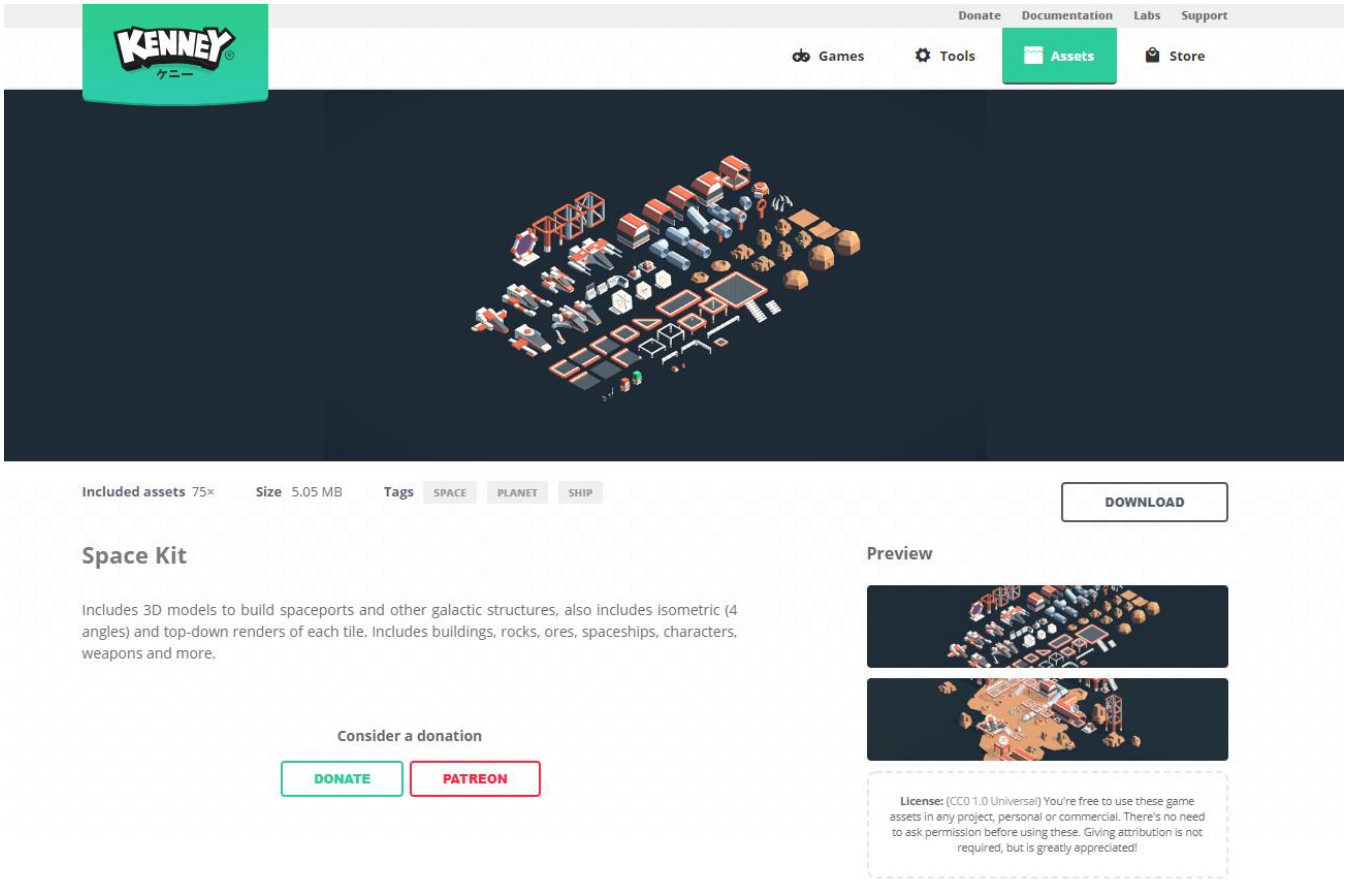
Inoltre grazie ad uno script associato al "character enemy" anche in questo caso è possibile facilmente senza intervenire sul codice cambiare alcuni parametri di forza del nemico.

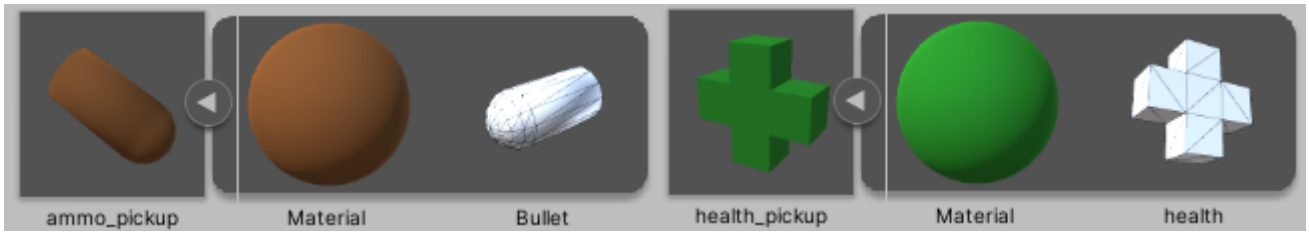
E' impostabile la distanza a cui il nemico ti può vedere e quindi sparare oppure inseguire, la distanza in cui ti perde di vista, il tempo di mantenimento anche se ti ha perso di vista, la velocità di fuoco , la pausa tra uno sparo e l'altro e la durata di uno sparo.

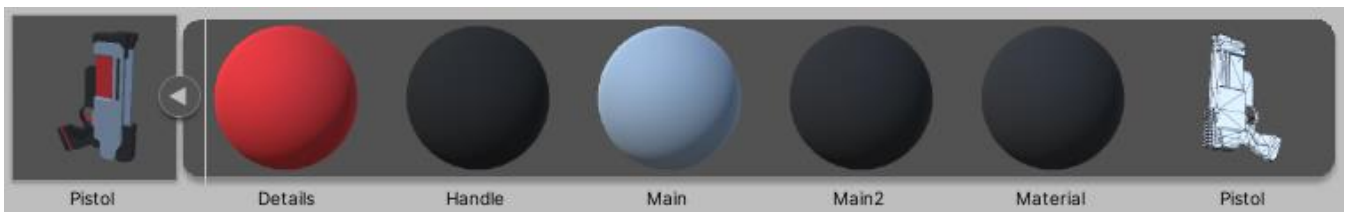


- # ASSET IMPORTATI

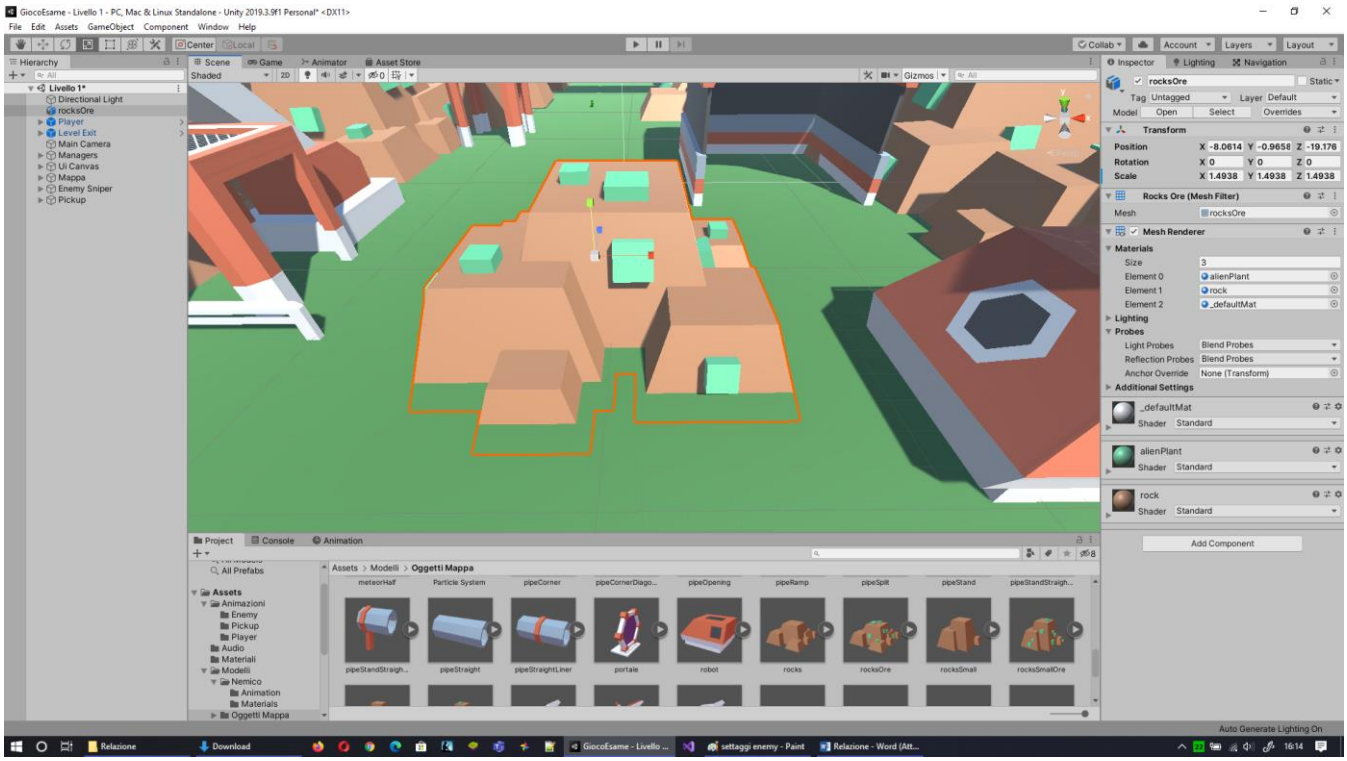
Per la creazione della mappa di gioco mi sono servito di alcuni asset forniti gratuitamente da dei sviluppatori che mi hanno permesso di creare quelle che sono le montagne, le piattaforme, il carattere del nemico, la pistola e gli oggetti che si possono raccogliere a terra





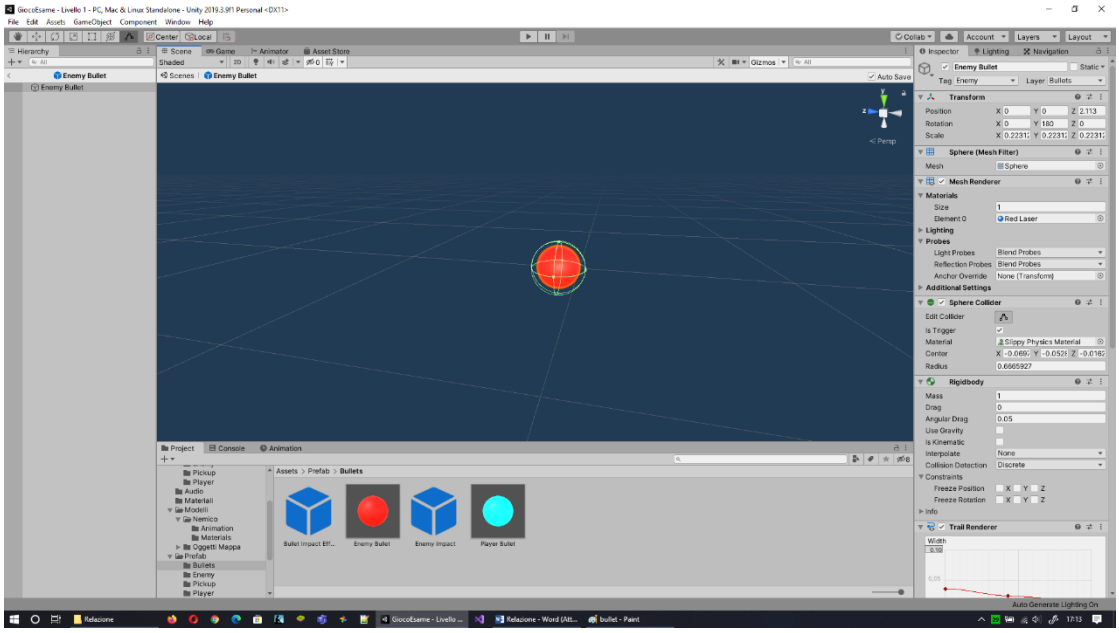



KENNEY
CHARACTER
ASSETS
+ FREE UPDATES
Kenney Character Assets is a bundle that includes multiple character models, skin textures, animations and accessories. The characters are rigged and work with most game engines including Unity and Unreal.
The assets in this bundle are crafted by Kay Lousberg and published by Kenney.
Contents
4 low poly character models
75 skins that fit each of the models
40 themed accessories
17 animations that work with each model
Included animations: Attack, death, idle, interact (ground), interact (standing), jump, kick, punch, racing (idle), racing (steer left), racing (steer right), running and walking, crouch, crouch (idle), crouch (walk), shoot
Features
Free regular updates adding new content
Public domain license (suited for unlimited commercial projects)
Includes source Blender (.blend) files, Unity package and vector files
By purchasing you support the creation of more assets
Preview

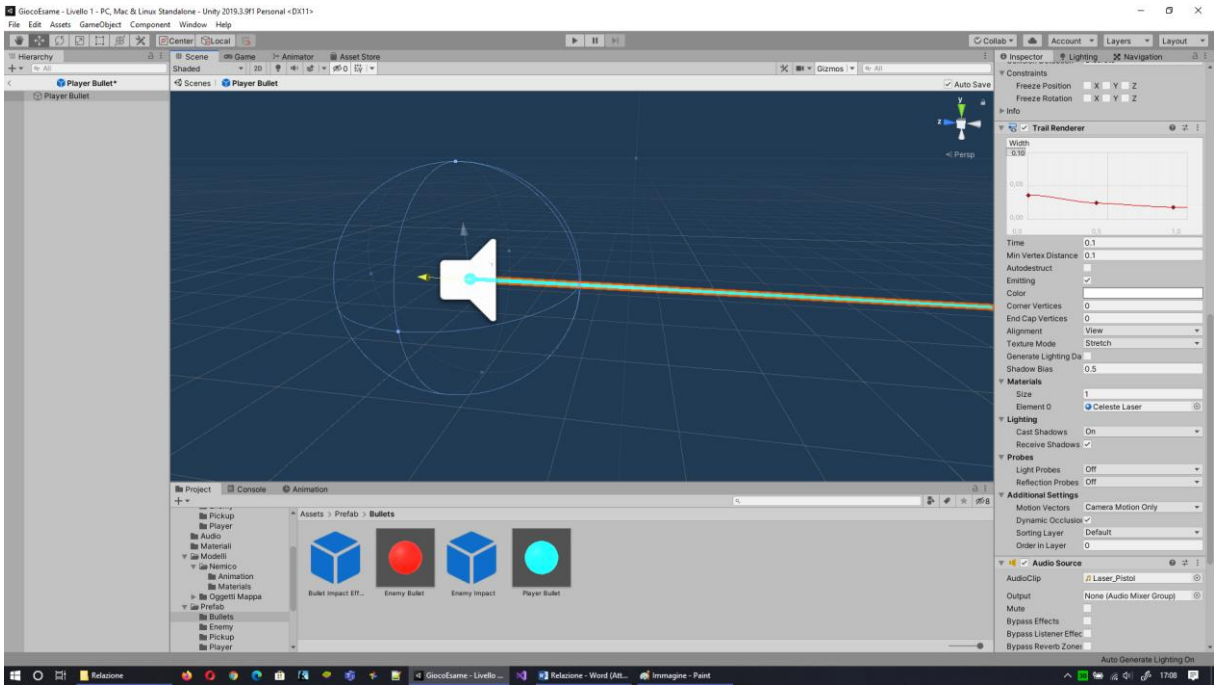


- **PROIETTILI ED EFFETTI**

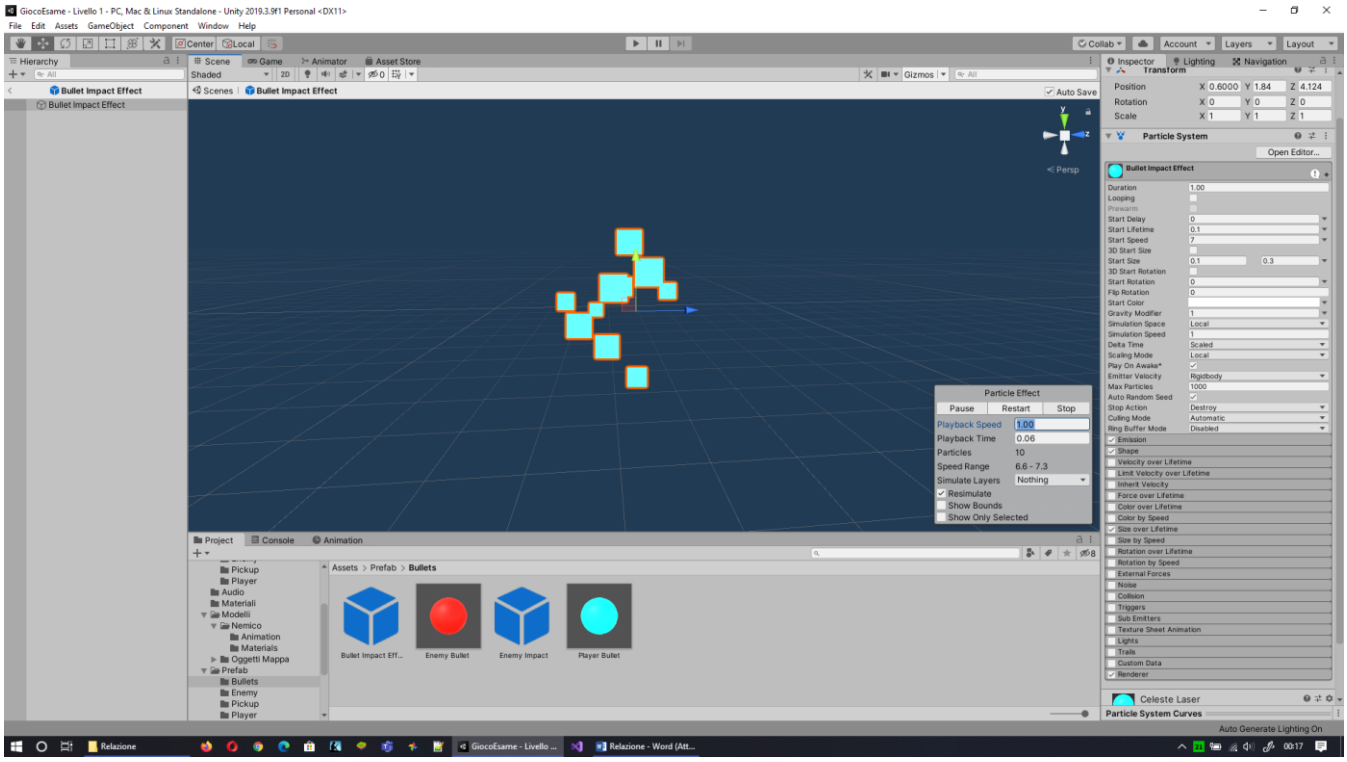
Il player e i nemici, come è facile intuire, per sparare devono servirsi di oggetti. In questo caso possiamo trovare due tipi di proiettili, il rosso è quello che sparano i nemici, mentre il celeste il player. Grazie alle proprietà degli oggetti creati in Unity è possibile impostare il "Collider" che consente di regolare l'ampiezza fisica che va ad impattarsi. Grazie a questa con una funzione che scatena l'evento di collisione è possibile inliggere danni richiamando l'apposita funzione.



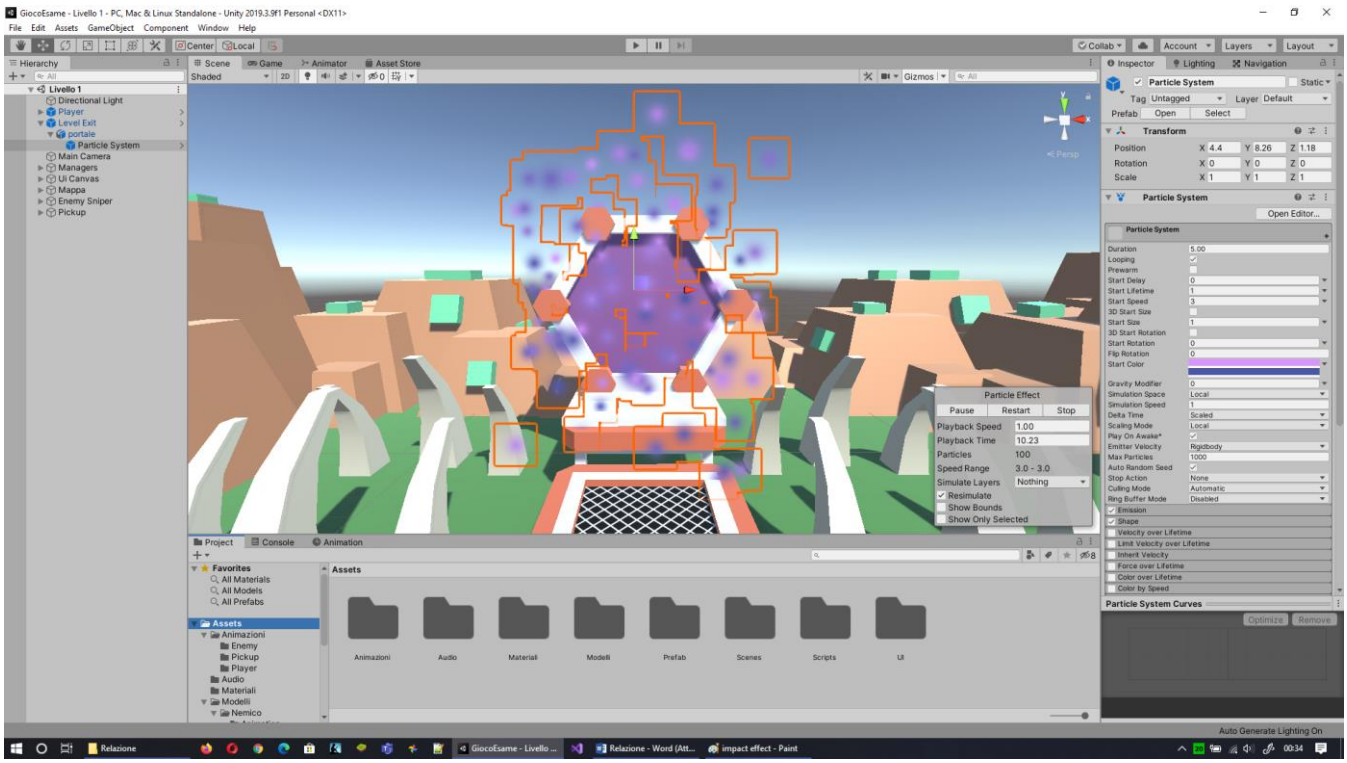
Inoltre per renderelo più accattivante ho inserito una scia che simula l'effetto di un raggio laser. Nonché il suono, ma questo verrà illustrato in una categoria successiva.



Nel momento in cui il proiettile impatta un altro oggetto si genera un effetto di disintegrazione che rende più realistico il colpo.

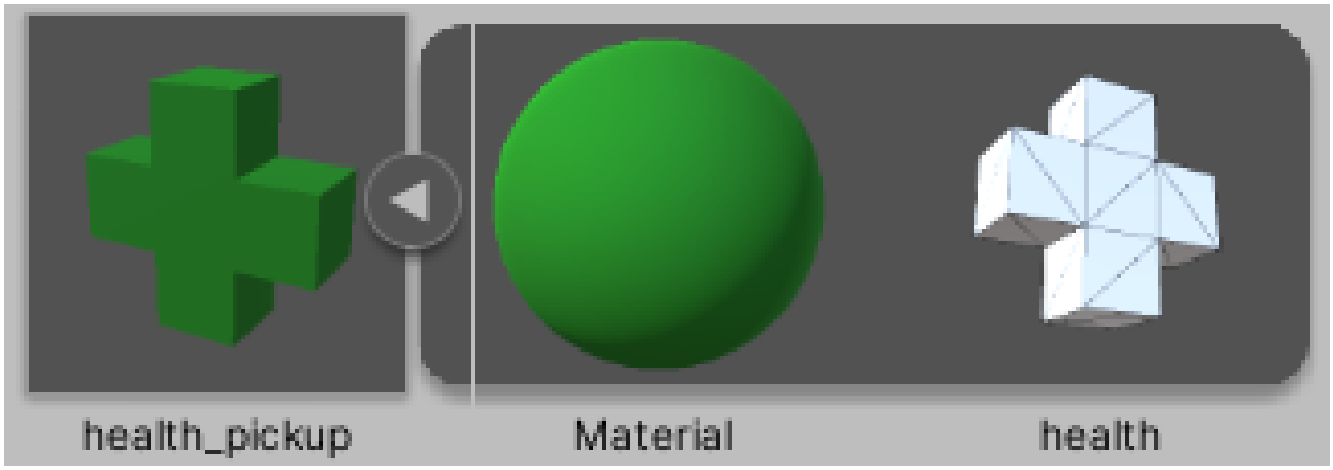


Inoltre anche il portale di fine livello è dotato di un effetto luminoso che gli permette di essere avvistato anche dalla lunga distanza.

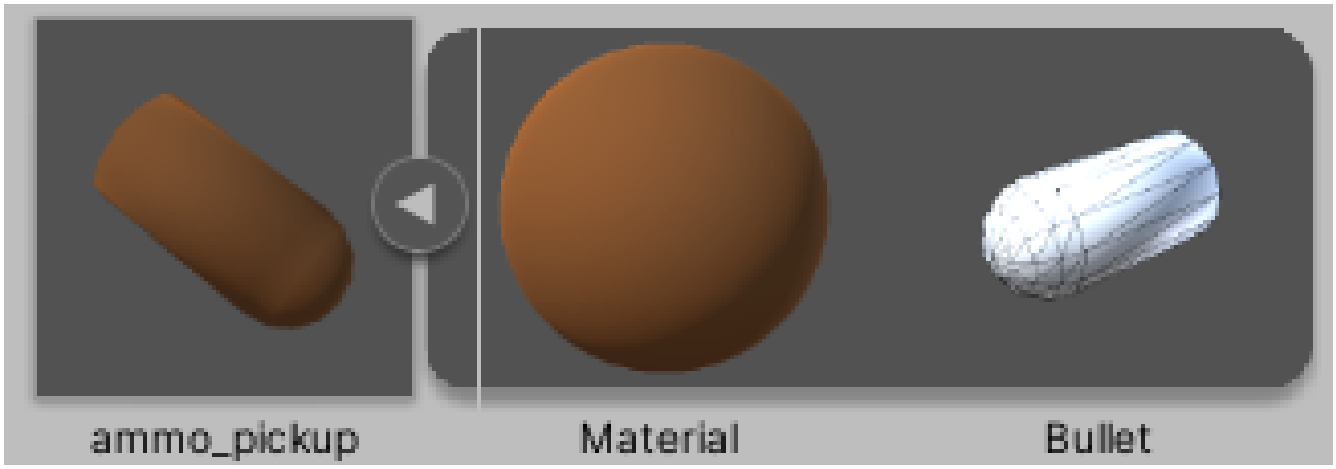


- # RACCOLTA OGGETTI

Il gioco consente di raccogliere due tipi di oggetti che si trovano all'interno della mappa. Il primo consente di ripristinare parzialmente lo stato della salute. Esattamente si ha una ricarica di 25 punti al passaggio del player sopra.



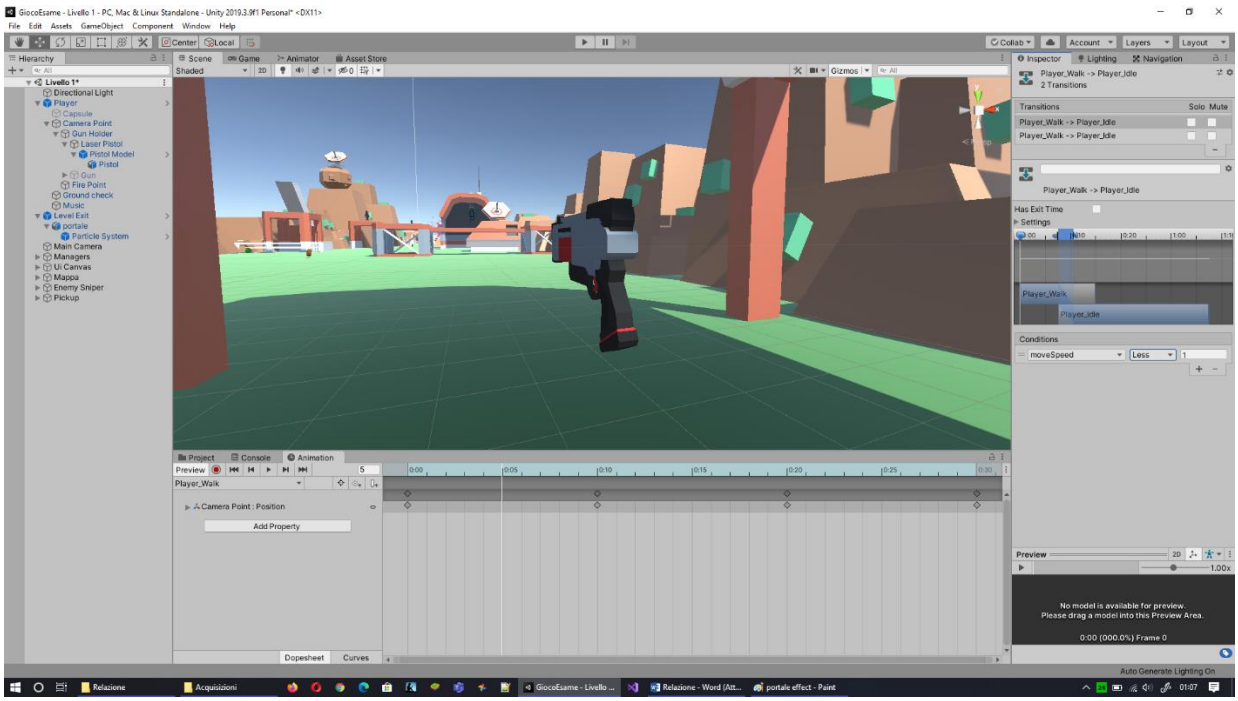
Il secondo invece funge da ricarica dei proiettili a disposizione. In questo caso è settato a 50 aggiuntivi.



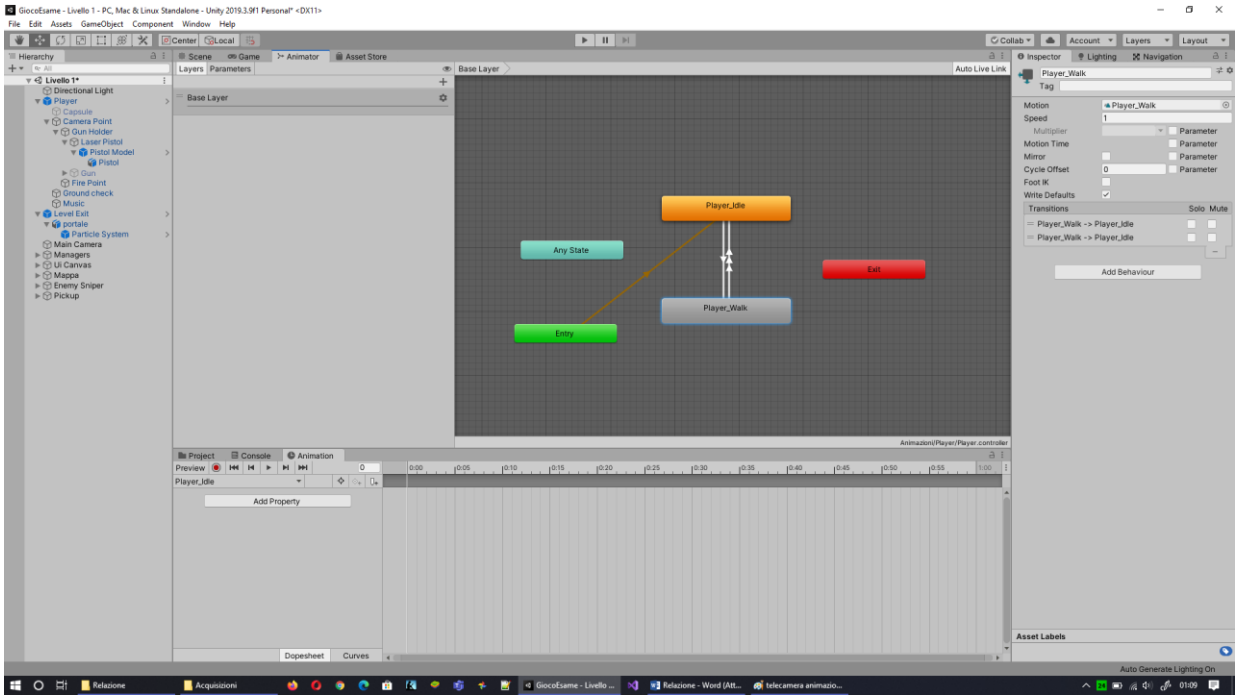
- **ANIMAZIONI**

Le animazioni consentono di rendere l'esperienza utente più soddisfacente e realistica. Infatti nel progetto sono presenti alcune di esse su determinati oggetti.

Nel momento in cui il player sta camminando la visuale si muove secondo l'asse delle Y, oscillando in alto e poi in basso, simulando i passi.

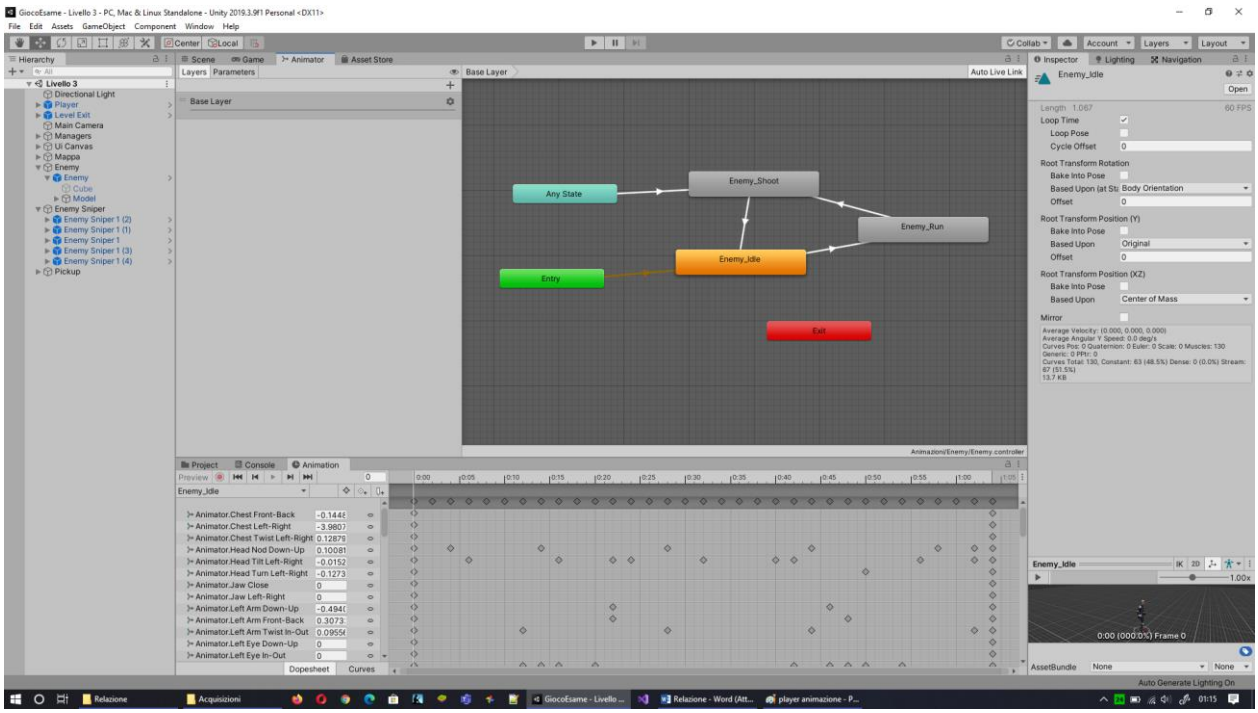


Questo è possibile grazie all'assegnazione di differenti stati: Inattivo e in movimento

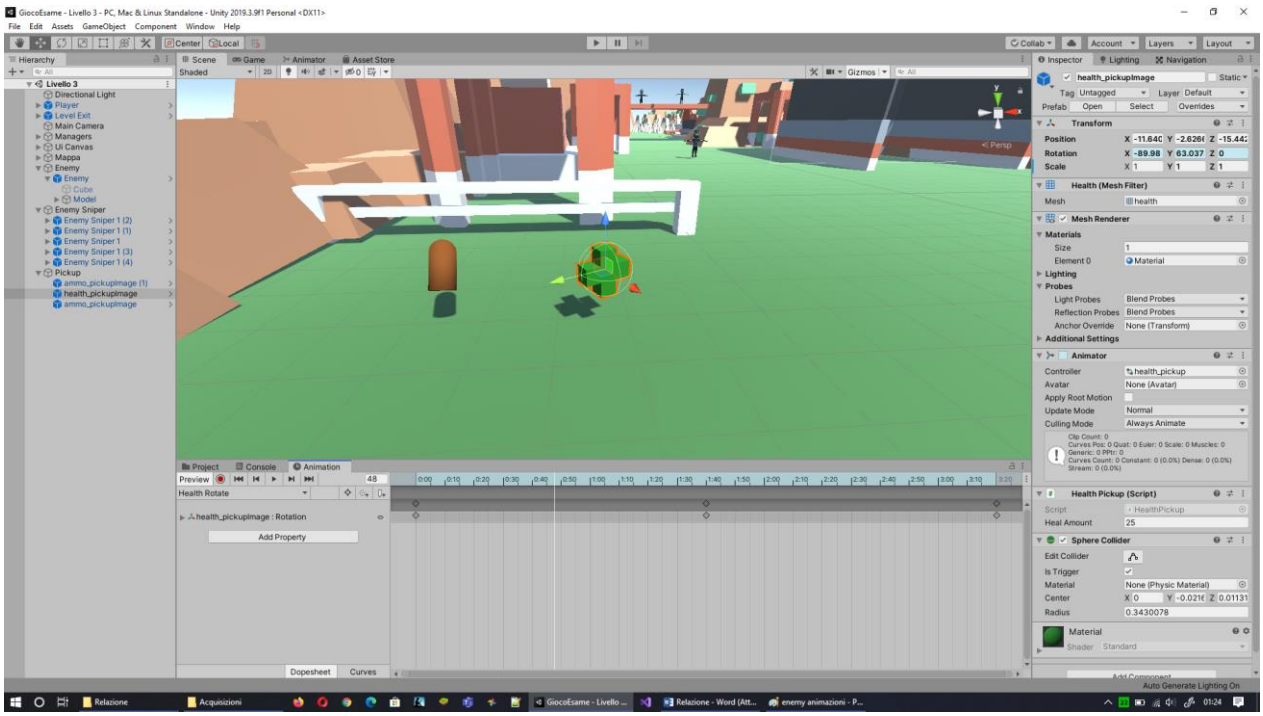


Per quanto riguarda il nemico c'è un complesso e completo schema di movimenti. L'immagine sottostante illustra il passaggio di stati da inattivo a corsa e sparo.

Questa impostazione vale solamente per il modello di nemico che è in grado di inseguire il player. Per l'altro tipo di nemico invece non è presente lo stato di "Enemy_Run" perché esso non dovrà correre.



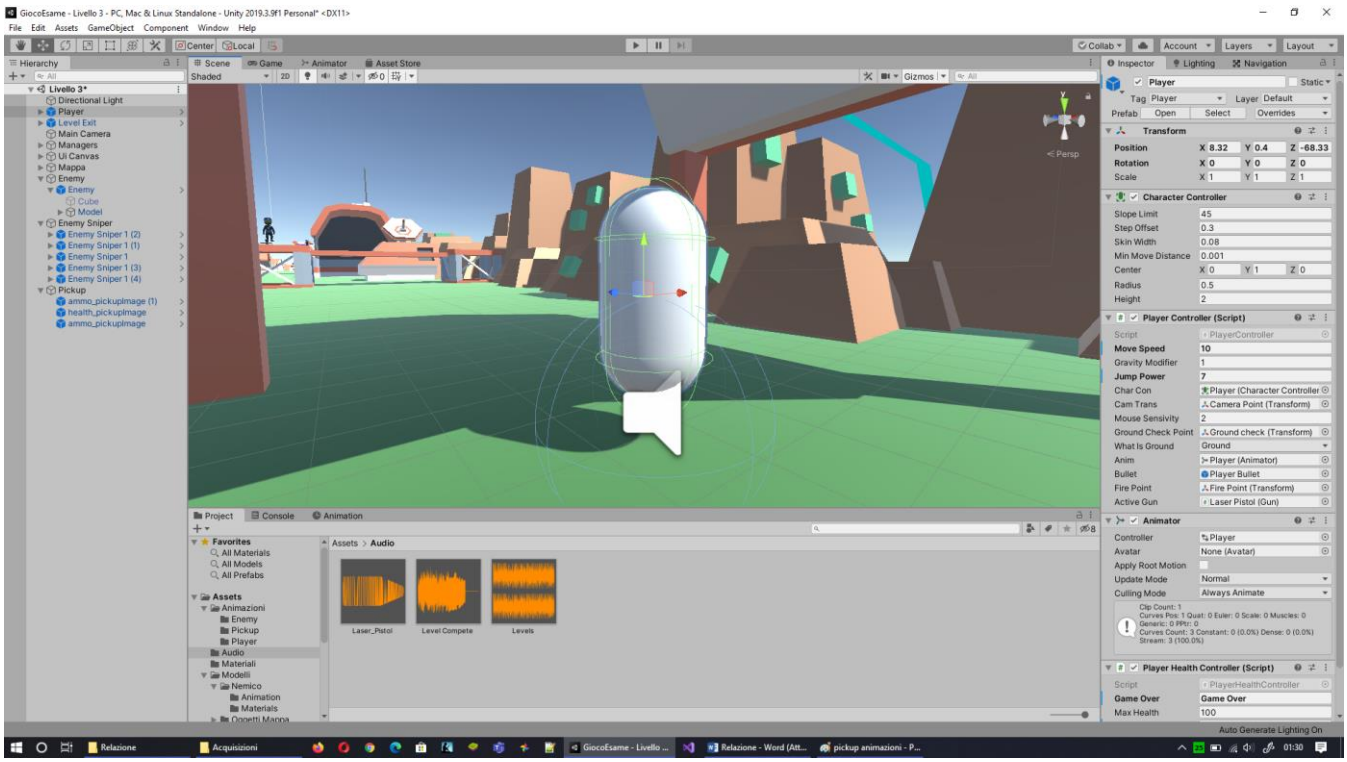
Per quanto riguarda gli oggetti da raccogliere, essi gireranno su se stessi.

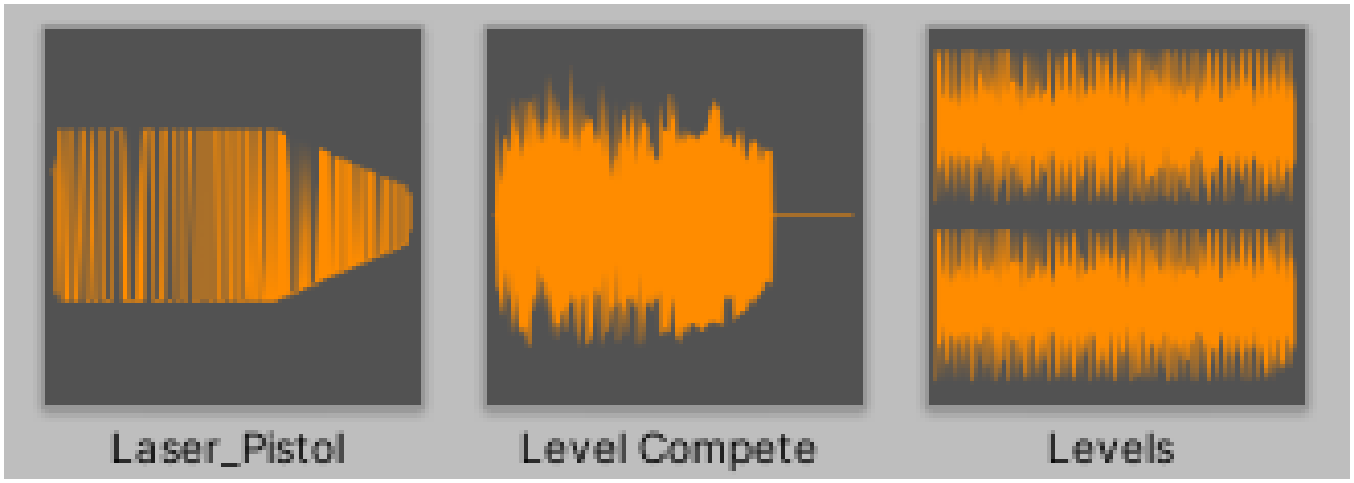


- **AUDIO**

Un buon gioco che si rispetti deve essere dotato di un audio che sia all'altezza, per questo sono stati introdotti tre differenti audio per diverse situazioni.

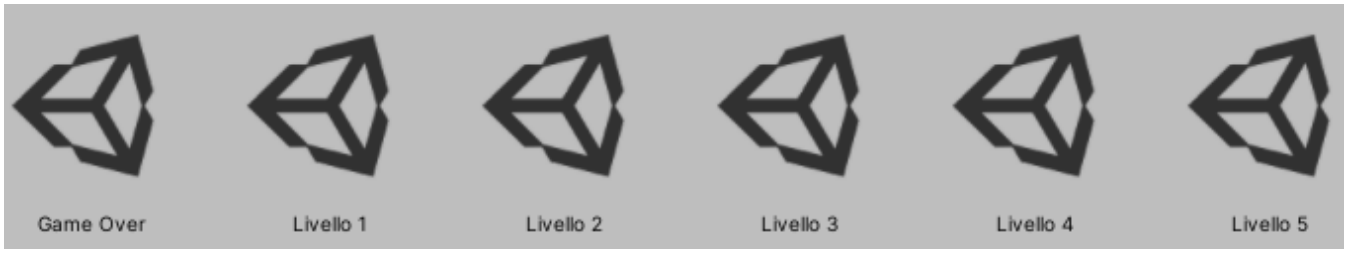
Un tipo di audio è presente durante la partita vera e propria, un altro nel momento in cui si viene sconfitti o si supera il livello e l'ultimo nel momento in cui si spara.



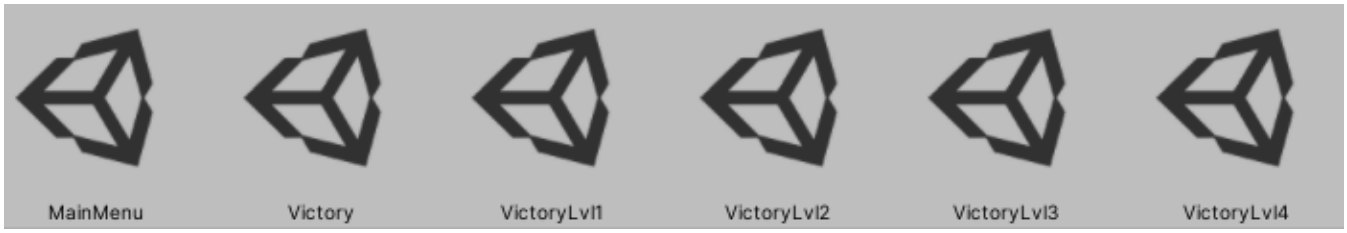


- # SCENE E STRUTTURA LIVELLI

Come da descrizione il gioco è concepito secondo uno schema di avanzamento di livelli al completamento di essi. In AlienShoot in particolare sono cinque e ognuno di questi presenta una difficolà e una mappa differente dagli altri. Al progredire dei livelli la difficolà di completamento è sempre maggiore e in particolare questo è dovuto al fatto che vi sono più nemici e la loro potenza di gioco è superiore.

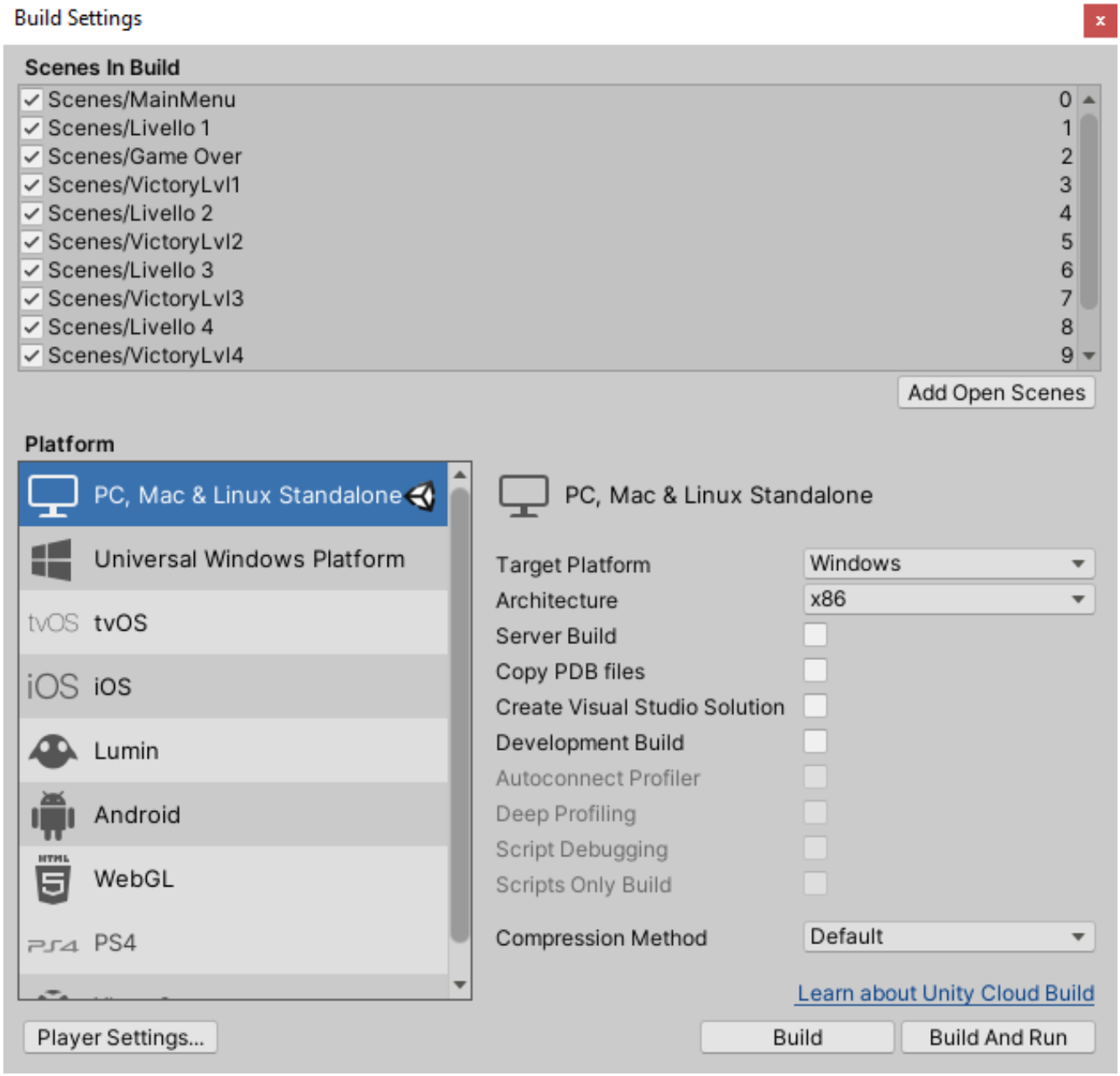


In Unity ogni livello e schermata è rappresentato da una scena. In questo progetto sono presenti oltre ai livelli, anche un menù principale, una schermata di game over e le varie schermate di transizione tra un livello e l'altro.



- # BUILD ESEGUIBILE E ALTRE APPLICAZIONI

Per la compilazione del programma e la generazione dell'eseguibile occorre impostare un ordine di apertura delle scene e in particolare in questo gioco sono impostate così:



Se volessimo estendere la giocabilità su più piattaforme, con Unity è possibile farlo convertendo l'esperienza utente in maniera rapida, eseguendo pochi accorigmenti per la compatibilità. Per fare un esempio possiamo prendere in esame il sistema Android, in questo caso occorre adattare i comandi per il movimento che siano compatibili con la risposta del touch screen.