ALMAMATER STUDIORUM – UNIVERSITÀ DI BOLOGNA

SEDE DI CESENA

FACOLTÀ DI SCIENZE MATEMATICHE, FISICHE E NATURALI

CORSO DI LAUREA IN SCIENZE E TECNOLOGIE INFORMATICHE

REALIZZAZIONE DI UN GIOCO MEDIANTE UNITY E BLENDER

Relazione in Computer Graphics

Relatore

Dott.ssa Damiana Lazzaro

Presentata da

Michela Marinelli

Anno accademico 2015-2016

III Sessione

Indice

Introduzione

In questa tesi si descrivono tutte le fasi del processo di creazione di un videogioco, a partire dalla progettazione, fino alla modellazione e allo sviluppo.

Questo verrà fatto attraverso la realizzazione di un gioco composto da tre livelli con scenari differenti: un bosco, un castello nel deserto e una stazione urbana.

Lo scopo del gioco è quello di raccogliere delle monete sparse nello scenario; una volta raccolte 10 monete il giocatore può passare al livello successivo. Le monete devono essere raccolte in un tempo massimo di 60 secondi, quindi può essere definito un gioco a tempo. Oltre al tempo sono messe a disposizione del giocatore delle vite, ad ogni livello il giocatore possiede 3 vite. Per aumentare la difficoltà sono stati inseriti degli oggetti che penalizzano il giocatore: sono presenti oggetti venendo a contatto con i quali il giocatore perde una vita ed oggetti venendo a contatto con i quali il giocatore perde una moneta. Tali oggetti sono diversi per ogni ambiente: nel primo troviamo i funghi di colori diversi dal normale che tolgono le vite e le monete; nel secondo sono presenti dei serpenti che tolgono le vite e delle ossa che tolgono vite o monete in base al colore; nel terzo sono presenti due treni che viaggiano in direzioni opposte, nel primo e nel secondo binario, che mettono in difficoltà il giocatore nella raccolta delle monete; se il giocatore viene colpito dal treno il livello termina.

Le tecnologie utilizzate per creare il videogioco sono Blender per quanto riguarda la modellazione e Unity per quanto riguarda la gestione del gioco, le interazioni con il mouse, le collisioni, etc.

Gli argomenti trattati nella tesi includono la progettazione, la modellazione ed infine lo sviluppo finale. Lo studio degli strumenti necessari alla progettazione è stato di fondamentale importanza per affrontare il lavoro di tesi.

Nel primo capitolo vengono descritte le fasi che portano alla progettazione del videogioco, tra le quali riveste un ruolo importante lo *Storyboard*, in cui vengono

disegnate a mano le scene che compongono il videogioco. Molto importante è anche la scelta dell'ambiente di sviluppo.

Nel secondo capitolo vengono analizzate le tecnologie utilizzate, Unity e Blender. Blender per la modellazione, Unity, il motore di gioco utilizzato per lo sviluppo. Per entrambi, si accennano concetti base per il loro uso e gli strumenti messi a disposizione.

In particolare nella prima parte del secondo capitolo viene presentato il software per la modellazione degli ambienti, Blender e vengono analizzate le tecniche più importanti di modellazione per la creazione degli oggetti. Nella seconda parte invece vengono trattati gli aspetti che riguardano Unity per la gestione del gioco.

Il terzo capitolo è riservato alla descrizione delle fasi di modellazione degli oggetti presenti nei diversi livelli.

Il quarto capitolo descrive la gestione del gioco e nell'ultimo capitolo vengono mostrati diversi Screenshot del videogioco per mostrare una rappresentazione del lavoro.

La piattaforma sul quale il progetto sarà giocabile è il PC. I sistemi operativi supportati sono Microsoft Windows, Linux e Mac, questo perché Unity permette di creare videogiochi multipiattaforma.

Capitolo 1

Progettazione e *Storyboard*

In questo capitolo vengono descritte la fasi necessarie alla progettazione di un videogioco, e ci si sofferma in particolare sullo storyboard che consiste nel disegnare a mano le scene che compongono il videogioco e delinearne la trama.

1.1 Scopo del gioco

Il nome del gioco è “Coins” che significa monete, questo nome viene dal fatto che il giocatore deve addentrarsi in tre ambienti per raccogliere delle monete.

Una volta che il giocatore entra in possesso di 30 monete, raccolte nei tre livelli, ha la possibilità di aprire le porte di un vagone di un treno e terminare così il gioco.

Il giocatore per poter superare ogni livello deve raccogliere 10 monete che poi porterà nel livello successivo. All’inizio di ogni livello il giocatore avrà a disposizione 3 vite e un tempo massimo di 60 secondi per completare il livello. In ogni livello sono stati inserite delle difficoltà di diverso tipo: una che toglie una vita al giocatore e l’altra che diminuisce il numero di monete che il giocatore che ha raccolto.

Se il giocatore non riesce a raccogliere le monete nel tempo a diposizione o dovesse perdere tutte e tre le vite, allora sarà costretto a iniziare nuovamente il livello.

1.2 Storyboard del gioco

Lo Storyboard di un gioco è un elemento importante per la creazione di un videogioco, in quanto fornisce un quadro generale di quello che sarà l’ambiente e la trama del gioco.

Si tratta di una serie di disegni a mano, che illustrano l’idea alla base del gioco.

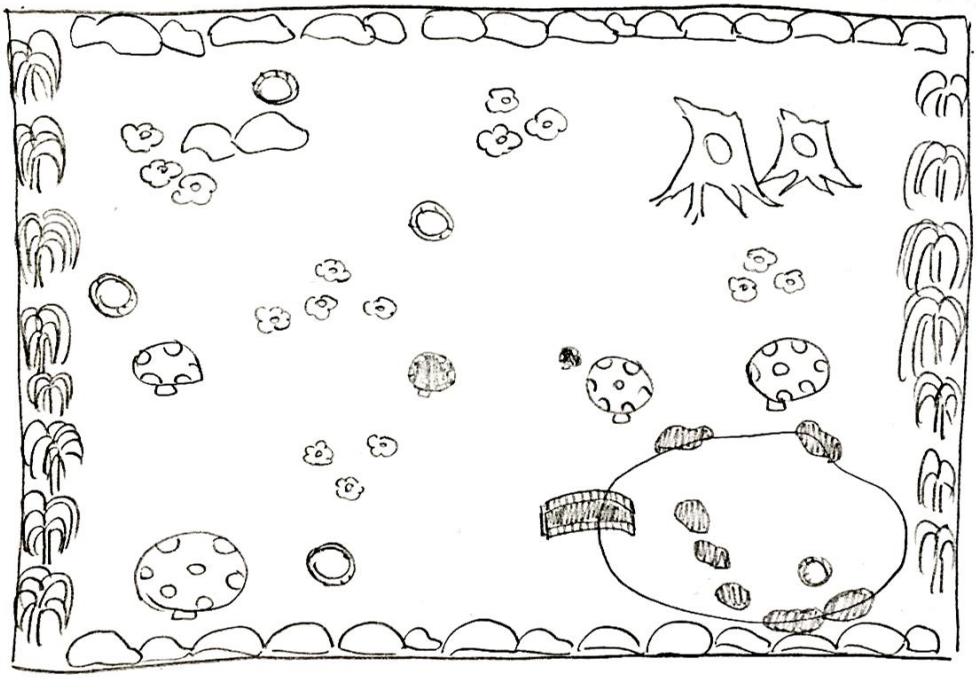


Fig. 1.1 Disegno dell'ambiente del 1° livello

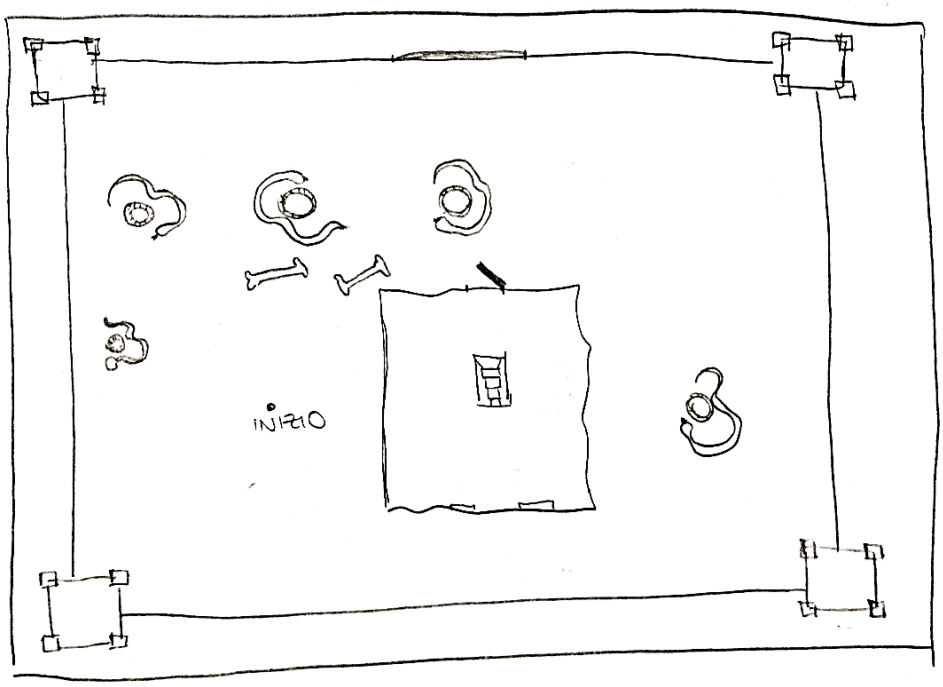


Fig. 1.2 Disegno dell'ambiente del 2° livello

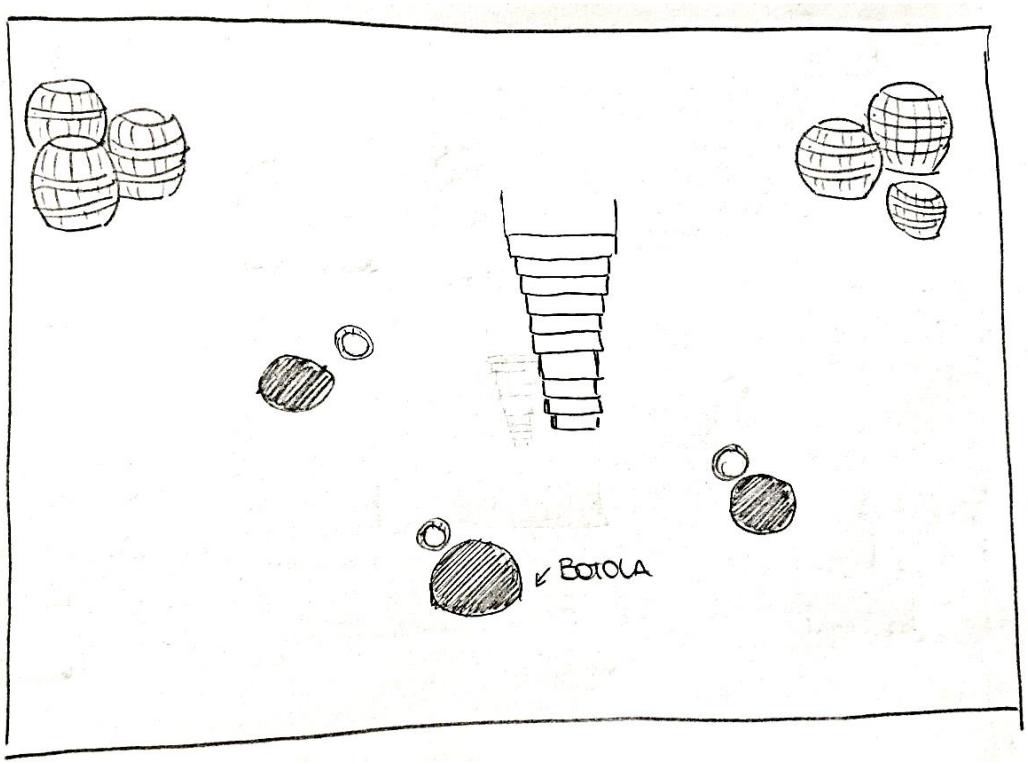


Fig. 1.3 Disegno dell'ambiente del piano sotterraneo del 2° livello

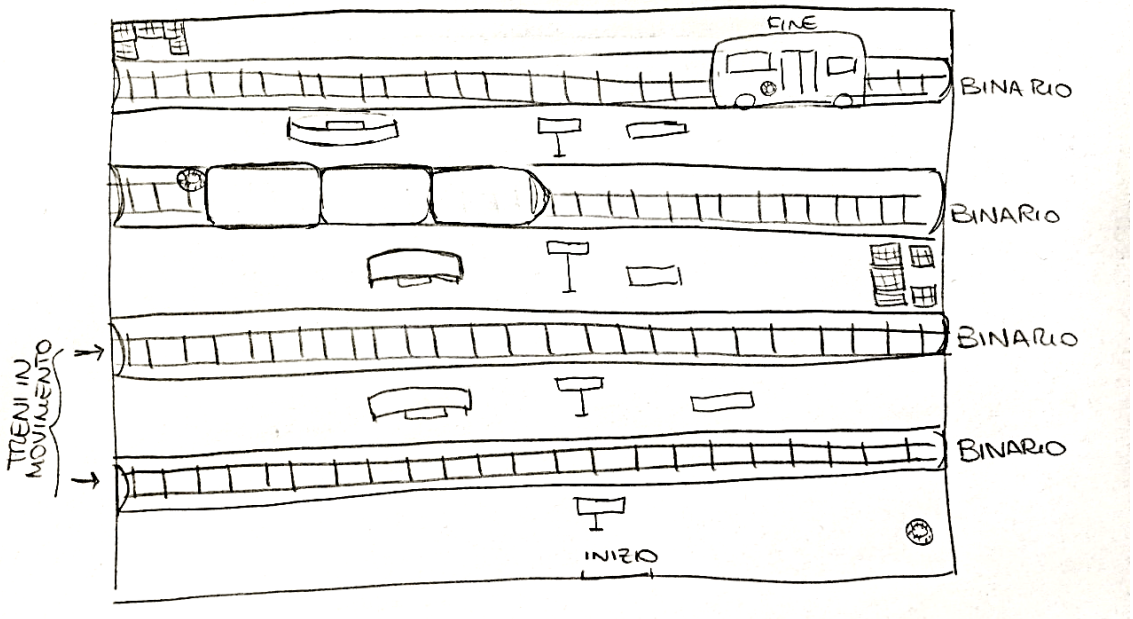


Fig. 1.4 Disegno dell'ambiente del 3° livello

1.3 Ambiente

Il gioco è stato pensato per un ambiente 3D. I vari oggetti che sono presenti negli ambienti sono stati creati attraverso il software di modellazione Blender.

La dimensione e la texture degli oggetti a volte non equivale a quella reale, infatti sono presenti funghi che non rispecchiano del tutto le caratteristiche reali. I funghi sono molto grandi e con colori fantasiosi.

1.4 Giocatore

Il giocatore si muove all'interno della scena ricevendo degli input da tastiera: 'w' muove il personaggio in avanti, 'a' e 'd' fanno muovere a sinistra e destra rispettivamente e 's' fa andare indietro, in alternativa possono anche essere utilizzate le frecce per effettuare questi spostamenti. Il giocatore, inoltre, può utilizzare il mouse per muovere la visuale e la barra spaziatrice per saltare.

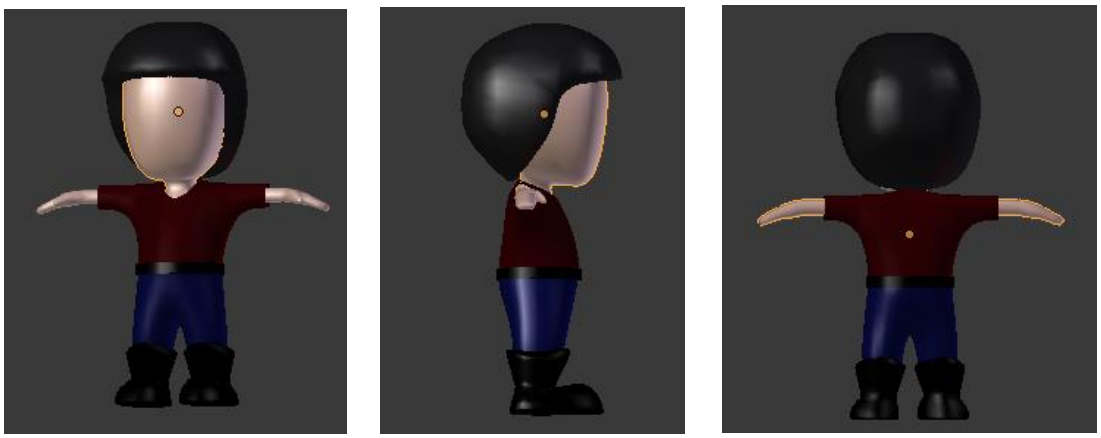


Fig. 1.5 Personaggio in vista frontale, laterale e posteriore

1.5 Difficoltà

La difficoltà del gioco aumenta con l'avanzare dei livelli. Infatti, nel primo livello non ci sono molte difficoltà nel raccoglimento delle monete. Passando al secondo livello le difficoltà sono maggiori in quanto sono presenti più oggetti "nemici", i serpenti, che mettono in difficoltà il giocatore nel prendere le monete, in quanto questi girano attorno ad esse. Un'altra difficoltà è data dal fatto che se non vengono raccolte le monete "protette" dai serpenti il giocatore non può proseguire il livello, perché non verrà aperta la porta che fa andare nel sotterraneo. Nel terzo livello la difficoltà sta nella presenza di treni in veloce movimento su più binari che mettono in difficoltà il giocatore che deve prendere delle monete presenti sui binari. In questo livello possiamo dire che la difficoltà è maggiore in quanto i treni a differenza degli altri elementi penalizzatori dei primi due livelli, tolgono tutte le vite al giocatore e il gioco riparte dal livello 3, mentre gli altri toglievano una vita e diminuivano il numero delle monete raccolte.

1.6 Livelli

Lo scenario scelto per i tre livelli è differente per ognuno di essi. Infatti nel primo livello l'ambientazione avviene in un bosco, nel secondo in un castello nel deserto e nel terzo in una stazione ferroviaria. Il passaggio da un livello al successivo è gestito da algoritmi implementati in script utilizzabili da Unity. Il gioco può anche essere pensato per sviluppi futuri aggiungendo altri livelli con nuovi scenari.

1.7 Punteggio e Tempo

Il punteggio del giocatore corrisponde al numero di monete che raccoglie. Le monete raccolte vengono mostrate a video in modo che il giocatore si possa rendere conto di quante monete gli mancano per poter passare al livello successivo. Viene mostrato al giocatore anche il tempo a disposizione, in modo che può vedere il tempo rimanente per completare il livello.

Capitolo 2

Tecnologie utilizzate

2.1 Blender

In questo capitolo vengono descritti gli strumenti e le tecniche utilizzate per la creazione degli oggetti che compongono il gioco. Parleremo di Blender e della modellazione 3D. *[1]*, *[2]*.

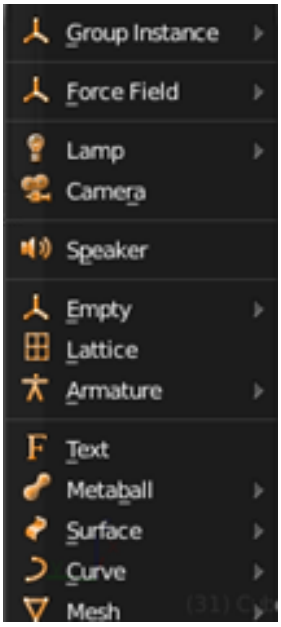
2.1.1 Cos'è Blender?

Blender è un software per Computer Graphics Open Source con cui è possibile creare oggetti tridimensionali, animare oggetti, creare giochi e tanto altro.
Blender mette a disposizione tutte le funzionalità per la modellazione, l'illuminazione, la generazione di immagini (rendering), l'animazione, inoltre dispone di funzionalità per mappature UV, simulazioni di fluidi, di rivestimenti, di particelle.
Questo software fu originariamente sviluppato dalla *Neo Geo* (studio di animazione olandese) come loro applicazione interna per *Computer Graphics.*
L'autore principale di Blender fu Ton Roosendaal che nel 1998 fondò la "Not a Number Technologies" per sviluppare il programma e distribuirlo. Dopo il fallimento dell'azienda Ton Roosendaal fondò la *Blender Foundation,* della quale è ancora oggi presidente, per guidare lo sviluppo futuro del software.

2.1.2 Modellazione 3D

Il processo che porta alla creazione di un oggetto viene detto *3D Modeling* e l'oggetto che si va a creare viene chiamato *modello 3D*.

Blender mette a disposizione strumenti diversi per facilitare il processo di costruzione di un modello in modo rapido ed efficiente:



Objects

Operazioni con gli oggetti nel suo complesso

Meshes

Lavorare con la rete che definisce la forma di un oggetto

Curves

Usare le curve per modellare e controllare gli oggetti

Surfaces

Modellare una superficie NURBS

Text

Strumenti testuali per mettere le parole in uno spazio 3D

Meta Objects

Gocce e Globuli

Ci sono principalmente due modi per realizzare un modello:

- Polygonal modeling: il modello è rappresentato attraverso dei vertici connessi da spigoli che formano una mesh poligonale. Questo tipo di rappresentazione è il più diffuso, soprattutto nell'ambito dei videogiochi, perché sugli hardware moderni la renderizzazione avviene velocemente.
- Curve modeling: il modello è rappresentato da superfici definite a partire da curve per rivoluzione, estrusione, Loft.

Uno dei modi messi a disposizione da Blender è la modellazione a partire da una Mesh di base.

Si possono scegliere diversi tipi di mesh presenti nel menu di scelta delle mesh.

Spesso viene usato il “box modeling” cioè partire con un cubo di base, per poi procedere estruendo le facce e muovendo i vertici per creare una “mesh” più complessa.

Gli strumenti principali che Blender mette a disposizione per effettuare queste modifiche sono:

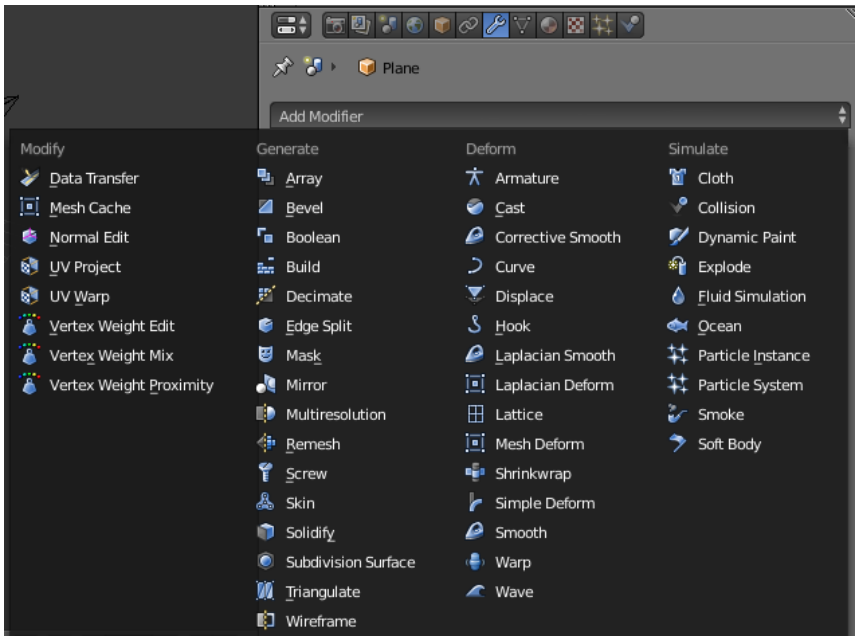
- *Translate*: permette di spostare tutto o parte dell’oggetto all’interno dello spazio 3D, si può effettuare premendo il tasto ‘G’ della tastiera;
- *Rotate*: permette di ruotare l’oggetto, si può effettuare premendo il tasto ‘R’ della tastiera ed è possibile specificare su quale asse effettuare la rotazione (X,Y,Z);
- *Scale*: permette di scalare il modello, si può effettuare premendo il tasto ‘S’ della tastiera, anche in questo caso possiamo scegliere su quale asse scalare l’oggetto, se non scegliamo l’asse, l’oggetto viene scalato in modo uniforme.

2.1.3 Modificatori

Un modificatore è l’applicazione di un “processo” o “algoritmo” ad un oggetto, in modo da conferirgli delle proprietà particolari. Uno dei modificatori considerato fra i più utili è il *Mirror* che specchia automaticamente la mesh lungo i suoi assi locali X, Y e/o Z, che passano attraverso il suo centro. Questo semplifica il lavoro per effetti simmetrici, in quanto è sufficiente lavorare solo su una metà.

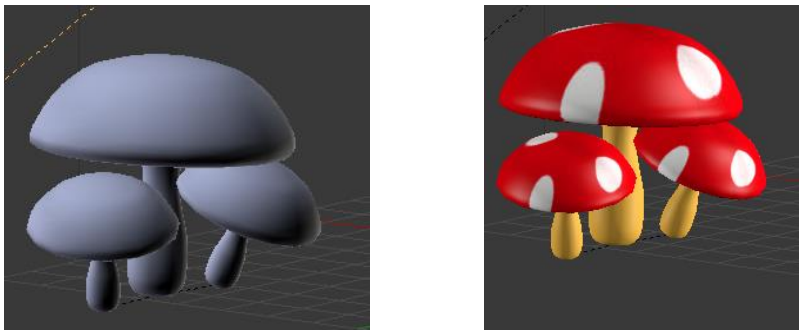
I modificatori si selezionano dal menu *Modifiers,* possono essere applicati in modo interattivo e non distruttivo nell’ordine in cui viene scelto. Questo tipo ti funzione viene definita “*modifier stack*”, cioè pila dei modificatori.

In un *modifier stack* l'ordine con cui dei modificatori vengono applicati ha effetto sul risultato.



2.1.4 Texture mapping

Per modificare il colore della superficie di un modello 3D ci si affida al *texture mapping*. Con *texture mapping* si intende il processo che porta all'applicazione di un'immagine sulla superficie dell'oggetto. Le texture necessitano delle coordinate di mappatura per esse applicate sul modello 3D, la texture viene così avvolta sull'oggetto.



(a) Senza Texture *(b) Con Texture*

Fig2.1 Esempio di texture mapping sui funghi

Prima di poter applicare una texture su un modello è necessario eseguire un *UV Mapping*. Questo porta a mappare tutta la superficie del modello 3D su un piano 2D, questo è fondamentale perché la mesh è tridimensionale mentre l'immagine che vogliamo applicare è bidimensionale. In genere nel risultato finale saranno presenti dei tagli, questo perché il modello deve essere diviso in regioni per essere posizionato sul piano 2D. In alcuni casi i tagli sono quasi invisibili, questo dipende dalla qualità dell'UV Map che si ha a diposizione.

2.2 Unity

In questo capitolo si prende in esame Unity, il motore di gioco scelto per sviluppare il gioco, descrivendo i concetti base, il funzionamento e le caratteristiche principali messe a diposizione.

2.2.1 Cos'è Unity

Unity è un ambiente di sviluppo creato per la creazione di videogiochi 3D o altri contenuti interattivi, quali visualizzazioni architettoniche o animazioni 3D in tempo reale. *[3]*.

Unity 3D è composto principalmente da un motore grafico 3D, un motore fisico, componenti per l'audio, componenti per l'animazione, un IDE grafico per la gestione dei progetti e permette di scrivere i propri script utilizzando JavaScript e C#.

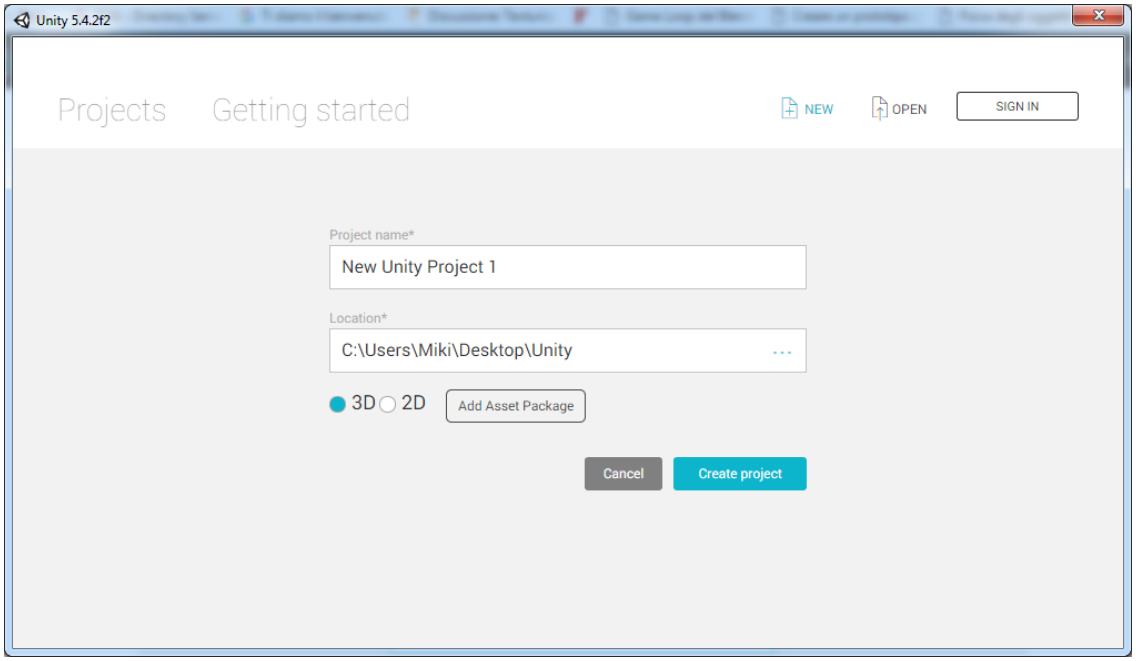
Unity è definito multipiattaforma perché il suo motore permette di "scrivere il gioco" una sola volta e realizzarlo o trasformarlo per ambienti o circuiti diversi: parliamo della creazione di uno stesso gioco per PC (Windows, Mac), Play Station, Wii, Xbox, Iphone, Ipad, cellulari con sistema Android.

La licenza è proprietaria ma è comunque disponibile una versione gratuita destinata ad uso privato o a piccole aziende.

2.2.2 Interfaccia di Unity

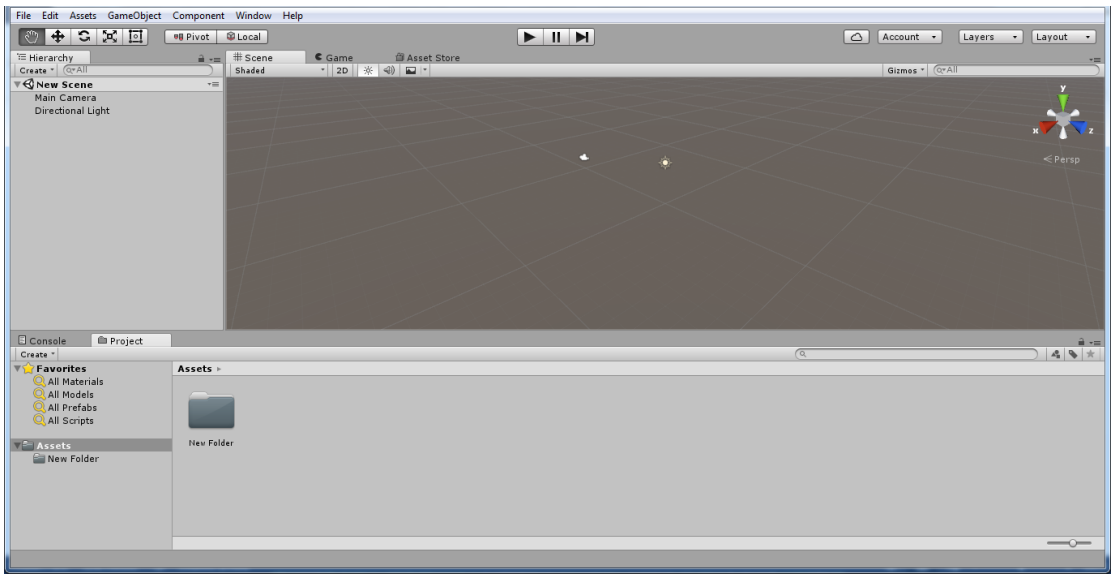
L'interfaccia di Unity è qualcosa di estremamente complesso. Molti abbandonano l'idea dello sviluppo di un gioco proprio perché non riescono ad orientarsi con la sua interfaccia. Quindi è importante spiegare il suo funzionamento all'interno di questa tesi.

Aprendo Unity 3D la schermata che troviamo è la seguente: il pannello **Projects** contiene tutti i progetti creati con Unity, mentre il pannello **Getting started** contiene un piccolo video introduttivo. I tre pulsanti sulla destra ci permettono di scegliere un progetto su un'altra directory, creare un nuovo progetto e gestire il nostro account Unity. Cliccando su New verremo portati su questa schermata:



Dobbiamo scegliere il nome del progetto e la directory sul quale salvarlo, inoltre possiamo scegliere possiamo scegliere se creare un progetto 3D o 2D e se importare degli **Asset Packages.**

Gli **Asset Packages** sono dei pacchetti predefiniti contenenti script, mesh, animazioni e molte altre cose che possono essere utili durante la creazione di un videogioco. Questi possono essere importati anche in un secondo momento.

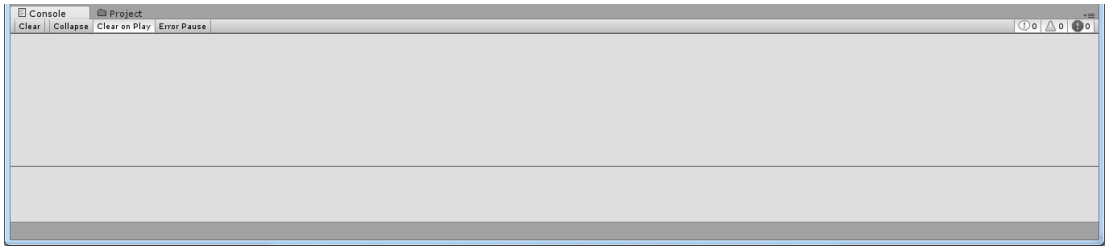


Nel pannello **Scene** viene mostrata la scena e gli oggetti che sono presenti all'interno di essa. Gli oggetti sono gli elementi che compongono il nostro gioco: telecamera, luci, modelli 3D…

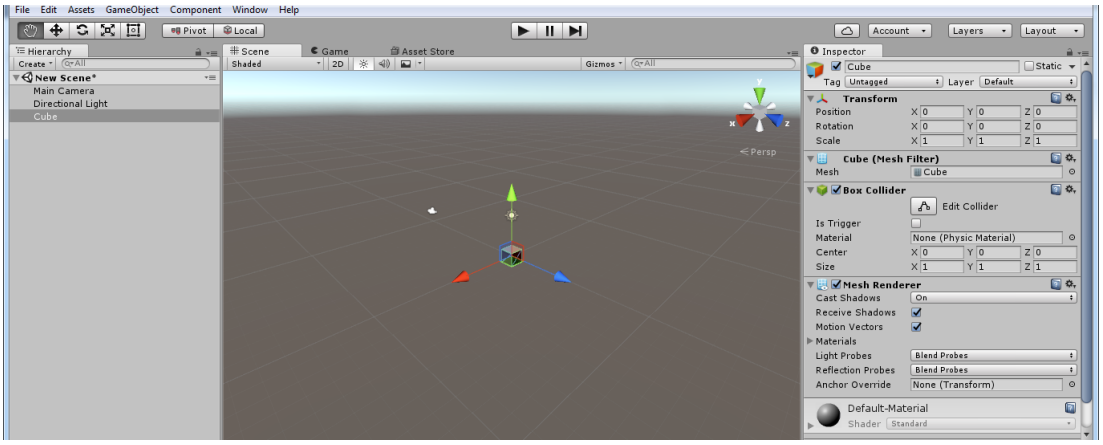
All'interno di ogni progetto possono essere presenti una o più scene. Le scene, come nel gioco che ho creato, possono essere considerate come i livelli del gioco.

Di default sono già presenti degli oggetti nella scena: una **Camera** e una **Directional light**.

Nel pannello **Hierararchy panel** vengono elencati tutti gli oggetti presenti nella scena. Con il pulsante *Create* si possono inserire nuovi oggetti nella scena, inoltre si possono creare delle gerarchie tra due o più oggetti, basta selezionare gli oggetti e trascinare un oggetto su un altro e fare uso di *Parenting* per creare una parentela. L'oggetto esterno verrà chiamato "padre", mentre quello interno sarà il "figlio". In questo modo quando sposteremo l'oggetto padre, questo sarà seguito dall'oggetto figlio; non possiamo dire la stessa cosa nel caso contrario.



Nella **Console panel** è fondamentale per testare il corretto funzionamento di uno script.



Nell' **Inspector panel** ci sono tutte le impostazioni relative all'oggetto selezionato.

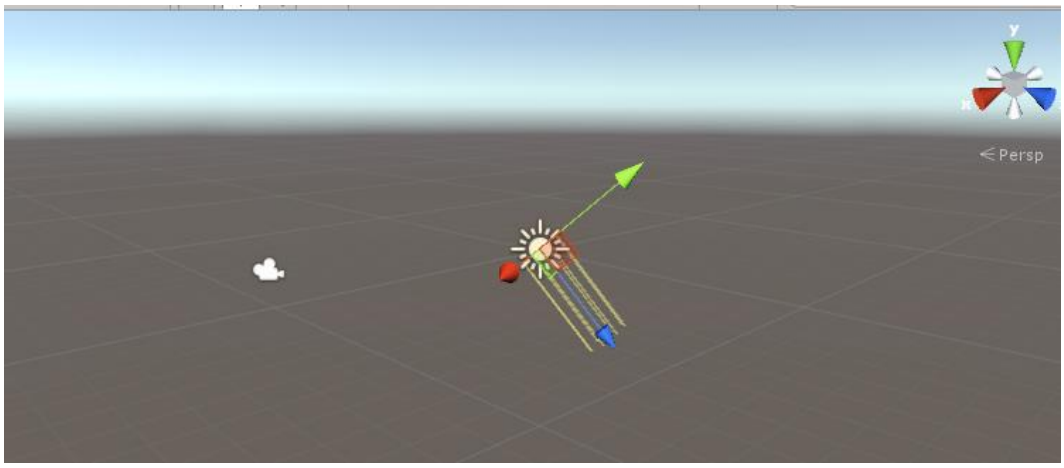
Nel **Game panel** viene mostrata un'anteprima del videogioco. La visuale che ci verrà mostrata sarà quella mostrata dalla Camera presenti nel gioco.

2.2.3 Tipi di luci

Per creare una luce basta selezionare *GameObject > Light > Directional light* (oppure un'altra luce).

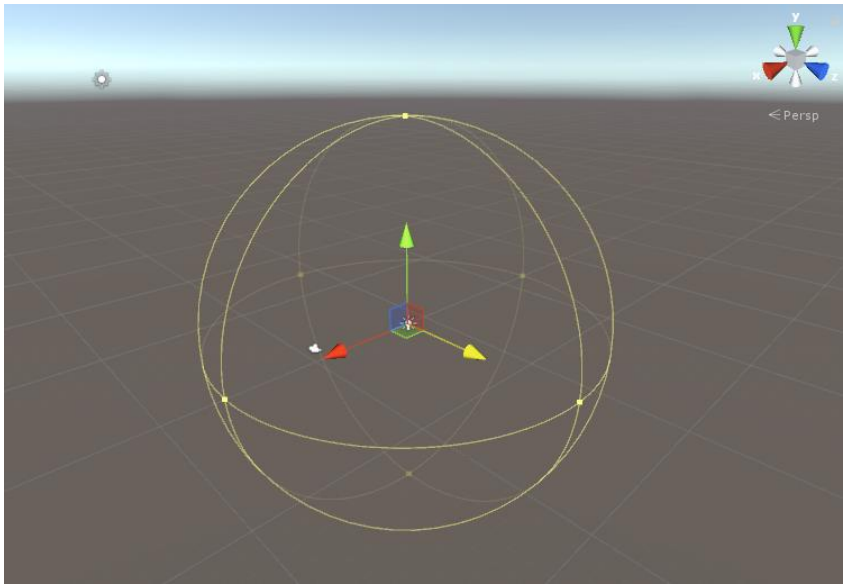
La ***Directional light*** è una luce che simula meglio l'illuminazione globale, questo perché i suoi raggi sono paralleli e a distanze infinite. Questa luce infatti influenza tutti gli oggetti presenti nella scena senza tener conto della distanza.

All'interno della scena una Directional light appare in questo modo:



I segmenti gialli specificano la direzione dei raggi di luce. Non è importante il punto in cui viene posizionata la Directional light perché il suo effetto non cambierà a seconda della posizione, l'illuminazione è condizionata però dalla rotazione, infatti se ruotiamo la Directional light l'illuminazione cambierà.

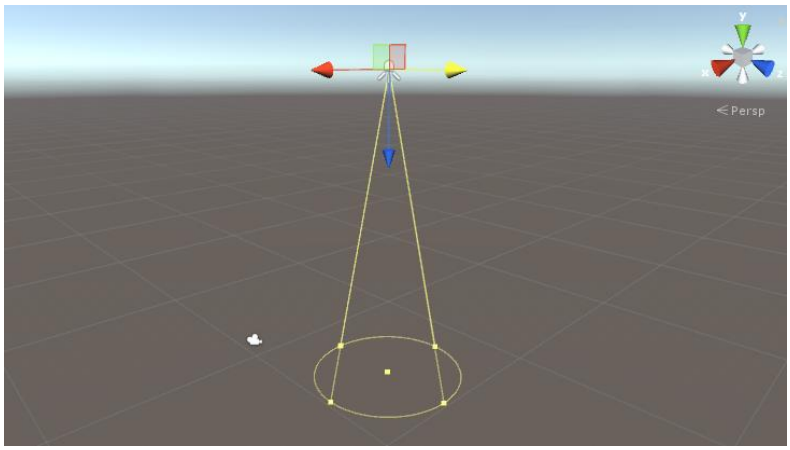
Le ***Point light*** sono luci puntiformi, adatte a creare effetti come fiamme, torce medievali.



Come si può vedere è rappresentata da una sfera che indica l'area di influenza. Sulla superficie ci sono 6 quadratini che possono essere spostati per cambiare l'ampiezza. L'oggetto che si troverà vicino al bordo di questa sferà riceverà la luce sono in maniera minima, mentre uno al centro la riceverà in pieno.

Per il suo effetto è importante la sua posizione, ma non è influente la sua rotazione.

Le ***Spotlight*** sono dei faretti che proiettano un cono di luce che ha un'ampiezza ed una distanza massima. Per questo motivo, nel caso di spotlight sono importanti sia posizione che rotazione. Si presentano in questo modo:



2.2.4 Shaders e materiali

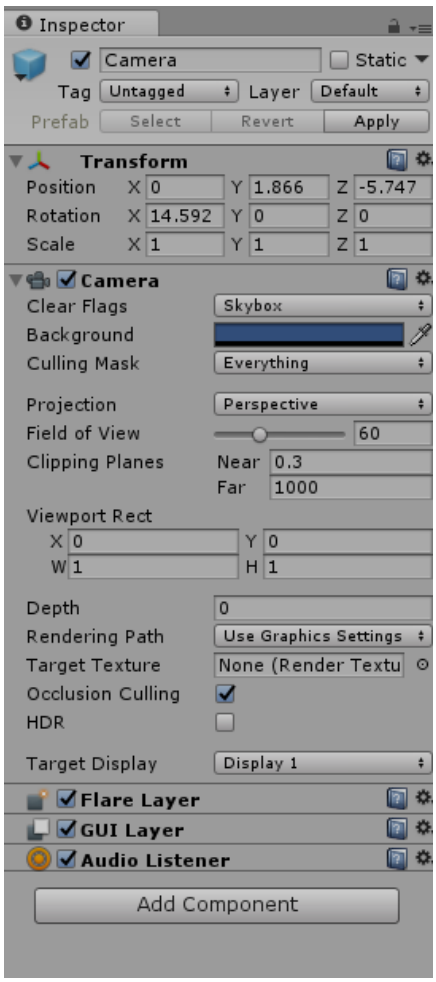
Gli shaders sono dei modelli matematici che indicano come rendere a schermo un oggetto. In pratica esprimono come un oggetto riceve la luce, che colore assume, se ha una traslucenza, se è lucido, opaco, trasparente e moltre altre caratteristiche. Ci sono shaders molto complessi che vengono utilizzati per piattaforme PC e altri semplificati, ottimi per piattaforme mobili.

I materiali in Unity hanno alla base uno shader che definisce come viene disegnato ed ombreggiato. Per creare un nuovo materiale basta cliccare col tasto destro sul Project Panel e selezionare *Create* > *Material* dal menu a tendina. Bisogna creare un materiale diverso per ogni tipo di oggetto di cui avremo bisogno. Ad esempio questi sono i materiali usati per il personaggio:

2.2.5 Telecamera

La telecamera in Unity è un oggetto, Game Object, che ha alcuni componenti di default. Questi elementi sono:

- GUI Layer, che renderizza gli elementi dell'interfaccia grafica sull'inquadratura;
- Flare Layer, che mostra i lens flare nell'immagine;
- Camera Component, che contiene tutti i settaggi e le impostazioni relative alla nostra telecamera.



In questa immagine possiamo vedere quali sono le impostazione del *Camera Component*.

2.2.6 Componenti della fisica

L'aggiunta di un componente *RigidBody* ad un oggetto vincolerà il suo movimento al controllo del motore fisico di Unity. Gli oggetti che possiedono un *Colliders* vengono definiti 'collider statici' in quanto, anche a seguito di un urto non si spostano. Per il personaggio abbiamo bisogno di un RigidBody, perché questo permette di ricevere urti anche da altri collider.

Il *RigidBody* dà all'oggetto anche la possibilità di seguire la gravità verso il basso.

Collider è la classe base da cui ereditano tutti i tipi di collider, ognuno dei quali ha una forma diversa. I colliders non devono per forza corrispondere alla geometria reale di un oggetto. Tra i vari colliders abbiamo il *Box Collider* che è una collisione base a forma di cubo. Se serve una collisione più dettagliata Unity mette a disposizione il componente *Mesh Collider* che prende un Asset mesh e costruisce la sua collider sulla base di tale 'rete'.

2.2.7 Script

Per aggiungere interattività in Unity è necessario usare degli script. Gli script risiedono nel progetto come gli altri Asset (texture, modelli,…) ed ogni script è rappresentato da un file. Unity prevede l'utilizzo di 3 linguaggi: C#, Javascript e Boo. Il linguaggio C# è quello mediamente più utilizzato in Unity.

Gli script in Unity sono tutti classi che ereditano da *Monobehavior*, una classe identifica una categoria di oggetti che hanno tutti le stesse proprietà. Ogni classe è contenuta all'interno di un file di testo, il quale contiene tutte le proprietà degli oggetti figli di quella classe. Inoltre all'interno di ogni classe saranno presenti tutti i metodi di quella classe.

Capitolo 3

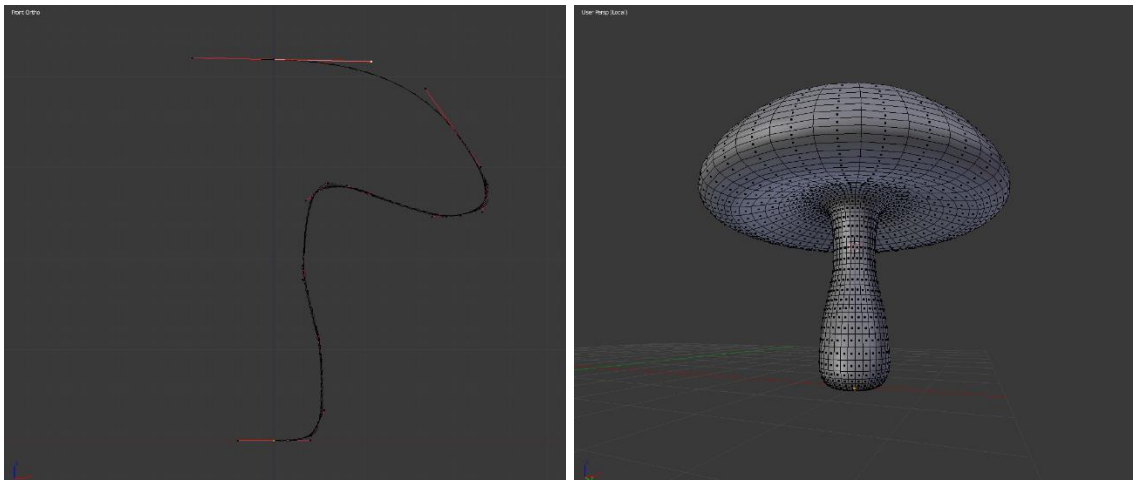
Fasi della modellazione

In questo capitolo si descrive la fase di modellazione degli oggetti che compongono le tre scene.

3.1 Modellazione del primo livello:

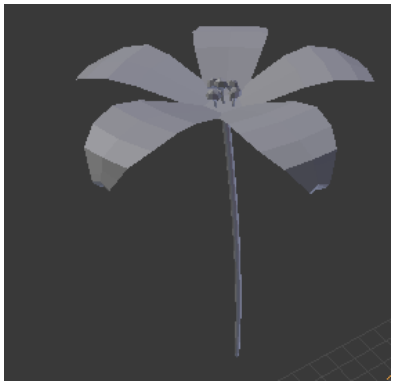
La modellazione del primo livello parte dalla creazione dei singoli oggetti che andranno poi a comporre la scena.

Per la modellazione dei funghi si è scelto di partire da un'immagine di riferimento e poi attraverso una curva di *Bezier* è stato tracciato il profilo. Una volta fatto il profilo viene applicato il modificatore *Screw*, che ruota il profili di 360° facendo ottenere così il solido. Quando si è soddisfatti del risultato si può convertire la curva in una mesh in modo da ottenere effettivamente un modello3D. A questo punto si creano i materiali e si assegnano alle varie parti del fungo per dare le colorazioni diverse.

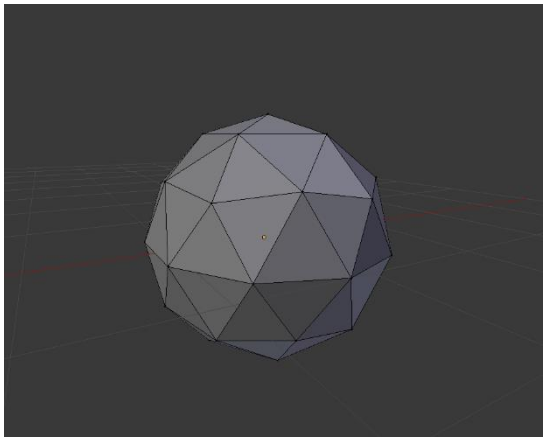
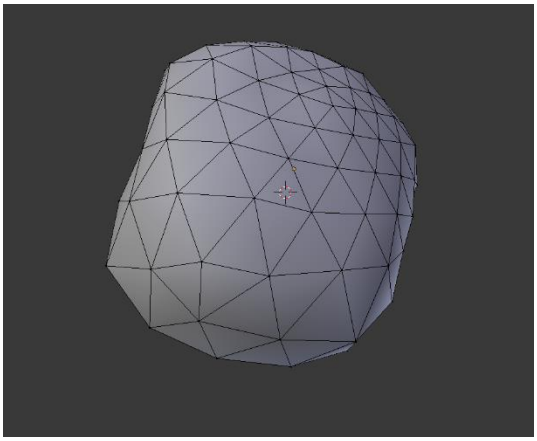


Per la modellazione dei fiori, lo stelo è stato realizzato a partire da un cilindro suddiviso varie volte per altezza e poi piegato utilizzando *Proportional editing*; i petali e i ciuffi d'erba sono stati creati a partire da un piano suddiviso varie volte e

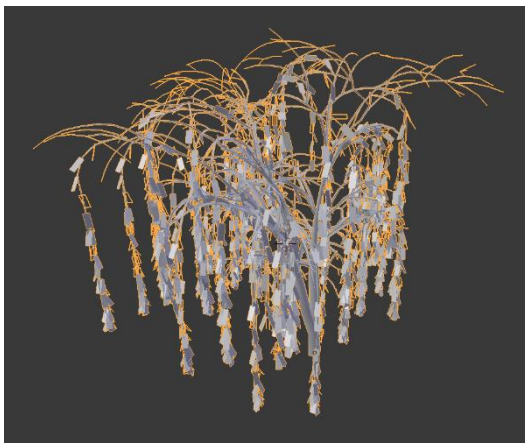
poi modificato utilizzando sempre *Proportional editing*, una volta fatto un petalo viene applicato il modificatore *Array*, utilizzando un Object Offset che ha come punto di riferimento un oggetto vuoto ruotato di 72°(per coprire tutta la circonferenza con 5 oggetti) sull'asse verticale z. Vengo poi sempre assegnate le varie parti per le diverse colorazioni.

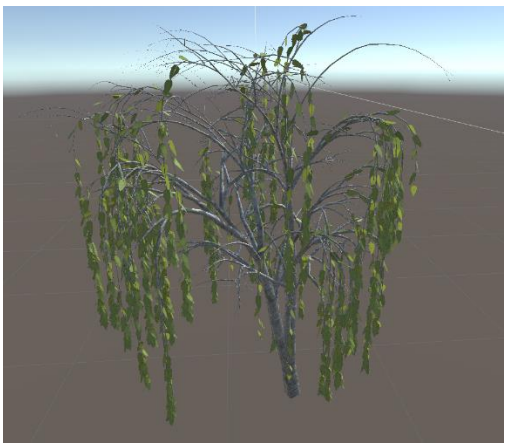



Le rocce sono state modellate a partire da un *Ico Sphere*, un icosaedro, facendo del semplice *Sculpting*.

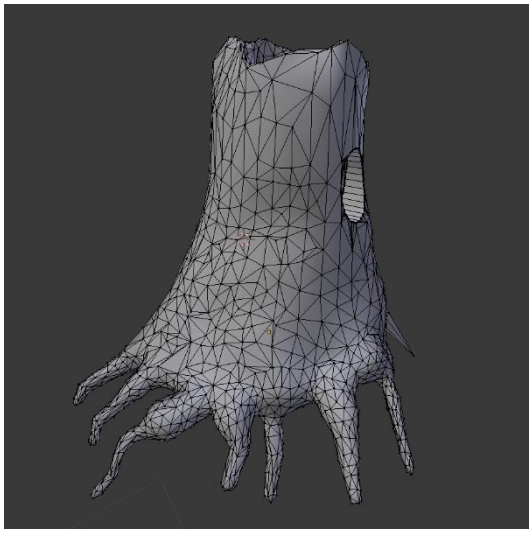



Gli alberi sono stati realizzati utilizzando un *Addon* di Blender chiamato *Supling* che permette di generarli proceduralmente sulla base di una serie di parametri. Infine è stata applicata una texture per il tronco e una per le foglie.

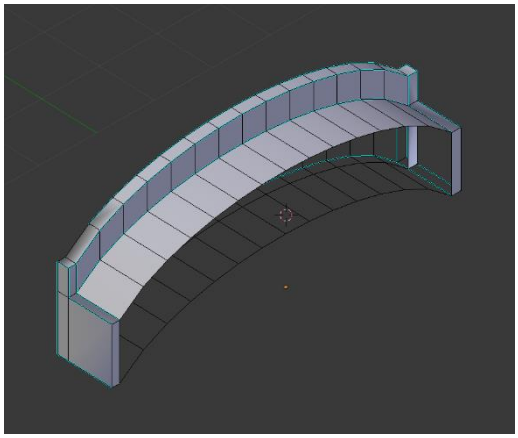


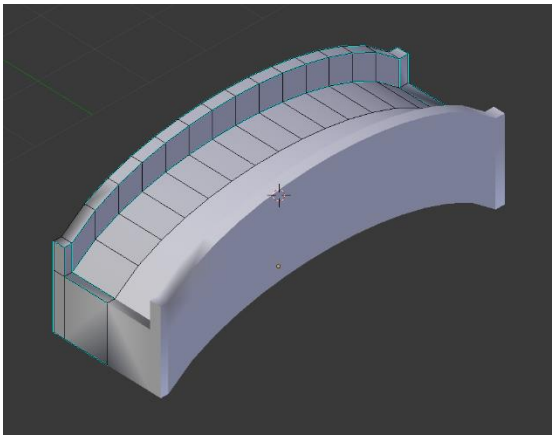


I tronchi dell'albero sono stati modellati a partire da un cilindro e applicando su di esso dello *Sculpting*, il buco è stato realizzato aggiungendo un modificatore di tipo Booleano, facendogli fare la differenza con un oggetto di forma cilindrica.



Il ponte è stato realizzato a partire da un cilindro tagliando la parte inferiore e schiacciandolo, a questo punto è stata estrusa la fascia laterale per creare il parapetto. La modellazione è stata realizzata modellando solo una metà del ponte e poi applicando su questa il modificatore *Mirror*.





Per la modellazione del terreno si è partiti da un semplice piano suddiviso varie volte andando a creare dei piccoli dislivelli sulla superficie. Per creare il laghetto è stato prima effettuato il buco sul terreno e poi applicato sopra un piano sul quale è stato aggiunto il materiale per dare l'effetto dell'acqua.

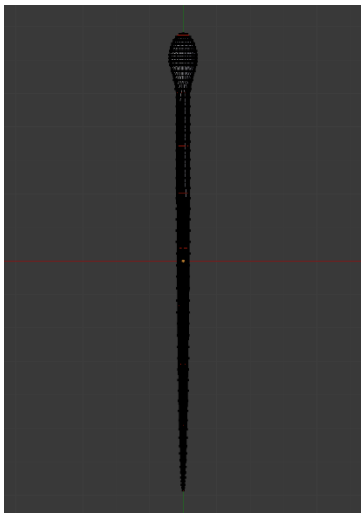
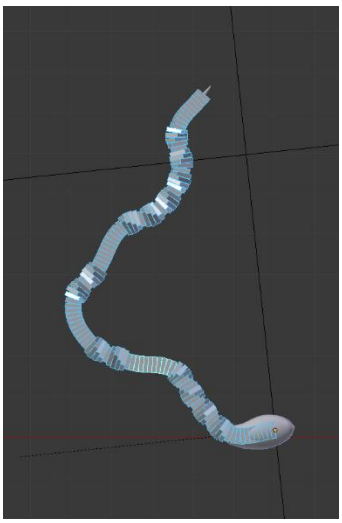
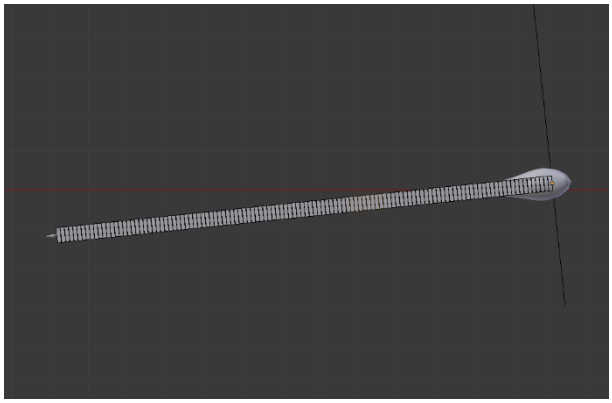
3.2 Modellazione del secondo livello:

Come per il primo livello la modellazione dell'ambiente parte sempre dalla creazione dei singoli oggetti che andranno poi a comporre la scena.

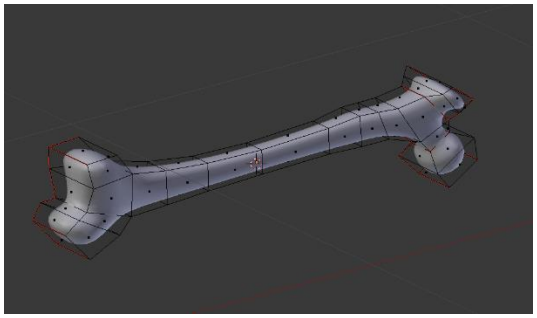
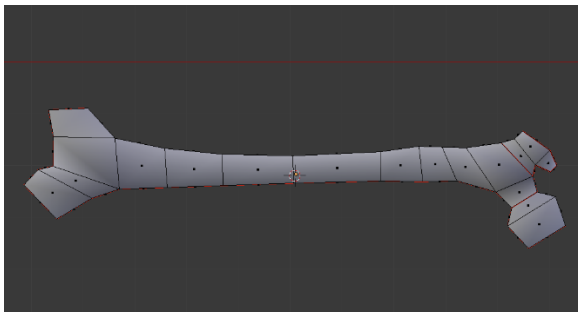
Le mura esterne e le torri sono state realizzate scalando un cubo e realizzando la merlatura con tanti altri cubi.

L'edificio all'interno è stato modellato a partire da un cubo eliminando la parte superiore e suddividendolo tante volte in modo da poter così creare dislivello della parte superiore delle pareti. Sono state eleminate alcune facce per creare porte e finestre. Per dare spessore al cubo è stato applicato il modificatore *Solidify*.

La modellazione dei serpenti è stata effettuata attraverso un cilindro, allargando la testa e stringendo la coda attraverso *Proportional editing.* L'animazione è stata realizzata mediante un *Rig* composto da 12 ossa distribuite lungo tutto il corpo del serpente. A questo punto le ossa sono state posizionate in varie pose per ricreare i vari stadi del movimento del serpente, sono stati poi inseriti tutti i frame delle rotazioni delle ossa.

Per la modellazione della ossa come prima cosa è stata presa un'immagine di riferimento. Estrudendo un cubo si è cercato di seguire il suo profilo. E' stato poi applicato il modificatore *Subdivision Surface* per arrotondare il modello.

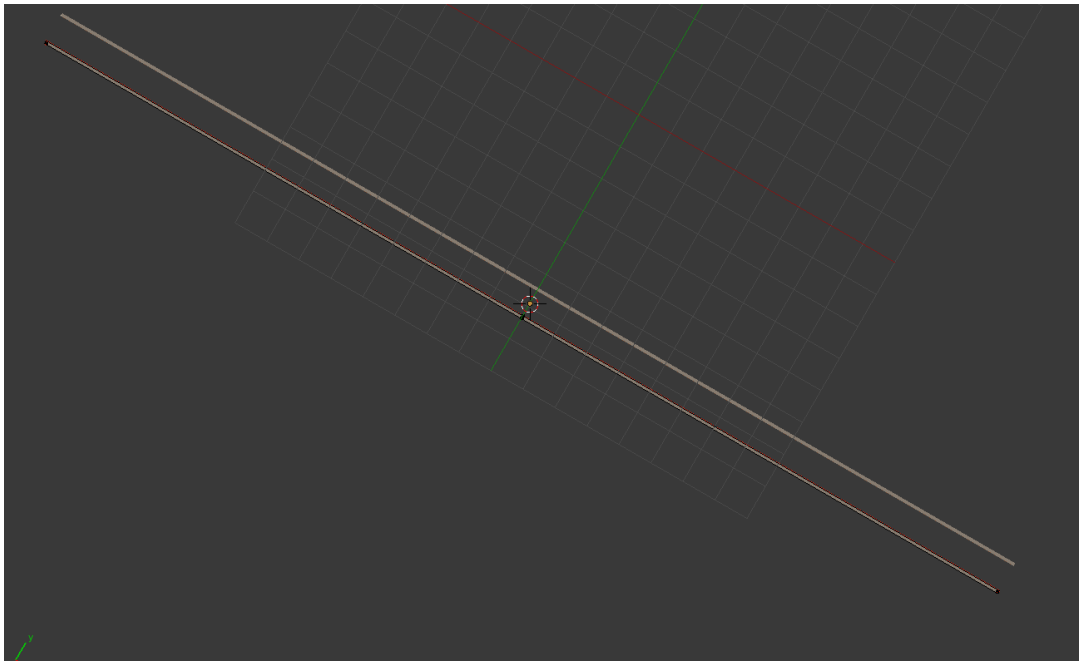



Altri oggetti semplici come il sotterraneo, le scale e le porte sono state realizzate utilizzando tecniche del tutto simili a quelle descritte in precedenza.

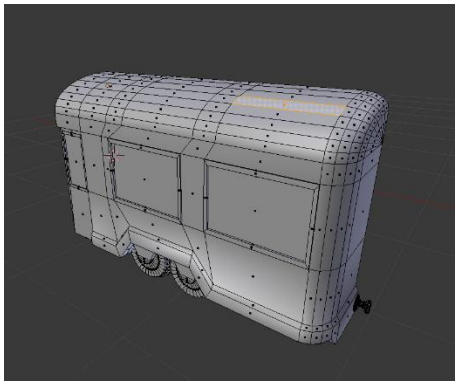
Le botti sono state modellate a partire da un cilindro e aggiungendo diverse suddivisioni orizzontali, a questo punto con *Proportional editing* si è ingrandita la parte centrale. Per creare le fasce in rilievo si sono selezionati alcuni loop di facce e sono stati estrusi verso l'esterno.

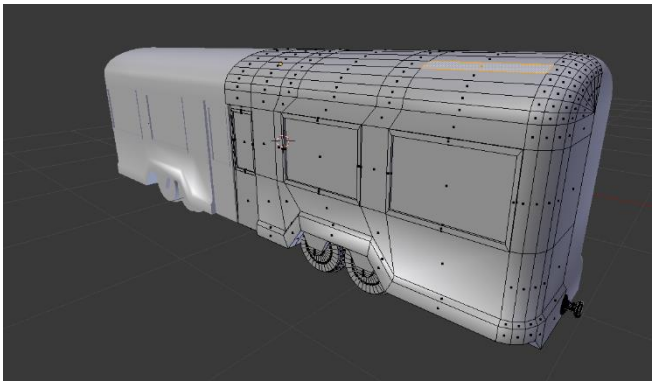
3.3 Modellazione del terzo livello:

Le rotaie sono state realizzate partendo da un cubo allungato al quale è stato applicato il modificatore Mirror per creare l'altra parte della rotaia. Poi sono stati aggiunte le vari traverse utilizzando sempre un cubo allungato e applicando su esso il modificato *Array*.



La modellazione dei vagoni è avvenuta sempre partendo da un cubo suddiviso in più parti e tramite spostamenti ed estrusioni opportune sulle varie facce si è ottenuto il modello desiderato. Inoltre partendo da cilindri sono state aggiunte le ruote. Per comodità è stato modellato solo un quarto del vagone e tutto il resto è stato creato dal modificatore *Mirror*.

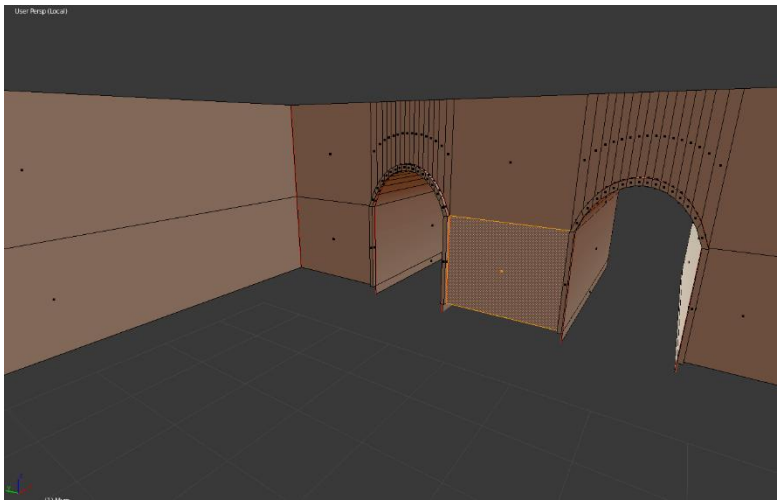




La panchina, le casse e il cartello sono stati realizzati partendo da cubi e estrudendo come necessario e nel caso del cartello utilizzando anche un cilindro.

La cabina è stata realizzata come il resto ad eccezione della tettoia che è stata ottenuta prendendo una piccola parte di un cilindro.

Il muro esterno e stato creato a partire da cubo dove poi sono state inserite delle gallerie ottenute creando un buco e poi andando ad arrotondare il lato superiore con *Proportional Editing*.

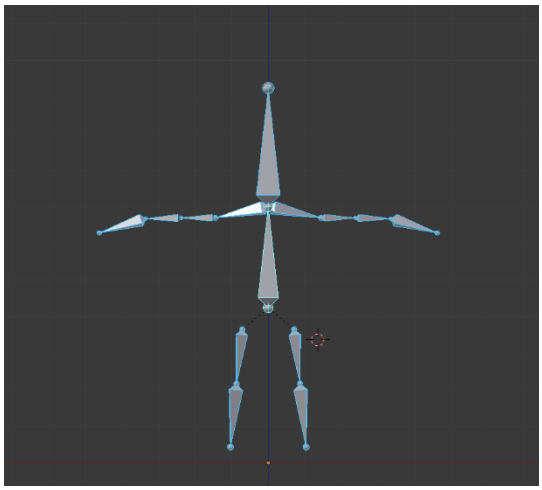


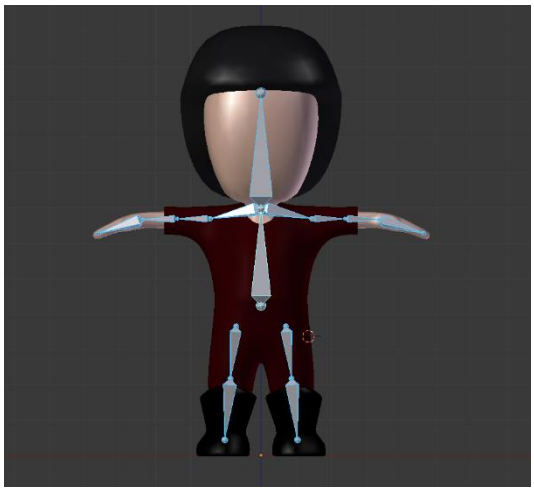
3.4 Modellazione del personaggio:

Per la creazione del modello del personaggio si è partiti da un'immagine di riferimento della quale si seguono le forme. Come in molti altri modelli si parte da un cubo che viene suddiviso ed estruso fino ad ottenere la forma desiderata. Viene anche aggiunto il modificatore *Subdivision Surface* per avere delle superfici più morbide ed arrotondate. In totale il personaggio è composto da 4 parti ottenute tutte con il metodo appena descritto. Queste sono: il corpo, il vestito, gli stivali e i capelli. Una volta che sono tutte ultimate vengono unite a formare un unico modello.

Per poter animare il personaggio è stato necessario creare un semplice *Rig*. Questo è composto da 15 ossa:

- 4 per il braccio destro;
- 4 per il braccio sinistro;
- 2 per la gamba destra;
- 2 per la gamba sinistra;
- 1 per la schiena;
- 1 per la testa.





Una volta realizzato il *Rig* questo viene impostato come *Parent* del personaggio in modo che quest'ultimo venga deformato allo spostamento delle varie ossa.

A questo punto rimane da animare il modello fino ad avere tutti i movimenti necessari: salto, corsa, camminata e attesa (idle).

Il processo di creazione di un'animazione consiste nel posizionare le ossa in varie pose e salvarle con l'aggiunta di *Keyframe* lungo la Time line. Tra un keyframe e l'altro i valori vengono interpolati in modo da avere una transizione fluida tra una posa e l'altra.

All'interno del gioco queste animazione verranno richiamate dagli script nei momenti opportuni.

Capitolo 4

Gestione del gioco

In questo capitolo vengono mostrati i principali script per implementare il comportamento del personaggio e degli oggetti.

4.1 Script:

- **Script utilizzato per il menu:**
- **Buttons:**

Gestisce i bottoni del menu principale. Contieni i metodi da eseguire quando questi vengono richiamati dai bottoni.

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

public class Buttons : MonoBehaviour {

    //Riferimento alla test del bottone per la musica
    public Text musicText;

    //Riferimento alla sorgente audio per la riproduzione della musica
    public AudioSource audioSource;

    //Metodo da eseguire alla pressione del bottone per la musica nel menu principale.
    public void musicButton()
    {
     //Se sta venendo riprodotta la musica questa viene fermata.
     //Inoltre si aggiorna il testo del bottone e si salva questa scelta in modo che sia dentro le scene che al riavvio dell'ap
plicazione possa essere mantenuto questo stato.
        if(audioSource.isPlaying)
        {
            audioSource.Stop();
            musicText.text = "Music: Off";
            PlayerPrefs.SetInt("Music", -1);
        }
        //Se la musica è ferma si fa l'operazione contraria
        else
        {
            audioSource.Play();
            musicText.text = "Music: On";
            PlayerPrefs.SetInt("Music", 1);
        }
    }

    //Metodo da eseguire alla pressione del bottone "Play" nel menu principale. Avvia la prima scena del gioco.
    public void playButton()
    {
        SceneManager.LoadScene(1);
    }

    //Metodo da eseguire alla pressione del bottone "Quit" nel menu principale. Esce dall'applicazione.
    //NOTA: Non funziona in editor
    public void quitButton()
    {
        Application.Quit();
    }
```

```
    //Setta il testo del bottone della musica in maniera adeguata ("Off" se non sta venendo riprodotta la musica, "On" nel caso contrario).
    void Start () {
        if (audioSource.isPlaying)
        {
            musicText.text = "Music: On";
        }
        else
        {
            musicText.text = "Music: Off";
        }
    }
}
```

Risultato dello script:

- **Script utilizzati per la gestione generale del giocatore:**
- **PlayerBeHaviour:**

Gestisce punti e vite del giocatore e le condizioni di vittoria/sconfitta. Tiene anche aggiornata l'interfaccia grafica con lo stato del gioco. Si occupa anche di cambiare o ricaricare la scena nei momenti appropriati (vittoria o sconfitta).

```
using UnityEngine;
using System.Collections;
using UnityEngine.UI;
using System;

public class playerBehaviour : MonoBehaviour {
//Riferimento statico all'istanza presente in scena di questo script. Utilizzato per accedere velocemente ai suoi valori da qualsiasi classe.
    public static playerBehaviour instance;

//Riferimenti ai vari elementi dell'interfaccia grafica
    public Text labelPoints;
    public Text labelLife;
    public Text labelTimer;
    public Text labelMessage;

    public Image UIBackground;
//Tempo impiegato a fare i fade da/a nero
    public float fadeTime=1f;
//Curva usata nei fade
    public AnimationCurve fadeCurve;

//Punti iniziali
    public int startingPoints = 0;
//Vita iniziale
    public int startingLife = 3;
//Punti necessari al superamento del livello
    public int targetPoints = 10;

//Limite di tempo del livello
    public int timeLimit=60;
```

```
//Action contenente metodi da richiamare ogni volta che vengono modificati i punti del giocatore
    public Action pointsChanged;
//Booleano che indica se il livello è terminato. Serve a bloccare l'input
    bool ended=false;
// proprietà che si occupa di mettere in pausa il gioco aprendo l'apposito menu
bool _paused;
 public bool paused
 {
     get
   {
         return _paused;
      }

      set
      {
         _paused = value;
         if(_paused)
         { //ferma il tempo e apre il menu
            Time.timeScale = 0;
            pauseMenu.SetActive(true);
         }
         else
         { //chiude il menu e fa ripartire il tempo
            Time.timeScale = 1;
            pauseMenu.SetActive(false);
         }
      }
   }

//Tempo rimanente. Il "set" della proprietà aggiorna automaticamente l'interfaccia grafica e gestisce
la sconfitta in caso di fine del tempo
    int _timeLeft;
    int timeLeft
    {
       get{return _timeLeft; }
       set{
          _timeLeft = value;
          if (labelTimer != null) {
             labelTimer.text = _timeLeft.ToString();
          }
          if (_timeLeft <= 0) {
             lose ();
          }
       }
    }

//Punti attuali. Il "set" della proprietà aggiorna automaticamente l'interfaccia grafica e gestisce la
vittoria in caso di raggiungimento dei punti necessari
    int _points=0;
    public int points
    {
       get{return _points; }
       set{
          if (!ended)
          {
             _points = value;

             if (pointsChanged != null) {
            pointsChanged.Invoke ();
             }
```

```
                if (labelPoints != null) {
                    labelPoints.text = "POINTS: " + _points;
                }
                if (_points >= targetPoints) {
                    win ();
                }
            }
        }
    }

//Vita attuale. Il “set” della proprietà aggiorna automaticamente l’interfaccia grafica e gestisce la sconfitta in caso di fine delle vite
    int _life;
    public int life
    {
        get{return _life; }
        set{
            if (!ended)
            {
                _life = Mathf.Max (value, 0);
                if (labelLife != null) {
                    labelLife.text = "LIFE: " + _life;
                }
                if (_life <= 0) {
                    ;
                    lose ();
                }
            }
        }
    }
//Viene settata l’istanza statica all’istanze correte dell’oggetto
    void Awake()
    {
        instance = this;
    }
 void Start () {
//Si inizializzano le varie variabili ai valori desiderati
        points = startingPoints;
        life = startingLife;
        labelMessage.text = "";
        timeLeft = timeLimit;
//Si fa partire il timer
        StartCoroutine (timer ());
//Si fa un fade da nero
        StartCoroutine (fade (1, 0));
    }
  void Update () {

// gestisce la pressione del pulsante di pausa
        if(Input.GetKeyDown(KeyCode.Escape))
        {

//mette il gioco in pausa
            paused = !paused;
        }
    }

//Coroutine che ogni secondo decrementa il timer di uno
    public IEnumerator timer()
    {
        while (timeLeft > 0) {
            yield return new WaitForSeconds (1f);
            timeLeft--;
        }
    }
//Metodo che incrementa/decrementa i punti di una determinata quantità
    public static void gainPoints(int amount)
    {
        instance.points += amount;
    }
```

```
//Metodo che incrementa/decrementa le vite di una determinata quantità
    public static void gainLife(int amount)
    {
        instance.life += amount;
    }
//Metodo che gestisce la sconfitta
    void lose()
    {
//Viene settata la fine del gioco
        ended = true;
//Si ferma il timer
        StopAllCoroutines ();
//Si fa un fade a nero
        StartCoroutine (fade (0, 1));
//Si esegue la coroutine "doLose" che fa operazioni ulteriori
        StartCoroutine (doLose());
    }
//Metodo che esegue una serie di operazioni necessarie in caso di sconfitta
    IEnumerator doLose()
    {
//Si scrive a schermo "GAME OVER"
        labelMessage.text="GAME OVER";
//Si disabilita il movimento del giocatore
        var myController=GetComponent<thirdPersonMovement> ();
        myController.enabled = false;
//Si aspettano 5 secondi e poi si riavvia il livello
        yield return new WaitForSeconds (5f);
        Application.LoadLevel (Application.loadedLevel);
    }

//Coroutine che anima la trasparenza del colore di "UIBackground" da un valore di partenza ad un
valore finale utilizzando una curva. Utilizzato per i fade da/a nero
    IEnumerator fade(float startingValue, float targetValue)
    {
        float time = 0f;
        Color c = UIBackground.color;
        c.a = startingValue;
        UIBackground.color = c;
        while (UIBackground.color.a != targetValue) {

            c.a = Mathf.Lerp (startingValue, targetValue, fadeCurve.Evaluate( time / fadeTime));
            UIBackground.color = c;
            yield return new WaitForEndOfFrame();
            time += Time.deltaTime;
        }
    }
//Fondamentalmente uguale al metodo "lose" solo per la vittoria
    void win()
    {
        ended = true;
        StopAllCoroutines ();
        StartCoroutine (fade (0, 1));
        StartCoroutine (doWin ());
    }
//Fondamentalmente uguale a "doLose" solo per la vittoria
    IEnumerator doWin ()
    {
        labelMessage.text="YOU WIN";
        var myController=GetComponent<thirdPersonMovement> ();
        myController.enabled = false;
        yield return new WaitForSeconds (5f);
//Si avvia il livello successivo o in caso non ce ne fossero più si riinizia dal primo
        Application.LoadLevel ((Application.loadedLevel+1)%Application.levelCount);
    }
}
```

- **thirdPersonMovement:**

Gestisce il movimento del giocatore e della telecamera. Muovendo il mouse la telecamera ruota attorno al giocatore. Il movimento verticale della telecamera è limitato da un massimo ed un minimo. Utilizzando l'asse verticale ed orizzontale definiti all'interno del sistema di Input di Unity il giocatore si muoverà nella direzione appropriata. Inoltre se il giocatore è a terra e si preme il tasto di salto esso salterà.

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

public class thirdPersonMovement : MonoBehaviour {

//Riferimento al GameObject del giocatore
    public GameObject player;
//Riferimento al component che gestisce le animazioni
    Animator playerAnimataor;
//Riferimento al component Transform del modello del giocatore
    Transform playerTransform;
//Riferimento al component Transform dell'oggetto al quale è attaccato questo script
    Transform myTransform;
//Riferimento al CharacterController (permette di gestire sia i movimenti che le collisioni)
    CharacterController myController;

//Transform del pivot per la rotazione verticale della telecamera
    public Transform cameraPivotVertical;
//Transform del pivot per la rotazione orizzontale della telecamera
    public Transform cameraPivotHorizontal;

//Velocità alla quale si sta muovendo al momento il giocatore
    public float currentSpeed;
//Booleano che indica se il giocatore è nel mezzo di un salto
    public bool jumping = false;

//Variabile che conterrà di quanto deve ruotare la telecamera orizzontalmente
    float horizontalCameraRotation;
//Velocità di rotazione orizzontale della telecamera
    public float horizontalCameraSpeed=80f;

//Variabile che conterrà di quanto deve ruotare verticalmente la telecamera
    float verticalCameraRotation;
//Velocità di rotazione verticale della telecamera
    public float verticalCameraSpeed = 80f;

//Vector3 che conterrà la direzione attuale di movimento del giocatore
    Vector3 movementDirection;
//Velocità di movimento del giocatore
    public float movementSpeed = 2f;

//Vettore frontale rispetto alla telecamera
    Vector3 forwardVector;
//Vettore che punta alla destra rispetto alla telecamera
    Vector3 rightVector;
//I due vettori precedenti vengono usati per fare in modo che il giocatore si muova correttamente
rispetto a dove punta la telecamera
```

```
//Massima rotazione verticale raggiungibile dalla telecamera
    public float maxCameraRotation = 60;
//Minima rotazione raggiungibile dalla telecamera
    public float minCameraRotation = 0;

//Rotazione attuale della telecamera
    Vector3 currentCameraRotation;

//Variabile che controlla l'altezza del salto
    public float jumpHeight;
//Variabile che controlla la gravità al quale il giocatore è soggetto (quanto cade velocemente)
    public float playerGravity;

//Booleano che indica se il giocatore è nel mezzo di un salto
    bool isJumping;
//Valore che indica quanto il giocatore ha già saltato. Serve a farlo muovere di una certa quantità
verso l'alto e poi farlo cadere
    float jumpedHeight;

//All'inizio vengono presi i riferimenti ai vari component necessarie ed inizializzate le variabili
necessarie
    private void Awake()
    {
        playerAnimataor = player.GetComponent<Animator>();
        playerTransform = player.GetComponent<Transform>();
        myTransform = GetComponent<Transform>();
        myController = GetComponent<CharacterController>();
        movementDirection = Vector3.zero;
        forwardVector = Vector3.zero;
        rightVector = Vector3.zero;
        isJumping = false;
        jumpedHeight = 0;
    }

    void Start () {

    }

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

//Gestisce la rotazione verticale della telecamera in base al movimento verticale del mouse
        verticalCameraRotation = Input.GetAxis("Mouse Y");
        cameraPivotVertical.Rotate(Vector3.right, verticalCameraRota
tion * verticalCameraSpeed * Time.deltaTime, Space.Self);

//Limita la rotazione all'interno del massimo ed il minimo stabiliti
        currentCameraRotation = cameraPivotVertical.rotation.eulerAngles;
        if(currentCameraRotation.x>180)
        {
            currentCameraRotation.x -= 360;
        }
        currentCameraRotation.x = Mathf.Clamp(currentCameraRotation.x, minCameraRotatio
n, maxCameraRotation);
        currentCameraRotation.y = 0;
        currentCameraRotation.z = 0;
        cameraPivotVertical.localRotation = Quaternion.Euler(currentCameraRotation);

//Gestisce la rotazione orizzontale della telecamera in base al movimento orizzontale del mouse
horizontalCameraRotation = Input.GetAxis("Mouse X");
cameraPivotHorizontal.Rotate(Vector3.up, horizontalCameraRotation * horizontalCameraSp
eed * Time.deltaTime, Space.World);
//Calcola il "forwardVector" ed il "rightVector" in base alla rotazione della telecamera

        forwardVector = cameraPivotVertical.forward;
        rightVector = cameraPivotVertical.right;
        forwardVector.y = 0;
        rightVector.y = 0;
        forwardVector.Normalize();
        rightVector.Normalize();
```

```
//Muove il giocatore in base ai tasti premuti dall'utente. La direzione di movimenti dipende dalla rotazione della telecamera in quanto usa i due vettori calcolati in precedenza.
        movementDirection = Vector3.zero;
        movementDirection += Input.GetAxis("Horizontal") * movementSpeed * Time.deltaTime *rightVector;
        movementDirection += Input.GetAxis("Vertical") * movementSpeed * Time.deltaTime*forwardVector;
        if (movementDirection.magnitude != 0)
        {
            playerTransform.rotation = Quaternion.LookRotation(movementDirection);
        }
//Imposta nella macchina stati (gestita da Mecanim) delle animazioni il valore di Speed in modo che venga utilizzata l'animazione appropriata (Idle se è fermo, Walk se si muove lentamente e Run si corre)
 playerAnimataor.SetFloat("Speed", movementDirection.magnitude*10);

//Gestisce il salto.
//Se non si sta già saltando è possibile saltare
        if(!isJumping&& Input.GetButtonDown("Jump"))
        {
            isJumping = true;
            jumpedHeight = 0f;
            playerAnimataor.SetBool("Jumping", isJumping);
        }
//Se si sta saltando e ancora non si è raggiunta l'altezza desiderata il giocatore viene mosso verso l'alto con velocità pari al contrario della gravità (in pratica è come se si invertisse la gravità)
        if(isJumping&&jumpedHeight<jumpHeight)
        {
            movementDirection.y = playerGravity * Time.deltaTime;
            jumpedHeight += playerGravity * Time.deltaTime;
        }
//Altrimenti il giocatore viene spinto a terra dalla forza di gravità
        else
        {
            movementDirection.y = -playerGravity * Time.deltaTime;
            if (isJumping && myController.isGrounded)
            {
                isJumping = false;
                playerAnimataor.SetBool("Jumping", isJumping);
            }
        }

//Il giocatore viene mosso nella direzione calcolata fino a qui
        myController.Move(movementDirection);
    }
}
```

- **Script per la gestione della musica:**
- **AudioManager:**

  Serve per gestire se l'audio deve essere attivo o no.

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

public class audioManager : MonoBehaviour {

    //Legge il valore salvato dal bottone della musica.
    //Attiva o disattiva la musica a seconda del valore letto.
    void Awake () {
        if(PlayerPrefs.GetInt("Music",1)<0)
        {
            GetComponent<AudioSource>().Stop();
        }
        else
        {
            GetComponent<AudioSource>().Play();
        }
    }
}
```

- **Script per la gestione generale degli oggetti con cui si può interagire:**
- **CoinsBeHaviour:**

Gestisce il comportamento delle monete. Le anima facendole ruotare su loro stesse e muovendole in alto e in basso. Inoltre quando entrano in contatto con il giocatore vengono distrutte e il punteggio viene incrementato di un punto.

```
using UnityEngine;
using System.Collections;


public class coinBehaviour : MonoBehaviour {
//Vector3 che conterrà la posizione iniziale dell'oggetto in modo da poterla sempre sapere anche dopo che esso viene spostato.
    Vector3 startingPosition;
//Ampiezza del movimento della moneta
    public float deltaMovement = 0.5f;
//Velocità del movimento
    public float speedMovement = 1f;
//Direzione di movimento. +1 indica verso l'alto e -1 verso il basso. Utilizzata per fare in modo che la moneta si possa muovere prima verso l'alto e poi verso il basso.
    float direction=1;
//Velocità di rotazione sul proprio asse verticale
    public float speedRotation = 1f;
    // Use this for initialization
    void Start () {
//Si salva la posizione iniziale della moneta
        startingPosition = transform.localPosition;
    }

    // Update is called once per frame
    void Update () {
//sposta l'oggetto lungo il proprio asse verticale di una quantità proporzionale alla propria velocità nel verso dettato da "direction"
transform.Translate (0, speedMovement * Time.deltaTime*direction, 0, Space.Self);
//se l'oggetto ha raggiunto il limite di spostamento verso l'alto o verso il basso viene limitata la sua posizione alla massima accettabile e si inverte il verso. Il limite di spostamento è dato dalla y della posizione originale +/- il "deltaMovement".
        if (transform.localPosition.y > startingPosition.y + deltaMovement) {
            transform.localPosition = new Vector3 (startingPosition.x, startingPosition.y + deltaM
ovement, startingPosition.z);
            direction = -1;
        } else if (transform.localPosition.y < startingPosition.y - deltaMovement) {
            transform.localPosition = new Vector3 (startingPosition.x, startingPosition.y - deltaM
ovement, startingPosition.z);
            direction = 1;
        }
//Ruota la moneta attorno al proprio asse verticale di una quantità proporzionale alla velocità
        transform.RotateAround (Vector3.up, speedRotation * Time.deltaTime);
    }

//Se il giocatore entra in contatto con il Collider della moneta essa viene distrutta ed al giocatore viene assegnato un punto.
    void OnTriggerEnter(Collider other)
    {
        if (other.tag == "Player") {
            playerBehaviour.gainPoints (1);
            GameObject.Destroy (gameObject);
        }
    }
}
```

- **EnemyBeHaviour:**

Gestisce gli oggetti negativi. Quando il giocatore ci entra in contatto vengono distrutti e gli causano una penalità in punti e/o vite.

```
using UnityEngine;
using System.Collections;

public class enemyBehaviour : MonoBehaviour {
//Penalità in punti data al giocatore in caso di contatto
    public int pointsPenalty=0;
//Penalità in vite data al giocatore in caso di contatto
    public int lifePenalty=0;
//Booleano che decide se l'oggetto deve essere distrutto quando entra in contatto con il giocatore
oppure no
    public bool destroyObject = true;

    // Use this for initialization
    void Start () {

    }

//Se il giocatore entra in contatto con il Collider dell'oggetto allora gli viene assegnata una penalità in
vite e/o punti. Inoltre l'oggetto viene distrutto se richiesto.
    void OnTriggerEnter(Collider other)
    {
        if (other.tag == "Player") {
            playerBehaviour.gainPoints (-pointsPenalty);
            playerBehaviour.gainLife (-lifePenalty);
            if (destroyObject) {
                GameObject.Destroy (gameObject);
            }
        }
    }
}
```

- **Script per il menu di pausa:**

- **PauseBottons:**

Contiene i metodi chiamati dai bottoni del menu pausa.

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

public class PauseButtons : MonoBehaviour {
    //Metodo da chiamare alla pressione del pulsante "Resume"
    //Riprende il gioco
    public void resumeButton()
    {
        playerBehaviour.instance.paused = false;
    }
//Metodo da chiamare alla pressione del pulsante "Quit" nel menu di pausa
//Torna menu principale (la scena 0).
    public void quitButton()
    {
        SceneManager.LoadScene(0);
    }
}
```

Risultato dello script:

- **Script per far ruotare i serpenti del secondo livello:**
- **RotateAroundAxis:**

Ruota un oggetto attorno al suo asse verticale ad una determinata velocità.

```
using UnityEngine;
using System.Collections;

public class rotateAroundAxis : MonoBehaviour {
//Asse attorno al quale ruotare
    public Vector3 axis = Vector3.up;
//Velocità di rotazione
    public float speed=10f;

//Viene effettuata la rotazione
    // Update is called once per frame
    void Update () {
        transform.Rotate (axis, speed * Time.deltaTime);
    }
}
```

- **Script per aprire le porte del secondo e terzo livello:**
- **AnimateAtPoins:**

Esegue un'animazione quando il giocatore raggiunge un determinato numero di punti. Utilizzato per animare le porte del secondo e del terzo livello.

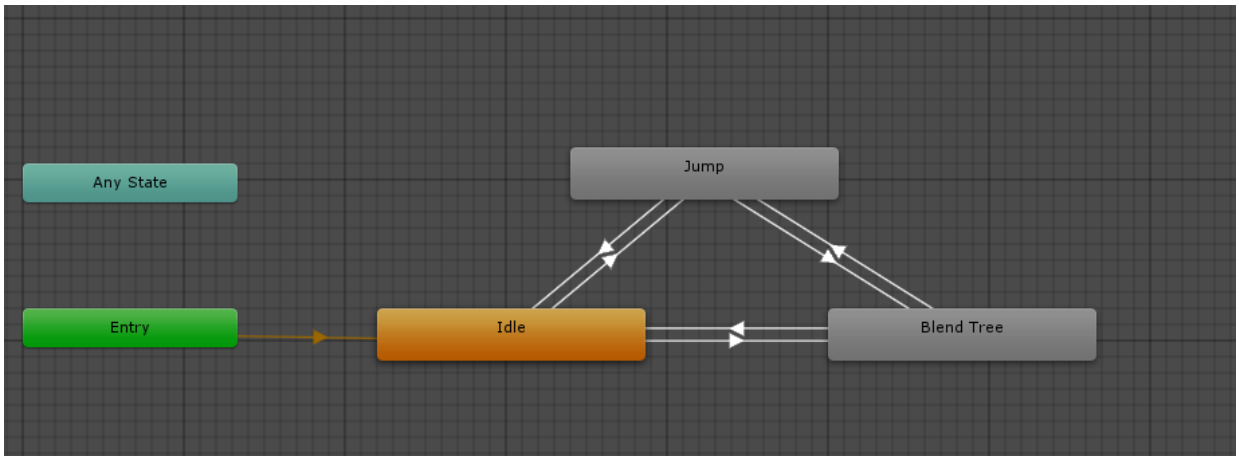
```
using UnityEngine;
using System.Collections;
public class animateAtPoints : MonoBehaviour {
//Quando il giocatore arriva a questa quantità di punti verrà iniziata l'animazione
public int targetPoints;
//Booleano che indica se l'animazione è già stata fatta in modo da non ripeterla
bool done=false;

void Start () {
//Alla creazione dell'oggetto registra il suo metodo "checkPoints" nell'Action "pointsChanged" del giocatore in modo che il metodo "checkPoints" sia richiamato ogni volta che i punti del giocatore cambiano
playerBehaviour.instance.pointsChanged += checkPoints;}
void checkPoints(){
//Ogni volta che cambia il punteggio del giocatore viene controllato se questo è sufficiente a far partire l'animazione e in caso affermativo l'oggetto viene animato.  Attraverso il booleano "done" si evita che l'animazione venga ripetuta più volte.
if (!done && playerBehaviour.instance.points >= targetPoints)
{           done = true;
             animate ();}
}
void animate()
{
//Fa partire l'animazione
              GetComponent<Animation> ().Play ();
   }
```

- **Script per far muovere i treni del terzo livello:**
- **Trainmovement:**

  Gestisce il movimento dei treni sui binari.

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

public class trainMovement : MonoBehaviour {

    //Posizione minima del treno (inizio).
    public Vector3 minPosition;
    //Posizione massima del treno (fine).
    public Vector3 maxPosition;

    //Durata dello spostamento da inizio a fine.
    public float duration;

    //Contatotre per lo scorrere del tempo.
    float time;

    //Componente transform dell'oggeto
    Transform myTransform;

    void Start () {
        //Recupera il componente transform.
        myTransform = GetComponent<Transform>();
    }

    void Update () {
         //Incrementa il contatore del tempo di: tempo transcorso dall'ultimo frame diviso la durata dell'a
nimazione.
        time += Time.deltaTime / duration;

        //Forza il contatore del tempo a restare compreso tra 0 e 1.
        while(time>1)
        {
            time -= 1;
        }

        //Interpola linearmente tra inizio e fine.
        myTransform.position = Vector3.Lerp(minPosition, maxPosition, time);
    }
}
```

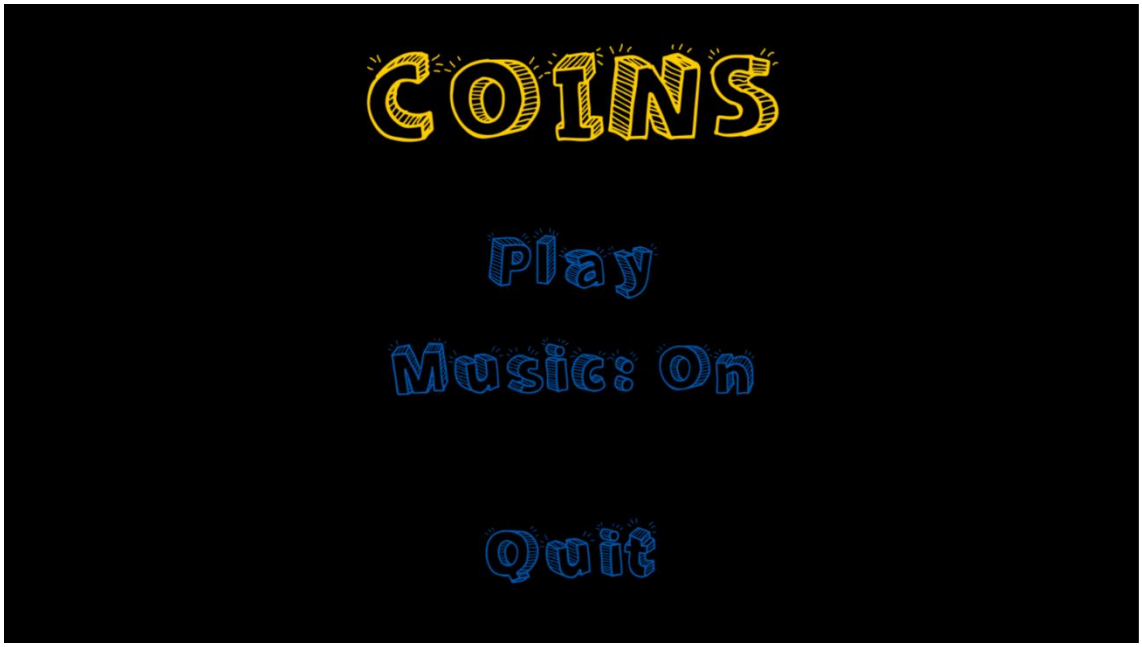
Questa **macchina a stati** gestisce le transazioni tra le varie animazioni. È controllata dallo script del movimento del giocatore "*thirdPersonMovement*".



Capitolo 5

Screenshot del gioco

In questo capitolo vengono mostrati alcuni *screenshot* relativi al gioco.

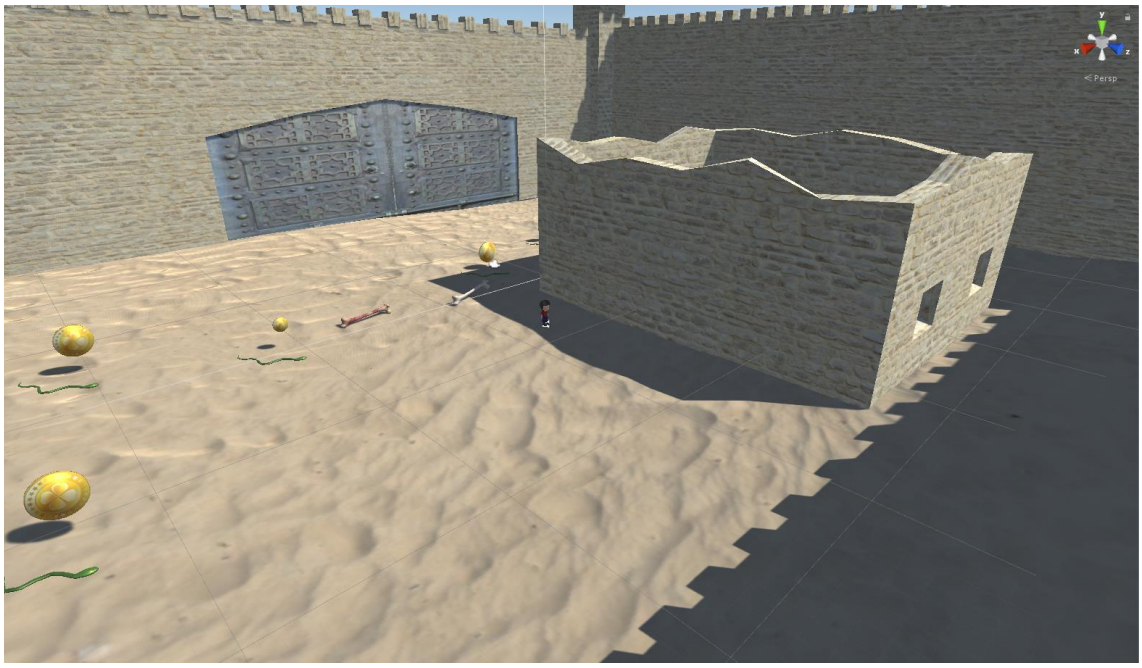


Questa è la scherma del menu principale, sono presenti tre bottoni: “Play” per iniziare la partita, “Music: On” il bottone per attivare o disattivare la musica, in caso di musica disattivata avremo “Music: Off”, e “Quit” per uscire.

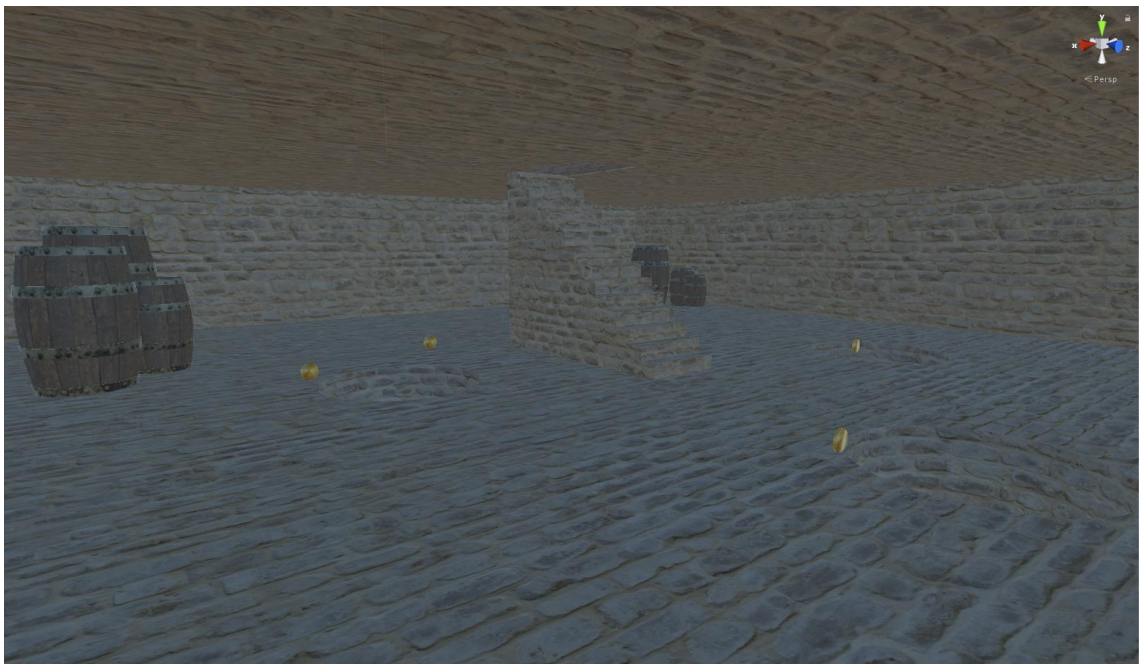
Questi screenshot mostrano il primo ambiente visto da diversi punti di vista.

Questi sono gli alberi presenti nella scena:

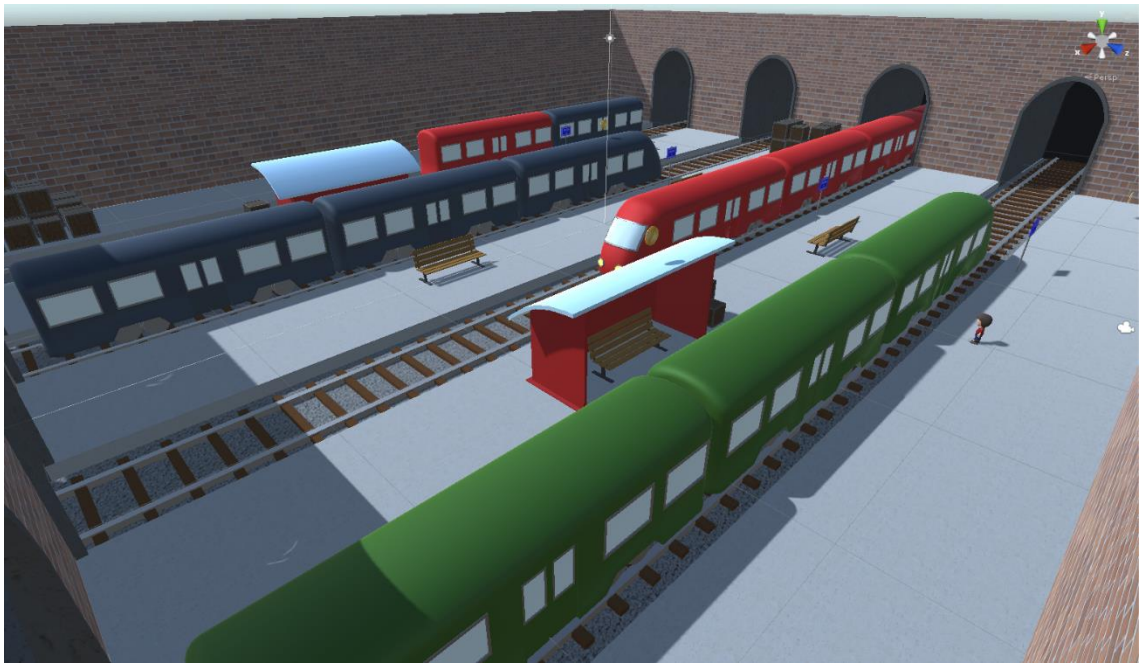
Questi screenshot mostrano il secondo ambiente:



Questo è il sotterraneo:

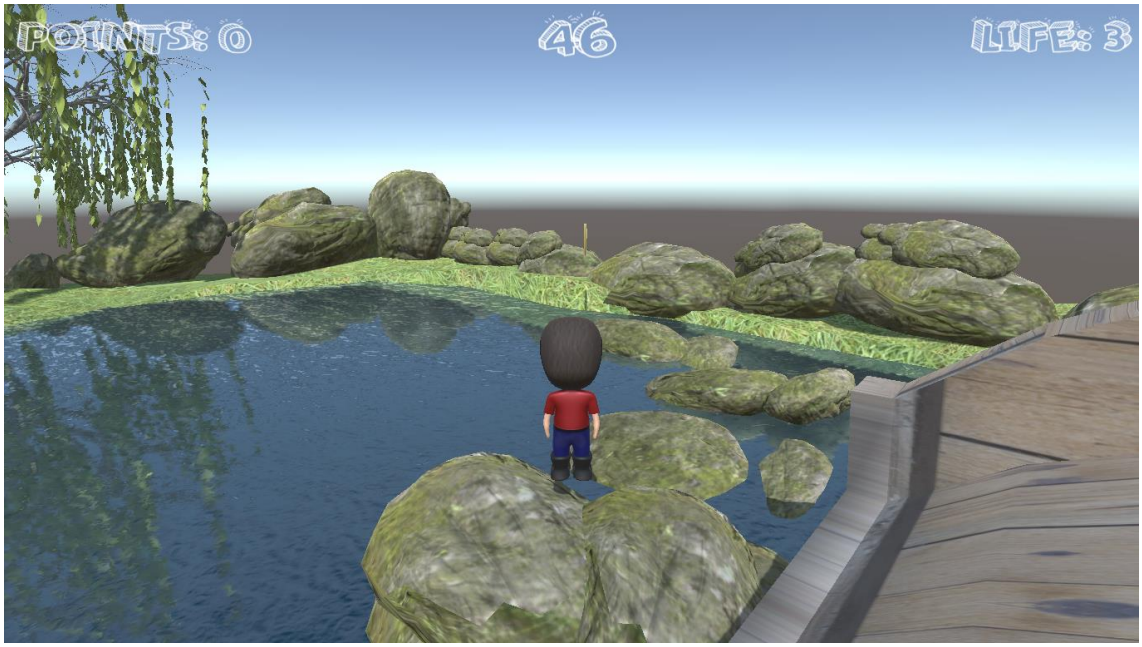


Questi screenshot mostrano l’ultimo ambiente:



Il giocatore all'inizio del primo livello si trova in questa situazione:in alto a sinistra nella schermata trova la voce "POINTS" che indica il numero di monete che il giocatore ha raccolto, al centro trova il timer che parte da 60 secondi e va a decrementarsi, alla destra trova la voce "LIFE" che indica le vite a sua disposizione e anche queste vengono decrementate se si prendono determinati oggetti nella scena.

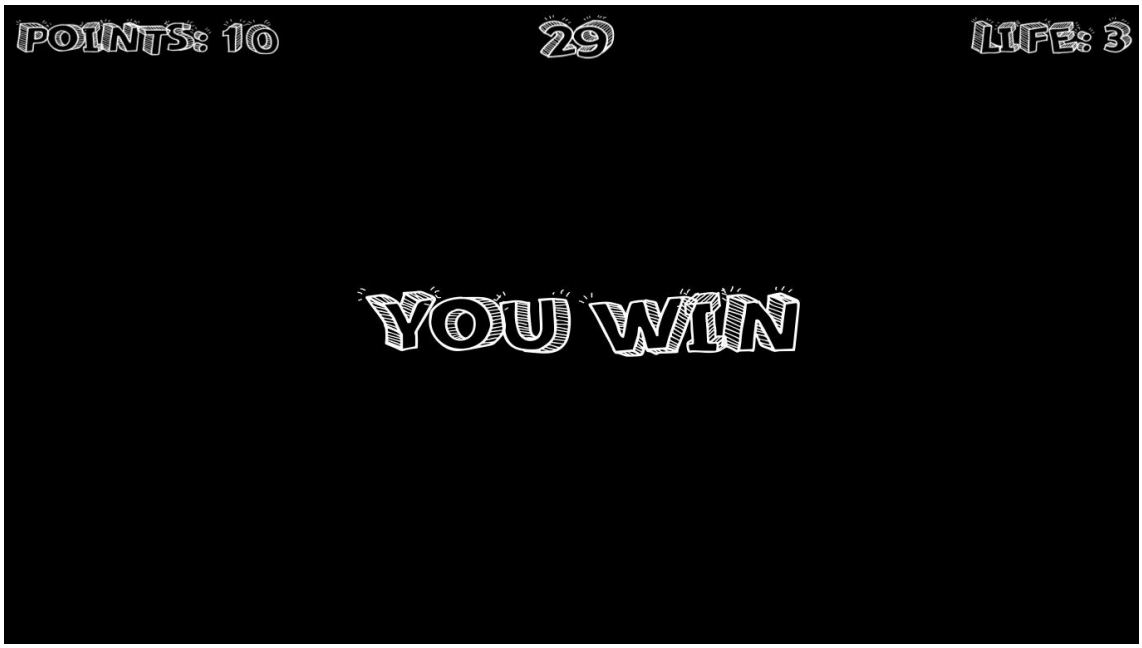
Questa è la visuale della scena dal laghetto.



In queste due schermate che seguono viene mostrato il momento in cui il giocatore si trova davanti ad un fungo che toglie una vita e cosa succede se avviene la collisione con esso.

Questa schermata ritrae il momento in cui in giocatore cade nel laghetto e questa azione fa terminare istantaneamente il gioco, in quanto vengono decrementate 3 vite dal contatore. Quindi la schermata si annerisce e viene mostrata a video la scritta “GAME OVER”, il gioco poi partirà dall’inizio resettando il time e il punteggio.

Questa è la schermata che appare quando il giocatore è riuscito a raccogliere le 10 monete necessarie per passare al secondo livello, nei 60 secondi disponibili.



Quando il giocatore passa al secondo livello si trova in questa situazione:

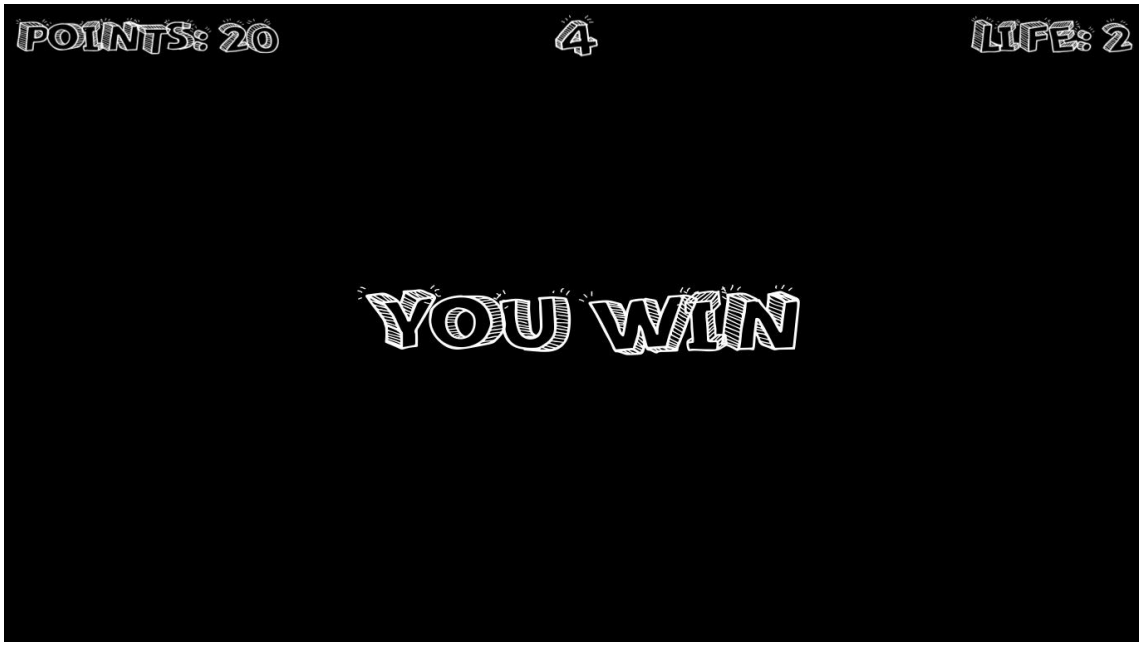
Possiamo vedere che il giocatore parte da un punteggio pari a 10, in quanto vengono salvate le monete raccolte nel primo livello.

Questo screenshot rappresenta un punto del secondo livello in cui il giocatore si trova davanti ad una porta chiusa che riuscirà ad aprire solo al raggiungimento di 15 monete come nella situazione sottostante.

Il giocatore a questo punto ha aperto la botola che porta nel sotterraneo raggiungendo un punteggio di 17 monete.

Quest'immagine ritrae lo scenario del sotterraneo con la presenza di botole aperte, dove il giocatore si deve avvicinare per poter raccogliere le monete. Se il giocatore cade all'interno di queste il livello termina.

Questo è lo screenshot della vittoria del secondo livello.



Questa immagine ritrae la scena del terzo livello, possiamo vedere il punteggio pari a 20 monete. Inoltre la scena ritrae il momento in cui sta passando un treno nel primo binario.

Mentre in questo screenshot possiamo vedere il momento in cui il treno nel primo binario sta uscendo dalla scena e invece è in arrivo un altro treno nel secondo binario.

In questa situazione vengono mostrati alcuni punti importanti del gioco: la presenza di scatole che in diversi punti della scena nascondono delle monete e la presenza di una moneta all'interno della galleria di un binario.

Questo è il menu che appare quando il giocatore preme “Esc” e allora il gioco viene messo in pausa. A questo punto il giocatore può decidere se riprendere il gioco o tornare al menu principale premendo “Quit” e così poi uscire dal gioco.

Questa scena ritrae il giocatore che si trova davanti al vagone del treno con le porte aperte, salendo nel treno trova l’ultima moneta presente nel livello e quindi una volta che il giocatore riesce ad aprire le porte del treno avendo 29 monete, il gioco è terminato.

Conclusioni

Il lavoro necessario per creare tutto quello serve per la realizzazione di un gioco è sicuramente lungo ed impegnativo. Infatti per ottenere un buon risultato è necessario avere delle basi teoriche sugli argomenti della Computer Graphics e avere inoltre buon occhio per creare dei buoni risultati dal punto di vista estetico.

Inoltre una buona parte del lavoro consiste anche nella precisione nel creare l'effetto desiderato sia nella modellazione degli oggetti che nell'applicazione dei materiali; questo infatti richiede spesso molto tempo perché non sempre il primo risultato è perfetto in base all'idea che avevamo in partenza.

Nella tesi sono stati trattati i punti più importanti della Computer Graphics che sono fondamentali per la realizzazione di un videogioco; argomenti quali il *3D Modeling*, la programmazione delle logiche di gioco e l'audio.

Per avere dei risultati finali di buon livello, bisogna avere molta esperienza nella pratica della modellazione e della logica di gioco. Infatti per diventare esperto nella realizzazione di videogiochi e per crearli in minor tempo bisogna avere alle spalle la realizzazione di diversi progetti.

Come spiegato nel primo capitolo, estremamente importante è farsi prima un'idea di quello che si vuole realizzare, facendo disegni a mano su l'effetto che vorremmo ottenere. Questo è importante perché può aiutare molto il lavoro che si deve realizzare.

Bibliografia

[1]. Blender foundations: The Essential Guide to Learning Blender 2.6, Roland Hess, 2010

[2]. Learning Blender, Pearson Education, Oliver Villar, 2014

[3]. Ryan Henson Creighton,“Unity 3D Game Development by Example Beginner's Guide”, PACKT Publishing, settembre 2010.