Realtà virtuale e Aumentata

Progetto VR con Unity

Creato da:

Matteo Breganni 869549

Indice

Introduzione

La seguente relazione descrive il mio lavoro su questa applicazione VR. Nelle sezioni iniziali vengono mostrati gli aspetti introduttivi e più generali, nelle sezioni successive invece vengono descritte tutte le sezioni di gioco in sequenza. Ciascuna sezione di gioco ha un'introduzione che è sufficiente per avere un'idea generale del contenuto e di cosa bisogna fare per proseguire. A seguire ciascuna introduzione c'è una descrizione più dettagliata dei contenuti, dove mostro ad alto livello il mio lavoro evitando di entrare troppo negli aspetti tecnici e descrivendoli solo dove li ritengo necessario. Questi paragrafi includono anche i riferimenti agli script e agli asset creati, modificati o importati.

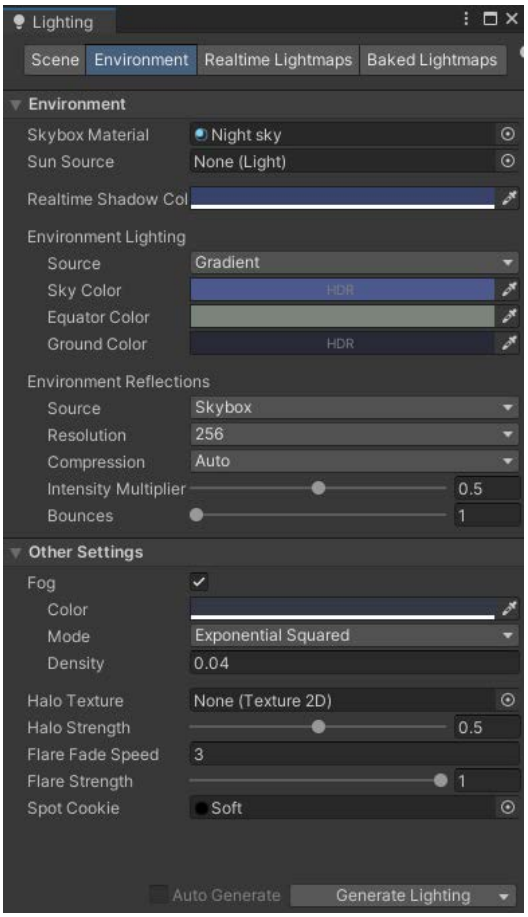
Ambientazione e terreno

L'applicazione VR che ho creato è un gioco a tema horror, l'ambientazione è notturna, con poca luce. Le interazioni cercano quindi di mantenere a grandi linee questo tema.

Il terreno è stato creato tramite il tool interno di Unity e utilizzando i packages "Conifers [BOTD]" per i modelli degli alberi e dell'erba, e "Grass And Flowers Pack 1" per ulteriori modelli dell'erba.

Ho quindi modellato l'altezza del terreno per rispecchiare la mia idea di gioco, utilizzando questi asset per creare una strada sterrata (utilizzando diversi materials di terreno, anche modificati, per renderlo meno piatto e banale) circondata da una fitta foresta.

Per rendere l'ambiente notturno e cupo, ho aggiunto uno skybox appropriato (riducendo la quantità di luce proveniente da esso), della nebbia e dei suoni in loop per simulare un clima di vento, una foresta abbastanza silenziosa, senza vita.



Interazioni

Nelle seguenti interazioni faccio riferimento ai comandi per mouse e tastiera, dato che questi potrebbero essere mappati in modi diversi in base al visore che viene utilizzato.

Il movimento nel gioco è fatto tramite i pulsanti “WASD”, quindi in modo semplice e continuo. È inclusa un’area di gioco, alla fine del gioco, dove è necessario utilizzare il metodo di teleport con i raycast interactors.

È stato aggiunto un sistema di collisioni per l’utente che utilizza il componente “Character Controller” per limitare i movimenti dell’utente se necessario, ma soprattutto per supportare una verticalità che avrei voluto dare al gioco. Di conseguenza, se il terreno presenta una piccola salita o discesa, il controller gestirà automaticamente la posizione dell’utente, evitando che questo passi attraverso agli oggetti di gioco.

Per quanto riguarda le interazioni vengono utilizzate quelle di tipo “Direct grab” (tramite pulsante G di qualsiasi mano) per la maggior parte, per esempio per aprire uno sportello, muovere una leva, lanciare qualcosa. Vengono utilizzate anche quelle tramite ray interactor per interagire con le varie UI (quella nella scena iniziale, quella dell’User Interface e quella dell’ultimo minigioco), per il movimento dei quadri nel puzzle della prima stanza della casa e per il teletrasporto nell’ultimo minigioco.

Per le mani ho utilizzato il modello di Ocolous che ho trovato su internet (aggiungendo materials personalizzati), che comprende anche alcune animazioni (per quando viene premuto il pulsante “grab” oppure “pinch”), per rendere ancora più esplicita l’interazione. Ho poi implementato uno script “HandHoverColor” che modifica il materiale delle mani a runtime, cambiandolo con un colore leggermente diverso quando la proprietà “Hover” è attiva, per rendere esplicita e semplice l’interazione con i grab interactors.

I ray interactor hanno una lunghezza dinamica, vengono allungati se necessario quando puntano ad un oggetto con cui si può interagire. È anche stato impostato un cambio di colore verso il blu quando si sta mirando ad un oggetto interactable.

Vengono utilizzate le “Interaction Layer Masks” per filtrare quali oggetti è possibile interagire con il metodo Direct e quali con il metodo Raycast.

Ovviamente le interazioni sono state create usando il package di Unity “XR Interactions Toolkit”, importandolo e settandolo da zero.

User Interface

È possibile mostrare o nascondere l’user interface in qualsiasi momento, tramite il pulsante B (Primary Button) della mano sinistra. Ogni UI Page (prefab creato) include un titolo e una descrizione che mostrano informazioni sulla sezione di gioco raggiunta, con dei suggerimenti su come proseguire. È presente anche un pulsante X che permette di chiuderla tramite ray interactor volendo, ma è possibile chiuderla semplicemente premendo B di nuovo.

Le schede mostrate al suo interno cambiano quindi dinamicamente, attivate da funzioni e da trigger zones, di modo da mostrare sempre informazioni sulla sezione di gioco attuale.

Ho quindi scritto lo script “CanvasVisibility” per rendere l’UI non visibile ad inizio del gioco (impostarla come invisibile manualmente dalla gerarchia dava problemi), lo script “InputButtonPressed” per mostrare l’UI quando viene premuto il pulsante, e lo script “ChangeCanvasInstruction” per passare automaticamente alla prossima scheda, grazie ad un numero incrementato internamente allo script.

Narrazione

Oltre alla user interface, ho voluto implementare un sistema di narrazione, con l’obiettivo di rendere più chiari gli obiettivi di gioco all’utente e aumentare l’immersione di gioco (evitando quindi che l’utente debba aprire l’UI di continuo per capire come proseguire).

Durante il gioco verranno riprodotti automaticamente degli audio di narrazione che ho creato registrando la mia voce e utilizzando il tool AI di ElevenLabs. Ho preferito questa opzione rispetto a quella text to speech perché registrando la mia voce, riuscivo a dare più emozione all’AI che quindi sembrava meno robotica. Il risultato è ovviamente non perfetto, sarebbe molto meglio avere un attore capace, ma è comunque un tipo di implementazione che ho voluto provare ad utilizzare.

Questi audio sono riprodotti automaticamente all’ingresso di trigger zones, oppure in specifiche funzioni che vengono eseguite al completamento o durante eventi di gioco specifici.

Ho scritto anche la funzione “PlayTwoAudiosInARow” nel caso dovessi assegnare ad una singola trigger zone la riproduzione di due audio di fila.

Le trigger zones sono generalizzate tramite lo script “Trigger” e il prefab “Trigger Zone” che ho creato.

Tutti gli audio di narrazione sono contenuti nella sezione “Narration Audios” della gerarchia.

Aree di gioco

Nelle seguenti sezioni saranno descritte tutte le aree di gioco in sequenza, dalla prima incontrata nelle 3 scene, fino all'ultima. Per avere un'idea generale dei contenuti è sufficiente leggere la parte introduttiva di ciascuna sezione (il paragrafo subito dopo la prima immagine), dopo la quale saranno riportati in maggiore dettaglio i contenuti della sezione, descrivendo il lavoro fatto senza entrare troppo in dettagli tecnici a meno che sia ritenuto necessario. In ogni caso saranno presenti tutti i riferimenti ai GameObject e agli Script creati, modificati e utilizzati all'interno del progetto.

Menu Iniziale (scena introduttiva)

La prima scena contiene una copia dello stesso terreno, con alcune piccole modifiche per creare quest'area ristretta dove l'utente può muoversi.

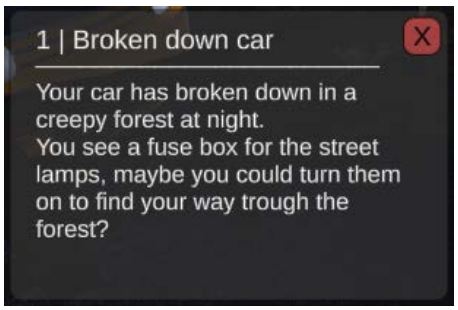
All'interno si può trovare una semplice UI che contiene alcune istruzioni sulle interazioni e come simularle tramite mouse e tastiera (non ho aggiunto riferimenti per i visori dato che suppongo siano diversi per ciascun visore), e un pulsante "Play" che permette di cambiare scena utilizzando la mia funzione "SceneChanger", iniziando il gioco.

Punto di partenza (scena di gioco)

L'utente si ritrova in una strada sterrata in una foresta. Da un lato c'è una staccionata che blocca la strada, dietro la quale c'è la sua auto con le luci accese, un vetro rotto e il cofano rialzato da cui esce fumo ad indicare che l'auto ha avuto un problema. Questo, quindi, fa capire all'utente che non deve interagire con l'auto stessa, ma deve seguire la strada, dal lato della staccionata in cui si ritrova l'utente.

Quando l'utente arriva in quest'area (automaticamente, visto che è il punto di partenza del gioco), vengono riprodotti due audio di narrazione introduttivi, per aiutare l'utente a capire in che situazione si ritrova.

L'UI riporterà le seguenti informazioni:



Staccionata

La staccionata utilizza il modello del package "RPG Medieval Props" (uno dei pochi asset del pack compatibili con la built-in render pipeline che ho usato) ed è stata creata creata manualmente alternando 2 prefab diversi di staccionata, allungandone e inclinandone uno sul pezzo di staccionata "rotto", per renderla un po' più interessante.

Ho poi deciso di semplificare le collisioni con l'utente andando a sovrapporre un box collider semplice, per evitare che l'utente possa andare in quel lato della mappa dove c'è l'auto.

Automobile

L'automobile è un modello importato dal package "Low Poly Destructible 2 Cars no. 8", utilizzando gli asset disponibili per personalizzare la macchina (si possono cambiare gomme, paraurti, etc). La cosa importante è che l'auto di questo package non è un oggetto singolo ma è l'insieme delle sue parti, il che lo rende molto personalizzabile e, come da nome del package, distruttibile. È quindi semplice spostare alcune sotto-parti per (nel mio caso) tenere una portiera aperta, avere il cofano aperto, impostare una texture di vetro rotto ad uno dei vetri.

Le due sorgenti particle system per il fumo che esce dal motore sono state create manualmente da me, di modo che appunto assomigliassero a del fumo, seguendo anche le collisioni superiori con il cofano rialzato e la direzione del vento globale. Per creare un particle system realistico ho quindi lavorato sulle sezioni emission, shape, color over lifetime, size over lifetime, external forces, collision e render.

Sono anche state aggiunte manualmente due spot lights su ciascun faro anteriore, di modo da rendere la scena ancora più realistica (dato che la luce interagisce con la staccionata e gli alberi) ed aggiungere una sorgente di luce. Ho aggiunto anche due point lights e due effetti halo sui fari, per dare appunto l'impressione che i fari siano accesi. Altrimenti ci sarebbe una sorgente di luce proveniente dai fari che però sarebbero scuri, spenti.

È anche presenta una sorgente di audio 3d che riproduce un basso rumore dell'auto in idle, per aggiungere più atmosfera.

Accensione luci

Poco distante dal punto di partenza, il giocatore può trovare un armadietto attaccato al primo palo della luce (da cui vengono emesse delle scintille per aiutare l'utente a notarlo) . Dovrà quindi interagire con l'armadietto aprendo l'anta (tramite grab interaction) e alzando la leva per accendere le luci sulla strada. Una volta accese le luci verranno riprodotti altri effetti visivi come scintille da tutti le lampade, suoni sia dalle lampade che dal fuse box, e l'utente potrà proseguire nel percorso. Vengono anche modificati delle parti del fuse box a runtime, mentre la leva viene alzata, come l'indicatore del voltaggio e il led che mostra se le luci sono accese o spente.

Fuse box

L'armadietto è stato importato dal package "Electical shield".

Tramite il mio prefab "Sparks Intermittent" vengono riprodotte delle scintille provenienti dal retro del fuse box in modo irregolare. Queste aiutano l'utente a notare la presenza dell'armadietto.

Come nel particle system del fumo, anche questo è stato creato da me, impostando tutte le sezioni necessarie (come anche il fatto che le scintille emettono luce, cosa che si nota molto quando le luci sulla strada non sono ancora state accese). È anche associato lo script "PlaySoundSparks" scritto da me, che riconosce quando viene effettuata un'emissione di scintille, e in base al numero di scintille emesse riprodurrà in modo casuale uno dei due suoni di scintille più corti oppure uno dei due suoni più lunghi, presenti nei GameObject duplicati con il mio prefab "Audio Source Zap".

Interazioni con sportello e leva

Ho aggiunto all'anta un Hinge Joint, mentre alla serratura ho aggiunto un XR Grab Interactable con un box collider e un fixed joint. In questo modo l'utente può aprire l'anta andando ad afferrare la sua parte di destra. Il movimento è però limitato nella sua rotazione, evitando che l'anta venga ruotata oltre ai suoi limiti naturali dato che ciò distruggerebbe il realismo.

Al fuse box è quindi associato lo script “FuseBoxOpened” che contiene una condizione che controlla quando l’armadietto è stato aperto, giudicando la rotazione dello sportello. Di conseguenza, se lo sportello viene richiuso, questo sarà considerato. Il motivo della presenza di questo script è quello di abilitare e disabilitare il secondo Grab Interactable presente nel fuse box, ovvero quello della leva. Se questo fosse sempre attivo, infatti, ci sarebbe il rischio che l’utente afferri direttamente la leva quando il fuse box è ancora chiuso.

La leva funziona in un modo simile, utilizzando Hinge Joint, Fixed Joint, Grab Interactable e Capsule Collider per restringere il campo di rotazione della leva, lasciandola anche reagire alla gravità nel caso venga lasciata a metà.

Reazioni al movimento della leva

La leva ha assegnato lo script “LightsLeverOn” che è incaricato di controllare quando l’angolazione della leva raggiunge il valore massimo permesso, e gestirne le conseguenze.

La leva viene quindi bloccata nell’angolazione massima, rimuovendo anche il collider, di modo che non si possa più interagire con essa.

A questo punto viene anche riprodotto un suono 3D in loop di un ronzio proveniente dall’armadietto, per indicare che è attivo. Per lo stesso motivo viene anche spenta la luce rossa e accesa quella verde all’interno del fuse box. Inoltre, è possibile notare come la lancetta dei volts dentro al fuse box reagisca in tempo reale al movimento della leva, aumentando e diminuendo.

Le luci dei 4 pali della luce vengono quindi accese, sostituendo al modello della lampada spenta, quello con la lampada accesa. I prefab dei pali della luce sono stati importati dal package “Pillars Pack” a cui ho apportato alcune modifiche per appunto creare la versione della lampada con la luce e quella senza (con material diversi, light e halo).

Ai pali della luce ho aggiunto anche il mio prefab “Sparks on command”, variazione del mio prefab “Sparks Intermittent” usato precedentemente, da cui vengono emesse le scintille solo al momento dell’accensione delle lampade, insieme alla riproduzione di un suono “Zap” in quel momento e di un altro suono in loop 3D proveniente dalle lampade di tipo “humming”.

La differenza tra i prefab “Sparks Intermittent” e “Sparks on command” è che il primo riproduce le scintille in modo intermittente, come spiegato precedentemente, mentre invece il secondo le riproduce singolarmente per una durata specifica, ovvero quando vengono accese le luci in questo caso.

È degno di nota il fatto che lo script contiene una funzione “startAndPauseAudio()” che viene chiamata nello start. Ho dovuto aggiungere questa funzione, in relazioni ai suoni di humming delle lampade, dato che altrimenti la riproduzione di tutti questi suoni comportava un freeze, secondo me causato dalla lunghezza dei loop che dovevano esser caricati in memoria nel momento dell’accensione delle luci. Usando questo metodo invece, vengono caricati in memoria nello start e semplicemente riprodotti più tardi.

Infine, l’accensione delle luci rimuove anche la barriera invisibile che ho aggiunto con il mio prefab “Invisible Barrier” che impedisce all’utente di proseguire finché non ha acceso le luci.

Percorso

Nella parte successiva del percorso è presente una Trigger Zone (prefab creato) che riproduce un suono cupo (come se ci fosse un mostro o un animale in lontananza) 3D proveniente dalla foresta, per aggiungere all’atmosfera horror (tramite lo script PlayMonsterGrowls).

Roccia con ombra del mostro

Il giocatore prosegue nel percorso, raggiungendo un palo della luce caduto la cui lampada continua ad accendersi/spegnersi e ad emettere scintille. Avvicinandosi, l’ombra di un mostro viene mostrata su una roccia, fino a quando l’ombra sembra correre verso il giocatore.

Palo della luce caduto

Il quarto palo della luce è l’ultimo che viene acceso dalla leva azionata precedentemente (i successivi sono scollegati, data la rottura del cavo che li collega). Quando la leva viene azionata, la lampada inizia ad emettere le scintille tramite il perfab intermittente. A questo punto lo script “IntermittentLightFromSparks” è in ascolto e controlla quando l’emissione delle scintille è sufficientemente grande, per disattivare la luce e riattivarla successivamente, creando quindi l’effetto di intermittenza anche sulla luce stessa.

Jumpscare

Quando il giocatore si avvicina in questa zona, attiva una Trigger Zone che mette in pausa quest'ultimo script in favore di "MonsterShadowFlashingLightTrigger" che gestisce in modo temporaneo il flickering della luce di modo che sia controllato in modo preciso (il flickering della luce viene riprodotto su tutte le lampade, non solo quella caduta, per dare l'impressione di un "evento paranormale"). L'intervallo tra la continua accensione e spegnimento delle luci è randomizzato all'interno di un range. Nello script viene tenuta traccia della lunghezza di ciascuna parte dell'evento rispetto a quella prevista, di modo da cambiare il comportamento del flickering quando il mostro inizia a correre, di modo da rendere consistente quella fase, ed evitare che lo spegnimento casuale della luce in quel momento nasconda l'ombra che inizia a correre.

Viene quindi mostrata l'ombra del mostro (il modello è impostato come invisibile) e tramite l'animator controller "Monster Jumpscare Animator Controller" da me creato, viene riprodotta inizialmente l'animazione "Crouching" e poi quella di "Crouch to Sprint" che mostra l'ombra correre verso l'utente per creare l'effetto di jumpscare. In questo caso il controller è semplice, riproduce le animazioni ad una velocità e in un intervallo ben definito.

Questa funzione utilizza le Coroutine di unity per introdurre dei delay quando necessario, e proseguire l'esecuzione nei prossimi frames. Questo sistema è usato in diversi scripts.

Il modello e le animazioni del mostro sono stati scaricati dal sito "Mixamo".

Insieme a questi effetti di luce e del mostro, vengono riprodotti anche diversi audio di flickering della luce, di narrazione, di suoni provenienti dal mostro e altri effetti horror.

Al termine dell'evento, il palo della luce riprenderà con gli effetti precedenti di scintille e luce intermittente.

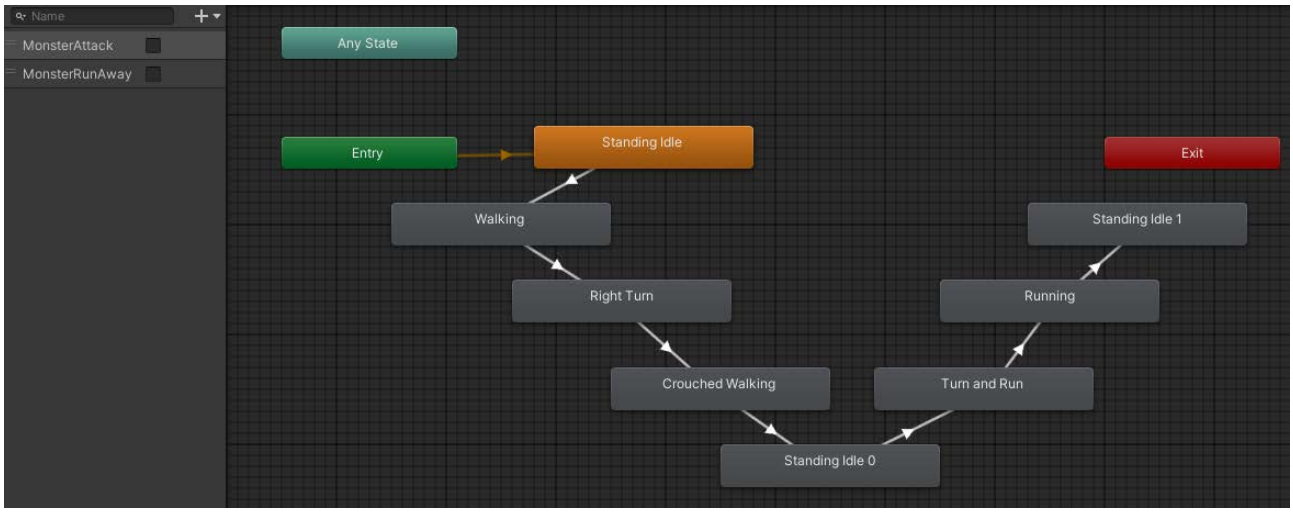
Questa e le altre pietre presenti nella scena provengono dal package "Rocks Package".

Attacco del mostro

Il giocatore prosegue nel percorso finché vede una figura umana nella distanza. Il giocatore cerca di catturare la sua attenzione, gli chiede aiuto, ma quando il soggetto si gira e inizia ad avvicinarsi, il giocatore si rende conto che non è una persona ma un mostro. Il mostro continua ad avvicinarsi finché un fulmine lo spaventa, facendolo scappare via.

Animazioni del mostro e fulmine

L'evento è azionato da una Trigger Zone, che fa partire la prima animazione dell'animator "Monster Attack Animation Controller", cambiando il parametro "MonsterAttack" a true tramite lo script "MonsterAttack", che riproduce anche i versi del mostro.



Tramite l'exit time, viene quindi riprodotta l'animazione di walking di modo che il mostro arrivi nel punto corretto della strada, seguita dall'animazione che lo fa girare a destra e da quella che lo fa avvicinare all'utente, con un'animazione dove il mostro è abbassato, per renderlo più spaventoso.

A questo punto nel successivo stato viene azionato lo script “MonsterAttackLightning” che attiva il fulmine importato dal package “Lightning Bolt Effect For Unity” (non è un granché ma non ho trovato di meglio) insieme al suono del fulmine e a quelli del mostro che grida scappando. Il fulmine viene quindi disattivato dopo 0.7 secondi utilizzando una coroutine, che disattiva anche la barriera che impediva all’utente di proseguire troppo avanti (di modo da mantenere il corretto punto di vista dell’evento).

Seguono quindi le prossime animazioni, dove il mostro si gira e scappa via dal fulmine. La funzione “MonsterAttackEnd” viene quindi attivata dall’ultimo stato dell’animator e disattiva il mostro.

La casa e il cane da guardia

Il giocatore arriva al termine del percorso, raggiungendo una casa circondata da una staccionata, protetta da un cane da guardia. Deve quindi trovare un modo per rendere il cane amichevole prima di poter entrare nel recinto. Trova quindi un pezzo di carne su un tavolo lì vicino, e lo lancia al cane. Il cane sarà quindi amichevole l’ora in poi, e il recinto verrà aperto.

Comportamento e movimento del cane

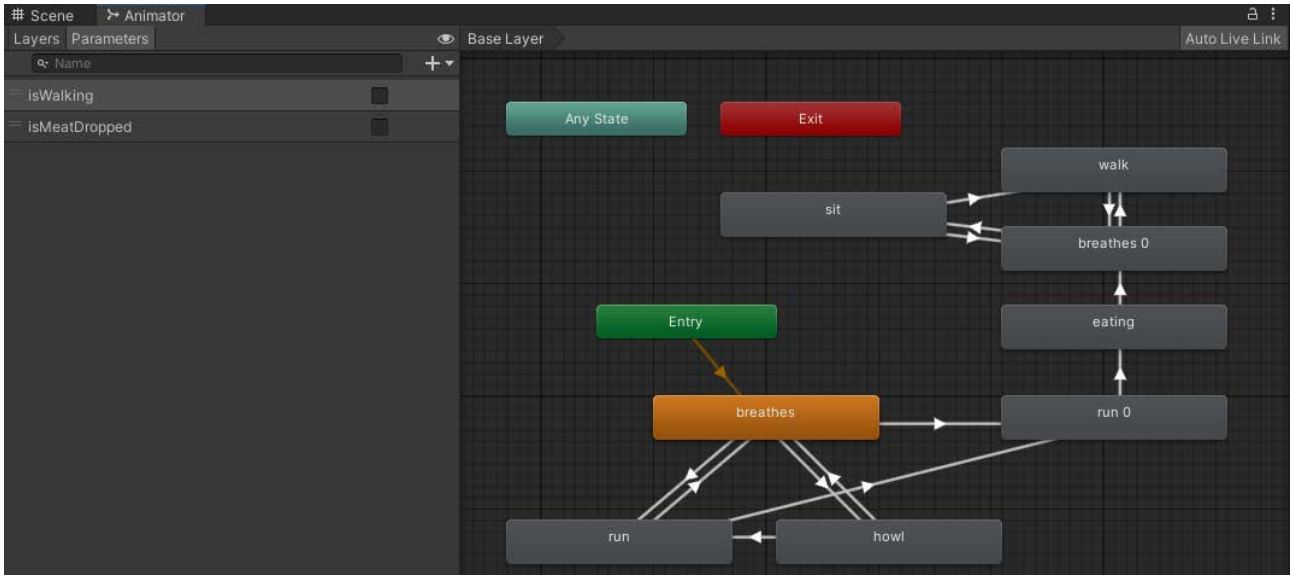
È presente una NavMeshSurface, settata in modalità “volume” di modo da essere generata solo per una zona ristretta di fronte alla casa. Sono stati usati dei NavMeshModifierVolume aggiuntivi con l’impostazione “Not Walkable” per definire ancora meglio la superficie (andando ad escludere alcune zone) dove è permesso il movimento del cane.

Il cane (importato dal package “Wolf Animated”) è quindi il NavMeshAgent, di cui ho impostato le dimensioni, la velocità di movimento e quella di rotazione. Ho impostato la stopping distance di partenza a 3, per evitare che si avvicini troppo alla staccionata o all’utente, verranno descritto meglio successivamente i dettagli della scelta.

Lo script “NavigationScript” contiene il comportamento e il movimento del cane e l’interazione con la carne e con l’utente. Quando l’utente si avvicina alla casa, il cane (rimanendo dentro la staccionata) raggiungerà la posizione della navmesh più vicina all’utente, seguendo quindi l’utente dall’interno della staccionata quando questo si muove. Però dato che l’utente si trova all’esterno della superficie navmesh (e allargarla non sistemerebbe il problema, dato che l’utente si ritroverebbe in una sezione staccata della nevmesh rispetto al cane, che quindi non riuscirebbe a raggiungerlo) viene usata la funzione NavMesh.SamplePosition() dove vengono specificati dei parametri che permettono di ritornare la posizione della surface più vicina all’utente, in un determinato raggio.

Questo stesso meccanismo è utilizzato (con parametri diversi) con la carne, di modo che quando la carne viene lanciata all’interno della recinzione, questa troverà una posizione della nevmesh surface vicina, e il cane seguirà questa posizione al posto di quella dell’utente.

All’interno dello script vengono quindi anche modificati alcuni parametri del “Wolf Animator Controller” per permettere la riproduzione delle animazioni corrette ma anche per modificare il comportamento del cane (ovvero le sua animazioni) dopo che quest’ultimo ha ricevuto la carne.



Possiamo qui vedere come il parametro “isWalking” gestisce la transizione verso le animazioni di corsa per entrambi i due comportamenti, mentre invece “isMeatDropped” permette il passaggio verso lo stato “run 0”, che porta quindi a rimanere in un loop di animazioni diverse che rappresentano il diverso comportamento del cane che quindi al posto di correre e ululare, camminerà e si siederà.

Le animazioni “howl” e “sit” sono riprodotte tramite un exit time dell’animazione di idle (breathes), ovvero ogni tot tempo.

L’animazione “eating” invece viene raggiunta quando il cane smette di correre (tramite il parametro isWalking) e ha quindi raggiunto la carne.

Lo script gestisce quindi anche il cambiamento del suono in loop del cane, da un più aggressivo iniziale ad uno più tranquillo successivamente.

Sono presenti anche due script collegati a degli stati dell’animator: “howlingDogState” fa in modo che quando l’animator entra nello stato “howling”, verrà riprodotto l’ululato del cane, mentre “wolfHasFinishedEating” va ad eliminare il GameObject della carne, di modo che lo script “NavigationScript”, non vedendo più la carne, tornerà al comportamento precedente dove il cane segue il giocatore.

Quando il cane riceve la carne, viene anche attivata una funzione dello script “gateController” che apre il cancello del recinto. Per evitare che la rotazione del cancello cambi istantaneamente (cosa non realistica), ho utilizzato “Quaternion.Lerp()” per creare una sorta di “animazione” tra la posizione iniziale e quella finale desiderata del cancello.

Il tavolino è stato importato dal package “Destructible Wooden Table”, mentre il pezzo di carne da “RPG Food Props DEMO”, con un grab interactable assegnato (con rigid body in modalità continuous detection, per evitare che passi attraverso il collider del terreno quando lanciato) e un attach point che personalizza il punto dove viene afferrato l’oggetto. Gli altri oggetti presenti sul tavolo sono contenuti in package che verranno citati successivamente.

Easter Egg

In questa sezione di gioco, ho inserito un piccolo easter egg. Al posto di lanciare il pezzo di carne al cane, il giocatore potrebbe trovare un sasso sulla destra del cancello (importato dal package “Rock_Pack”), e tirarlo addosso al cane.

Questo riprodurrebbe il seguente audio: “haha, okay that was mean”.

Per quanto riguarda il funzionamento, è presente una trigger area sul sasso, che tramite il solito script “Trigger” andrà a controllare se avviene una collisione con un oggetto con tag “Dog”, chiamando lo script “EasterEgg” in caso la collisione venisse rilevata.

Il cortile della casa

Il giocatore riesce ad entrare nel cortile, ma la porta della casa non si apre. Procede quindi ad esplorare il giardino, in ricerca di un meccanismo che sblocchi la porta.

Contenuto del cortile

La staccionata è stata creata utilizzando i prefab contenuti nel package “Free Hut Pack” (di cui ho utilizzato altri asset come la cabina e il cesto nel cortile) insieme al “Fence Layout Tool” che permette di creare la struttura della staccionata in modo più semplice e veloce, senza dover posiziona ogni singolo oggetto manualmente.

Il cortile contiene altri asset, ricavati dai packages “Old table and chairs”, “Cracked Tree Trunk”, “Bush-Craft Starter Pack”, “Tools and Logs” e infine da “Fallen Tree Barrier” ho utilizzato il tronco d’albero caduto posizionato sulla destra della casa, che è usato come barriera tra il cortile e il gioco delle lattine (descritto nella prossima sezione). Alcuni di questi asset hanno subito modifiche nei modelli, materials, dimensioni e colliders.

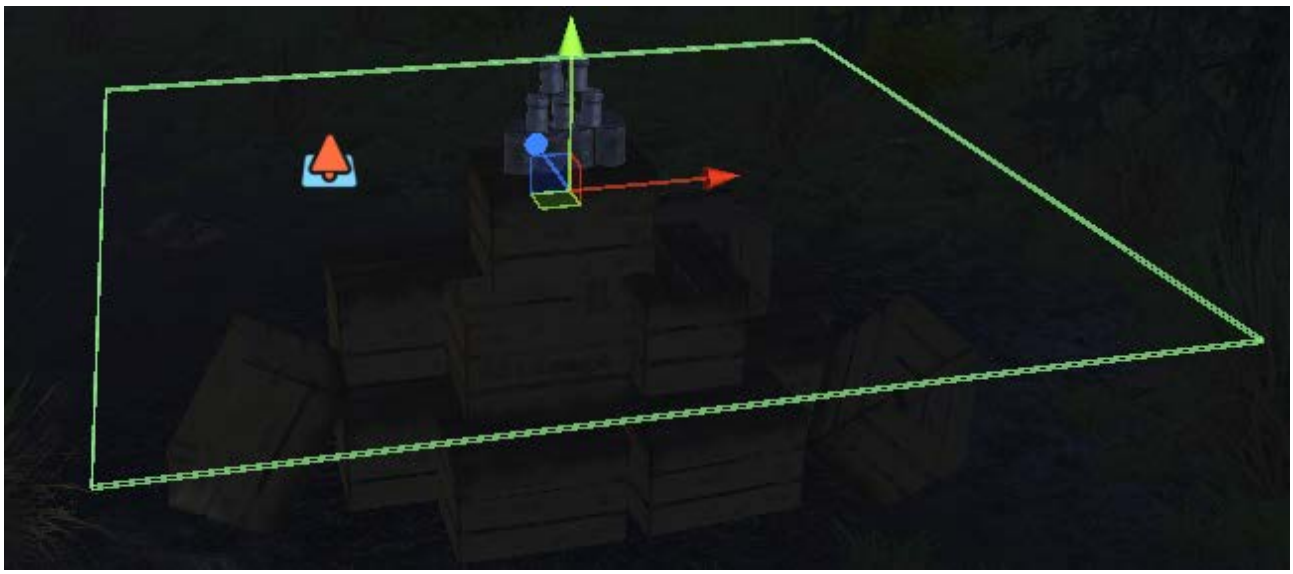
Il gioco delle lattine

Il giocatore trova sul lato destro del cortile delle palle di metallo in un cesto, e delle lattine impilate sopra delle casse. Lancia una sfera alle lattine colpendole e facendole cadere. La porta della casa viene quindi aperta.

Funzionamento del gioco

Ciascuna lattina (dal package "Free cans opened pack") sopra alle casse (dal package "Dirty Wooden Crate") ha un rigidbody e un collider, di modo che possano essere colpite e mosse da una collisione. Ognuna ha anche un tag "can" che sarà utile dopo. Allo stesso modo, tutte le sfere di metallo hanno un rigidbody, un collider un XR Grab Interactable di modo che l'utente possa afferrarle con la mano (ma non con il raycast, dato che usano l'interaction layer mask "Throwable").

Per fare il riconoscimento di quando le lattine vengono colpite e fatte cadere, è presente una trigger zone sottostante a cui è associato lo script "Trigger" già usato in precedenza, ma settando come tag filter "Can", di modo che quando una lattina passa per il detector, viene chiamata la funzione nello script "Cans Game".



Lo script "Cans Game" contiene la logica che abilita il grab interactable della porta della casa, abilita il movimento della porta e riproduce un suono di apertura della porta.

Il puzzle dei quadri

L'utente apre la porta ed entra nella prima stanza della casa, dove trova il corpo di una persona senza pelle. La prossima porta è chiusa. Il giocatore deve quindi risolvere il puzzle dei quadri, appendendoli al muro nell'ordine corretto, ovvero in base a quale parte di gioco rappresentata nei quadri abbia preceduto o seguito le altre nel gioco (la soluzione è l'auto nel primo quadro, seguita dalla lampada caduta, seguita dalla casa).

La prima stanza

Il modello della casa proviene dal package "Survival old House", a cui ho dovuto aggiungere tutti i colliders (aggiungendo nuovi oggetti invisibili) sui muri, sul pavimento e sulle scale. Ho poi aggiunto un pavimento e un soffitto differenti proveniente da un altro package ("Dark Fantasy Kit") dato che il modello della casa aveva dei buchi su entrambi.

Sono state aggiunte anche le porte, dal package "Free Wood Door Pack", a cui ho aggiunto tutti i meccanismi necessari a renderle interattive tramite grab interactable (simile sistema a quello usato precedentemente con il fuse box, ma con l'aggiunta di un collider speciale che viene rimosso quando il minigioco è stato completato), e quelli che vanno a bloccarle/sbloccarle a seguito del completamento di un minigioco.

Ho quindi arredato la stanza con svariati oggetti provenienti dai seguenti packages: “Cabin Environment”, “Furniture FREE Pack”, “Next-gen Furniture Pack”, “Dark Fantasy Kit”, “Horror Starter Pack”, “Brain Meal”, “Blood splatter decal package” e “Horror Axe”, aggiungendo colliders e modificando i prefab (dimensioni, colori, luci…) quando necessario.

Ho scelto di utilizzare luci provenienti da candele e lanterne all’interno della casa, per mantenere un’atmosfera cupa, creando composizioni con gli oggetti dei package come la sedia con il sangue e il cadavere senza pelle.

Quest’ultimo è stato importato dal package “Skinless Zombie” come avatar umanoide. Ho creato un’animazione personalizzata da zero (“Laying dead”), spostando le sue componenti per raggiungere la posizione sulla sedia che si può vedere nell’immagine precedente.

Funzionamento del minigioco

Ho creato manualmente il modello di un chiodo (di dimensioni abbastanza grande, così da poter essere visto facilmente), che ha un sphere collider e un XR Socket Interactor (con interaction layer mask “Painting”, di modo che non si possa appoggiare nessun altro oggetto).

Il modello del quadro proviene dal package “Picture frames with photos”, a cui ho aggiunto i materials e le textures create su photoshop, modificando quelle del package.

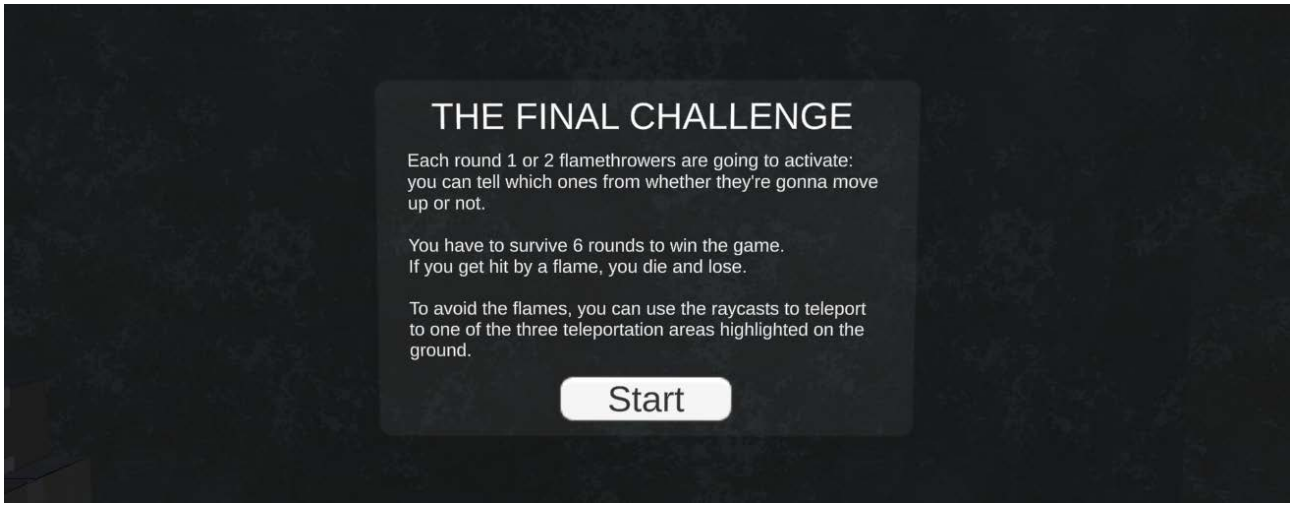
L’interazione con i quadri è fatta tramite grab interactable, impostato per funzionare sia con le mani ma anche con i ray interactor. Ho creato anche un attach transform personalizzato per cambiare il punto in cui viene afferrato l’oggetto.

Lo script “CheckPaintingsSockets” controlla quindi l’oggetto attaccato al socket di ciascun chiodo, verificando se l’ordine sia corretto e nel caso questo sia verificato, lo script procede a sbloccare la seconda porta (abilitando il grab interactable e rimuovendo il collider di protezione, come prima) e a riprodurre un suono.

Trappola dei lanciafiamme

L’utente entra nell’ultima stanza della casa e inizia l’ultimo minigioco premendo il pulsante sul muro. Premendo il pulsante play, inizierà il gioco. Il giocatore deve sopravvivere 6 round evitando i lanciafiamme, muovendosi tramite raycast teleport.

Interfaccia per iniziare il gioco



Sul muro della stanza è presente un'interfaccia che descrive il gioco per rendere chiaro cosa deve fare l'utente. Tramite raycast il giocatore deve quindi premere il pulsante "Start" che attiverà lo script "FlamethrowersGameController" iniziando il gioco e nascondendo l'UI.

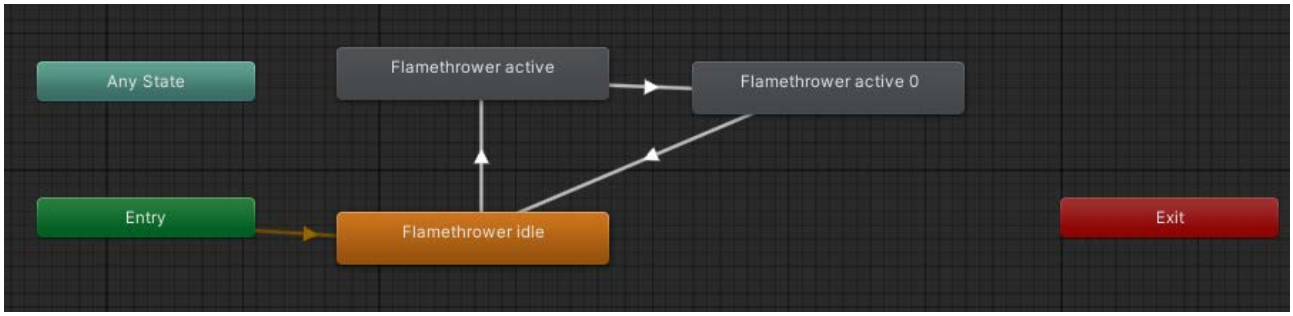
Funzionamento dei lanciafiamme

Il modello della torretta è stato importato dal package "Sci-fi Turrets". I lanciafiamme sono posizionati in un lato della stanza, sopra a delle scatole importate dal package "PBR Cardboard Box" e "Cardboard Boxes Pack HD".

Il particle system del fuoco è stato creato manualmente, per avere il pieno controllo sull'estetica e sulle dimensioni, ed è stato inserito all'interno di una variante della torretta. Ciascuno di questi prefab contiene anche da una trigger zone di fronte alla torretta, che verrà controllata dallo script "FlamethrowersGameController".

L’animation controller del lanciafiamme

A ciascuno dei 3 lanciafiamme è stato assegnato lo stesso animator “Flamethrower Animation Controller”.



La transizione dallo stato di idle al primo stato active è controllata tramite la variabile booleana “flameThrowerActive”. I due stati idle e active rappresentano due posizione statiche della torretta, ovvero la posizione dove la torretta è abbassata e la posizione nella quale punta davanti. Il passaggio da uno stato all’altro riproduce quindi l’animazione di attivazione o disattivazione della torretta.

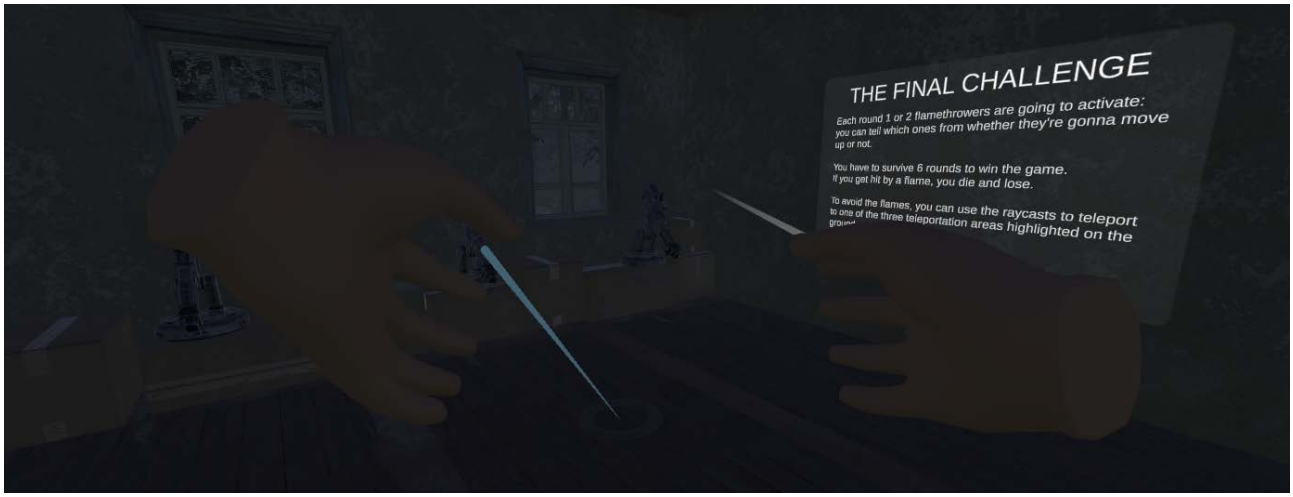
Il secondo stato active viene raggiunto automaticamente dopo che le animazioni sono state mostrate nei tempi impostati, di modo da semplificare e coordinare i meccanismi che devono essere attivati quando la torretta inizia a sparare la fiamme. In questo stato è presente lo script “FlamethrowerEmission” che nell’”OnStateEnter” attiva il particle system del fuoco, riproduce il suono del lanciafiamme, abilita la collision area e disabilita il parametro “flameThrowerActive” per evitare il ciclo. Nell’”OnStateExit” invece comunica al FlamethrowersGameController di passare al prossimo round, perché l’animazione è terminata.

Il game controller

Lo script “FlamethrowersGameController” quindi riceve il comando di inizio del gioco dall’UI, e tramite la funzione “startGame()” chiude la porta della stanza, disabilita il movimento continuo dell’utente (di modo da forzare l’utente ad usare il teleport tramite raycast) e inizia il primo round.

Ciascun round della funzione “nextRound()” attiverà le torrette specificate: nei primi 3 round sarà un singolo lanciafiamme ad essere attivato, mentre nei successivi 3 ne verranno attivati 2 contemporaneamente.

Il giocatore deve quindi usare il raycast interactor per muoversi su uno dei 3 assi di legno sul pavimento. Ciascun asse ha un Teleportation Anchor che ne gestisce l’interazione con il raycast, tramite il layer mask “Teleportation”. È stato anche stata specificata una posizione nella quale l’utente viene teletrasportato, indirizzandolo a guardare verso i lanciafiamme, a prescindere dal punto della teleportation area selezionato. È anche stato aggiunto un reticolo, modificando quello presente nel package XR.



Per controllare se l'utente è stato colpito dalle fiamme vengono quindi usate le trigger areas, che richiamano la funzione "gameLost()" del controller nel caso rilevino una collisione con l'utente. Queste aree sono attivate e disattivate dagli script presenti negli stati e nelle transizioni dell'animation controller (FlamethrowerEmission e FlamethrowerCollisionStop).

Se una collisione viene rilevata, la funzione "gameLost()" riprodurrà un urlo e dopo pochi secondi effettuerà il cambio di scena verso la "End Scene". Se invece il giocatore riuscirà a sopravvivere a tutti i rounds, verrà teletrasportato direttamente alla "End Scene" al termine del gioco.

In base al caso, verrà settato un parametro "PlayerWon" nel registro delle preferenze dell'utente, di modo da poter sapere, nella prossima scena, se l'utente ha vinto oppure ha perso.

Scena finale

La scena finale è semplicemente una copia di quella iniziale, che mostra all'utente il messaggio "You won!" oppure "You lost!" in base al valore del parametro "PlayerWon" recuperato dalle preferenze dell'utente.