Grau en Disseny i Desenvolupament de Videojocs

PROJECTE FINAL DE GRAU

Desenvolupament d'un joc amb Unity implementant diferents mecàniques basades en el moviment.

Autor:
Pablo Espinàs Expósito

Tutors:
Gustavo Patow

MEMÒRIA

Convocatòria:
Juny 2024

Departament:
Informàtica, Matemàtica Aplicada i Estadística

INDEX DE CONTINGUTS

1. Introducció i objectius

1.1 Introducció

El gènere parkour en els videojocs sol consistir en la incorporació de mecàniques i sistemes basats en la habilitat de percebre l'entorn i reaccionar amb rapidesa als obstacles presentats per poder moure's amb soltesa per l'escenari i així arribar al objectiu de la manera més eficaç i ràpida possible. Amb un nivells dissenyats per donar creativitat als jugadors a l'hora de trobar la millor ruta o superar els obstacles utilitzant els objectes i habilitats que s'hi presenten dintre de cada escenari.

Molts videojocs que han aplicat els sistemes de parkour destaquen per les seves mecàniques de moviment fluides i dissenyades amb precisió per fer sentir al jugador una sensació de moviment gratificant, a la vegada que desafiant. Amb uns dissenys de nivells que dona peu al jugador de sortir-se del camí establert per trobar altres rutes cap al objectiu. Donant lloc a millores no només de temps, si no també un augment considerable de la percepció del jugador al entorn i a la habilitat de resoldre problemes ràpidament.

Desgraciadament, en els últims anys hi ha hagut un nombre considerables de videojocs que utilitzen la mecànica del parkour com un sistema secundari sense acabar de profunditzar-lo i fent que sigui simple i, a vegades, repetitiu. Relegar el principal mètode de moviment a una mecànica secundària pot comportar molts cops a la simplificació de escenaris, fent-los de fàcil accés i convertint el desplaçament del nivell en un tràmit per arribar al objectiu en comptes de formar part de l'experiència jugable.

Per aquest motiu ens proposem a dissenyar i desenvolupar un videojoc per intentar trencar la costum de relegar el parkour com a unes mecàniques secundàries i crear un joc amb nivells desafiants on el jugador pugui provar les seves habilitats motores i la seva creativitat per poder-se'n sortir del repte que se li presenta davant.

1.2 Motivacions

Les motivacions que ens han impulsat en realitzar aquest projecte han sigut els següents:

- Posar en pràctica totes les habilitats apreses en el grau.
- Millorar el domini d'un motor de videojocs per anar agafant experiència
- Poder aprendre més en profunditat els reptes que hi comporten el dissenyar un videojocs amb mecàniques de parkour
- Poder veure amb exactitud que és el que s'ha de fer per aconseguir que el moviment formi part de la jugabilitat i fer-lo divertit.
- Experimentar amb els nivells i les diferents interaccions que un joc de parkour pot arribar a oferir.

Aquestes són les principals motivacions que han sigut clau a l'hora de triar de fer aquest projecte. No només amb la finalitat de realitzar-ho com a projecte de grau, sinó també aprendre i gaudir de tot el trajecte.

1.3 Propòsit i objectius del projecte

El propòsit del projecte principalment es realitzar un prototipus d'un videojoc on el principal tema sigui la mobilitat entre escenaris com aspecte a destacar i la fluïdesa en els moviments del personatge.

Els objectius del projecte principalment són:

- Aprendre a dominar les funcionalitats que incorpora el motor de Unity
- Acaba de dominar el llenguatge C#
- Guanyar experiència a l'hora de trobar solucions a problemes que no tenen una resposta clara.
- Aprendre a dissenyar nivells que complementin les mecàniques programades per així guanyar fluïdesa i oferir una experiència de joc més còmode per l'usuari.
- Familiaritzar-se amb software d'edició de imatges i de modelatge 3D.
- Adquirir la capacitat de estructurar amb eficàcia un projecte de grans dimensions de manera que afavoreixi la comprensió, la possible modificació i desenvolupament del projecte
- Aprendre a implementar un sistema de controls de consola amb el plugin de Input System.
- Habituar-se a crear unes interfícies clares per facilitar la comprensió dels jugadors.
- Millorar la habilitat a l'hora de dissenyar mecàniques que poden combinar-se entre si amb fluïdesa.

1.4 Distribució de tasques

Sent un joc el qual el principal tema es l'apartat mecànic, la distribució de tasques d'aquest projecte quedaria així:

Estètica	5%
Narrativa	10%
Mecàniques	35%
Tecnologia	50%

El percentatge relatiu a l'estètica es refereix als dissenys de diferents objectes que s'utilitzen par a la formació tant de l'escenari com per al dissenys de les interfícies, ja sigui el menú principal, el menú de pausa, game over, etc.

La narrativa d'aquest projecte no té un pes massa important, però si el suficient per a intrigar al jugador a seguir explorant els nivells i donar un context del videojoc que busca captivar al espectador.

En l’apartat de les mecàniques es tracta de descriure detalladament totes les accions que pot realitzar el jugador dintre del videojoc, complementant-se amb una explicació detallada de relacions d’aquestes mecàniques amb els objectes del entorn i quelcom.

Finalment, el percentatge de més pes es destina a la tecnologia, que es centra en la implementació de tots els apartats anteriors al motor gràfic Unity. Aquest apartat es centra en el control del jugador, transició entre nivells, implementació de sons, gestió dels atributs del jugador que es van guardant entre aquests nivells i qualsevol altre aspecte que hi guardi relació amb el motor Unity.

2. Estudi de viabilitat

Però tot i abans de començar a plantejar-se el disseny del projecte, és molt important fer un petit estudi sobre la viabilitat que té el videojoc i també ser plenament conscients de les limitacions que l'envolten. En aquest punt investigarem quins seran els recursos que necessitarem per poder realitzar el projecte amb èxit. Per fer-ho, realitzarem un estudi tant de mercat, en el qual analitzarem quins són els videojocs que més s'adeqüen al nostre videojoc com del model de negoci. No només això, si no que haurem de definir un perfil del jugador mitjà que provarà el projecte i una avaluació de la tecnologia que serà necessària per poder desenvolupar el videojoc sense cap mena de problema.

2.1 Recursos humans

Per a la gran majoria de projectes tecnològics, una de les coses més importats és la formació d'un equip de persones amb diferents habilitats i fortaleses que puguin aportar diferents punts de vista per així donar el millor producte. I els videojocs no són diferents. Per al videojoc, faran falta els següents rols essencials per generar la millor versió:

- Programador: Encarregat de tot el tema de la codificació i implementació del joc.
- Dissenyador de nivells: Persona responsable de crear i estructurar els nivells del jocs, així mateix com la planificació dels diferents camins que podrà agafar el jugador.
- Artista: Encarregat del disseny visual dels objectes dels escenaris, menús i diferents efectes visuals per donar-li al joc un sentit de l'estètica únic.
- Dissenyador de so: Responsable de la implementació dels efectes de caràcter auditiu: ja sigui els efectes de so com la musica que sona en els nivells per donar més immersió.
- Dissenyador de mecàniques: Persona encarregada de pensar i dissenyar totes les accions que hi pot realitzar el jugador: Saltar, córrer, utilitzar objectes, etc. Depenent del tipus de joc haurà de treballar conjuntament amb altres departaments. En el nostre cas ha de cooperar amb el dissenyador de nivells per poder implementar-ho tot correctament.
- Testers: Gent, no necessàriament treballadors o gent del grup, que s'encarrega de revisar que totes les peces del joc funcionen correctament entre elles i informar dels errors trobats o possibles queixes que hi poden tenir respecte certs apartats.

El desenvolupament d'aquest projecte ha estat realitzat per un sol membre, encarregat de realitzar totes les tasques esmentades.

2.2 Recursos tecnològics

Per realitzar el projecte amb la millor qualitat possible, es molt important disposar d'un hardware suficientment potent per poder-hi treballar en el videojocs de manera ràpida i eficaç.

En el nostre cas, utilitzarem les següents especificacions pel hardware:

Targeta de vídeo	NVIDIA GeForce GTX 1660 SUPER
CPU	AMD Ryzen 5 3600 6-Core Processor
RAM	16GB
Sistema Operatiu	Windows 10

També farem ús d'un comandament de XBOX One per a les proves dels controls del videojoc.

A part del Hardware, també requerirem de varis Software que ens ajudaran a desenvolupar el projecte. En el nostre cas utilitzarem una gran quantitat de programes gratuïts:

- Unity 2022.1.19f1: El motor de joc que utilitzarem pel desenvolupament del videojoc, també inclourem els plugin (apart dels que ja hi venen per defecte) Input System per a la implementació dels controls amb comandament i el Universal RP pel sistema d'il·luminació.

- Visual Studio 2019: Principal editor de codi que utilitzarem pel desenvolupament del joc. Útil també per debugar el joc en temps real.

- Blender: Eina de modelatge 3D que utilitzarem per a la creació d'objectes pel joc.

- MSPaint: Programa d'editor de imatges senzill que farem servir per a l'edició d'algunes textures senzilles i que no requereixen de molts detalls.

- Google Drive: Servei d'emmagatzematge que ens serà molt útil per a la gestió de copies de seguretat del projecte.

Igualment, també farem ús d'algun Software de pagament per acabar de perfeccionar el nostre projecte:

- Adobe Photoshop (24.19€/mes): Eina d'edició d'imatges que farem ús per a la creació de textures més detallades i d'alguns apartats d'interfícies com el menú de controls o el menú principal.
- Sony Vegas Pro 16 (19.99€/mes): Editor de vídeo utilitzat per a l'edició de l'arxiu multimèdia inserit en el menú principal.

Encara que nosaltres utilitzarem aquest Software de pagament a causa que tenim millor domini d'aquestes eines que amb altres, hi ha vàries alternatives de programes amb llicències gratuïtes que poden complir les mateixes funcions. Per sort, el Software de pagament va ser adquirit anteriorment al projecte, per tant no ha suposat cap inversió en l'àmbit tecnològic.

2.3 Viabilitat econòmica

2.3.1 Recursos tecnològics

A causa de la utilització de Software de pagament i de Hardware, haurem de realitzar una aproximació dels possibles costos associats amb aquests recursos. En el nostre cas, aquests recursos ja els havíem adquirit prèviament per tant no han suposat cap cost tècnic. Tot hi així, farem una estimació de les inversions que hauríem de fer en el hipotètic cas d'haver de pagar els costos:

RECURSOS TECNOLÒGICS	**COST**
Cost Ordinador	1500€
Comandament de joc	76.89€
Adobe Photoshop	24.19€/mes
Sony Vegas Pro 16	19.99€/mes
COST TOTAL (6 mesos)	**1841.97€**

El cost total l'hem calculat tenint en compte el temps que vam estar utilitzant els Software de pagament, no correspon al total de temps que vam estar per el desenvolupament del projecte.

2.3.2 Recursos humans

Pel que fa als costos dels recursos humans, en el nostre cas també haurà de ser 0, ja que tot el projecte ha estat desenvolupat per una persona de forma gratuïta. Però farem una estimació hipotètica dels pressupostos dels integrants de l'equip, suposant que contractem a un professional de cada perfil principal:

- Programador: **13 €/hora**
- Dissenyador de nivells: **16 €/hora**
- Artista: **13 €/hora**
- Dissenyador de mecàniques: **16 €/hora**
- Tester: **10€/hora**
- Dissenyador de so: **12 €/hora**

Els costos per hora que hem trobat a la plana web www.payscale.com, són en relació a España.

2.4 Estudi del mercat

Un cop tenim establert les idees general del videojoc que anirem a desenvolupar, hem de començar a analitzar quin és l'estat actual de videojocs que s'aproximin al gènere que volem desenvolupar. Fer un anàlisi de la competència ens proporcionarà una base bastant sòlida a l'hora de triar les decisions respecte a dissenys i implementacions del projecte, ja que volem que el nostre producte arribi al públic objectiu i tingui èxit.

2.4.1 Estat de l'art

En aquests apartat, explorarem els millors videojocs del gènere parkour per poder comprendre quines són les característiques clau que necessita el joc per poder satisfer el públic d'aquest gènere. A més també ens podran donar idees sobre diferents mecàniques i estils que podríem agafar de referencia en el projecte. Els principals jocs que hem decidit analitzar són jocs que comparteixen el gènere amb el joc que hi desenvoluparem, o que compten amb uns controls o mecàniques similars. A continuació es comentarà breument els jocs que hem estudiat i els seus aspectes més destacats, així com les característiques que hem pres de referència.

Mirror's Edge Catalyst(2016)

Mirror's Edge Catalyst és un videojoc que pertany al gènere d'acció/aventura i intenta ser una mena de continuació/reinici del seu predecessor "Mirror's Edge" (2008). En aquest joc es narra el passat de Faith Connors, una corredora encarregada de repartir cartes i paquets fora del marge de la llei en una ciutat controlada per les grans corporacions, la qual haurà d'utilitzar les seves habilitats acrobàtiques per descobrir els secrets que hi guarda la ciutat.

El joc destaca per les mecàniques fortament vinculades amb el moviment i exploració pel seu món: saltar, córrer, rodar, lliscar i altres moviment acrobàtics seran mecàniques que el jugador haurà d'utilitzar per aconseguir superar els obstacles dels nivells. L'estil visual es minimalista, utilitzant els colors freds per mostrar el aspecte estèril del món en contrast amb colors càlids, com el vermell, que s'utilitza per donar-li al jugador indicacions pel món (Veure Figura 1). Hem utilitzat aquest estil minimalista com a referència pel dissenys del nostre propi estil del videojoc.

Figura 1: Protagonista de Mirror's Edge Catalyst corrent per sobre d'un edifici.

Dying Light (2015)

Dying Light es un joc de terror/supervivència on el món ha estat infestat per un virus que converteix a la gent en zombis. El protagonista haurà d'aprendre a moure's utilitzant l'entorn i habilitats de parkour per així assegurar la seva supervivència i, amb sort, trobar una cura per a la infecció (veure Figura 2).

Una de les seves característiques més aclamades pel públic és el sistema de parkour, el qual haurà de ser indispensable per poder-se moure per la ciutat, ja que de no ser així es molt probable que els zombis acabin atrapant-lo. Aquest sistema permet al jugador explorar amb llibertat la totalitat del món, podent escalar edificis, sortejar obstacles i, fins i tot, arribar a estructures llunyanes utilitzant un ganxo.

En el nostre joc, hem agafat inspiració del ganxo per realitzar una mecànica semblant.

Figura 2: Protagonista de Dying Light desplaçant-se per la ciutat utilitzant les seves habilitats apreses.

Ghostrunner (2020)

Ghostrunner és un videojoc del gènere plataformes/acció ambientat en un mon apocalíptic on la raça humana esta a la vora de l'extinció. El jugador es posa en la pell d'un Ghostrunner, persones millorades tecnològicament per formar part d'una policia d'elit, al qual se li ha encomanat la missió d'acabar amb una intel·ligència artificial que governa als humans.

El joc destaca per ser un plataformes frenètic, amb un nivell de dificultat elevat i on es castiga el més mínim error. El jugador haurà de dominar els controls i mecàniques per poder superar els nivells, a més de estar disposat a jugar el mateix nivell varies vegades per tal d'acabar de polir les habilitats.

El seu disseny de nivells es basa en un estil inspirat en el cyberpunk (veure Figura 3), on les plataformes i estructures estan disposades en un espais pràcticament buit com si hi flotessin, donant al jugador una visió molt ample del nivell per així pensar en una bona estratègia.

De tots els jocs estudiants, és del que més inspiració hem tret. Hem agafat moltes de les seves mecàniques com a referència i el seu disseny de nivells ens ha servit molt per donar-nos idees a l'hora de fer el nostre.

Figura 3: Protagonista de Ghostrunner observant la situació abans de començar a avançar cap el final del nivell.

2.4.2 Comparació en el mercat

A continuació, hem realitzat una taula per comparar els diferents aspectes de cadascun dels jocs analitzats. Pel que hi podem veure en la taula, Mirror's edge catalyst és el que més destaca com a joc més rendible dels analitzats degut a la relació qualitat-preu. Si bé la seva jugabilitat té algunes coses que hi poden fer que l'experiència del jugador es vegi afectada de manera negativa, segueix sent un joc amb una qualitat jugabilistica molt bona i amb un preu bastant baix pels estàndards actuals.

Joc	Gènere	Gràfics	Jugabilitat	Preu
Mirror's Edge Catalyst	Acció/Aventura/Parkour	3D, Futurista	8	19.99€
Dying Light	Terror/Supervivència/Parkour	3D, Realista	7	29.99€
Ghostrunner	Plataformes/Acció/Parkour	3D, Futurista	9	29.99€

2.4.3 Model de negoci

En el món dels videojocs hi ha molts tipus diferents de models de negoci, però tots poden ser agrupats en quatre grups més grans que ens poden ajudar a veure quines són les tendències que tenen les empreses de generar beneficis en el món dels videojocs:

- **Free-to-play:** Diem que un joc és free-to-play quan l'usuari pot tenir accés al videojoc de manera gratuïta. Molts jocs que entren dintre d'aquest grup solen generar ingressos a partir de publicitat que apareix dintre del joc.

- **Pay-to-play:** Es el model on la gran majoria de videojocs hi pertanyen, es basa en que els jugadors paguen una quantitat fixe de diners per adquirir el joc i tenir accés a tots els continguts.

- **Freemium:** Aquest model de negoci es semblant al de pay-to-play, però amb la desavantatge de que no tens disponible tot el contingut del joc un cop havent pagat, si no que s'ha de pagar una quantitat addicional per poder desbloquejar la resta.

- **Subscripció:** Els jugadors han de pagar una tarifa regular cada cert temps per poder accedir al contingut del joc. Un cop deixen de pagar la tarifa, els jugadors deixen de tenir accés a aquest. Molts jocs solen combinar aquest model amb algun dels d'amunt per treure el màxim benefici possible.

Un cop havent valorat tots els models de negoci, hem decidit utilitzar el model de pay-to-play, donant la oportunitat de que qualsevol persona que hagi comprat el joc tingui disponible tot el contingut que hi ofereix el producte de manera permanent.

El caràcter innovador del joc i el que el diferència dels anteriors jocs de parkour analitzats es una rejugabilitat única, donant la oportunitat al jugador de, un cop han superat el joc, tornar a jugar els nivells però fixant-se amb alguns detalls que no s'hi havien donat compte en el primer cop. Això proporciona rejugabilitat i permet al jugador una oportunitat d'explorar els mons que d'una altre manera no ho haurien pogut fer.

2.5 Públic Objectiu

A l'hora de dissenyar un joc, és obligatori identificar i reconèixer el públic objectiu al qui va dirigit el videojoc. En el nostre cas, volem atraure jugadors amb una edat entre els 16 i els 30 anys. Tot i que el joc podria ser apte per a qualsevol de les edats, la principal raó de limitar el nostre rang d'edat es deu a la dificultat del videojoc, ja que es requereix d'una coordinació ulls/mans molt bona per acabar dominants els controls i les mecàniques necessàries per gaudir el joc sense molts problemes.

2.6 Perfil del jugador

El perfil del nostre jugador seria algú a que li apassiona els reptes, que no es desanima quan perd i segueix intentant-ho fins que té èxit. També busquem als amants del parkour, persones que gaudeixen de la sensació de llibertat i fluïdesa que pot proporcionar aquest gènere.

3. Planificació i eines de treball

3.1 Diagrama de Gantt

Abans de posar-nos en contacte amb el tutor del Treball de Fi de Grau, ja vam començar a fer una pluja d'idees sobre el que seria el projecte final. Un cop ens van acceptar la idea vam començar a planificar totes les tasques que hauríem de fer per donar el millor producte final.

Per aquesta raó, vam començar a dissenyar un diagrama de Gantt que ens ajudés a visualitzar tant les tasques com el temps per cadascuna d'elles. Al haver-hi començat amb el videojoc molt abans que la resta, això ens va donar una avantatge significativa a l'hora de gestionar-nos el temps per a cada tasca. Finalment ens va quedar el quadre de la Figura 4:

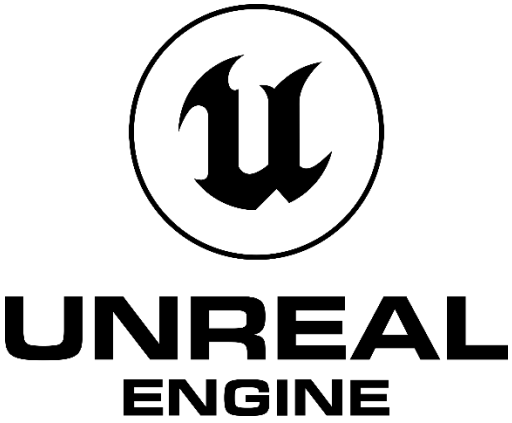
	Maig (2023)	Juny (2023)	Juliol (2023)	Agost (2023)	Setembre (2023)	Octubre (2023)	Novembre (2023)	Desembre (2023)	Gener (2024)	Febrer (2024)	Març (2024)	Abril (2024)
Pluja d'idees												
Decisió de mecàniques												
Implementació de càmera												
Implementació de mecàniques bàsiques												
Implementació de mecàniques avançades												
Correcció i millora de mecàniques												
Dissenys i implementació dels menús												
Dissenys de nivells i narrativa												
Implementació de nivells i narrativa												
Implementació de efectes de so i música												
Implementació suport comandament												
Correcció i millora d'interfícies												
Proves i balanç												

Figura 4: Diagrama de Gantt del projecte

3.2 Motor del joc

En el món dels videojocs, una de les primeres decisions que s'han de prendre es la de triar amb quin motor gràfic es vol fer servir per desenvolupar. Hi ha un gran nombre d'alternatives per triar. Depenent del tipus de videojoc que es vulgui fer hi haurà uns motors que tindran eines implementades que poden ajudar i facilitar el desenvolupament del joc. També es comú en empreses grans de crear un motor propi per tenir total domini de les eines que es necessitaran pel videojoc.

Unreal Engine és un motor desenvolupat per Epic Games. Destaca per ser un motor centrat en els jocs 3D, donant diverses eines tant per a desenvolupadors com a artistes. Està escrit en C++, però compta amb un sistema de blueprints que fan la programació en aquest motor molt més visual.



Godot es un motor de videojocs centrat en els jocs 2D i 2.5D (encara que té varies eines per a 3D) que va ser crear per dues persones però més endavant va ser la pròpia comunitat de Godot qui va començar a treballar en la millora del motor. Sent de codi lliure i multiplataforma, ofereix a molts usuaris la oportunitat de crear un joc sense ser-hi molt costós. Utilitza el seu propi codi de programació anomenat GDScript, encara que hi ha suport oficial per el C++ i el C#.

Pel projecte ens ha semblat Unity com la millor elecció de motor gràfic. Aquest motor es basa en un entorn de desenvolupament integrat lliure anomenat Mono, dissenyat principalment en el llenguatge C#. Compta amb un suport de compilació per a més de 20 plataformes, les quals destaquen PC, mòbils i WebGL, a més d'un gran nombre d'eines i components per a la creació de jocs 3D, 2D i fins i tot de realitat virtual.

Gran part del pes que ha tingut a l'hora d'escollir aquest motor ha estat la familiaritat que hi tenim utilitzant-lo, ja que durant el grau ha estat el motor que més hem utilitzat per realitzar diferents tasques. A més, aquest motor posseeix una de les comunitats més gran del món en quant a motors gràfics, fent que, a l'hora de realitzar cerques per resoldre dubtes, augmentin les probabilitats de trobar gent que han aconseguit solucionar problemes semblants als nostres.

3.3 Editors de codi

Com a editor de codi ens hem decidit pel predeterminat de Unity, el Visual Studio 2019. Aquest editor permet debugar projectes en temps real apart de ser un dels editors més potents actualment.

3.4 Programari artístic

Pel tema del disseny artístic ens hem decantat per el programa Blender, especialitzat en el modelatge d'objectes 3D i animació. Aquests programa l'hem escollit per tenir la qualitat de ser multiplataforma, lliure, gratuït i amb un pes bastant reduït, apart de ser un dels programes més utilitzats pels artistes de videojocs i, per tant, amb una gran comunitat per facilitar-nos el trobar solucions a problemes.

Conjuntament amb el Blender, també farem ús de Adobe Photoshop cs6 per a les textures dels objectes creats i interfícies dels menús. Sent un dels software líders en editors de fotografies. És un programa que ja hi teníem certa experiència gràcies a alguns projectes del grau i ens ofereix certa comoditat a l'hora de dissenyar les textures.

Finalment, també farem ús del editor d'imatges predeterminat de Windows, el MsPaint. Un programa bastant més simple que el Photoshop, però ens resultarà molt útil per fer lleugeres modificacions a algunes textures que no necessitin molts de detalls.

3.5 Altres

Pel tema d'edició del vídeo que utilitzarem pel menú principal, farem ús del Sony Vegas Pro 16. Encara que és un Software molt potent, només el farem servir per edicions bàsiques de vídeo. La principal raó per la que hem triat aquest i no altre és pel tema comoditat, ja que és el programa que més habituats estem a l'hora d'editar medis audiovisuals.

Finalment, pel sistema de còpies de seguretat de les diferents versions del videojoc farem servir Google Drive, el sistema de servei de núvol proporcionat per Google que permet no només emmagatzemar arxius, sinó també compartir-los en línia, facilitat la transferència de documents.

4. Conceptes previs

El videojoc ha estat desenvolupat per a un jugador experimentat en el gènere de parkour. En aquesta secció, parlarem dels conceptes bàsics que el jugador ha de tenir assimilats a l'hora de jugar al videojoc ja que no s'explicaran a dintre d'aquest.

El primer concepte que s'ha de tenir clar a l'hora de tractar amb un joc de parkour és la possibilitat de que hi hagin vàries maneres de moure's pel nivell, ja sigui utilitzant camins alternatius com realitzant diferents maniobres segons l'estil de joc. Aquestes alternatives es poden veure en els nivells mitjançant ajudes visuals. Aquestes ajudes poden variar depenent de l'estil artístic de cada videojocs, però normalment solen utilitzar-se dos recursos visuals:

- **Ús de llums en objectes:** Aquestes llums poden actuar com a indicadors visuals que es donen als jugadors per tenir una orientació clara pel nivell. Es sol utilitzar aquest estil en jocs a on predominen escenaris obscurs o amb poca il·luminació. Un exemple de videojoc molt clar en utilitzar-ho és el Mirror's Edge Catalyst (Figura 5)

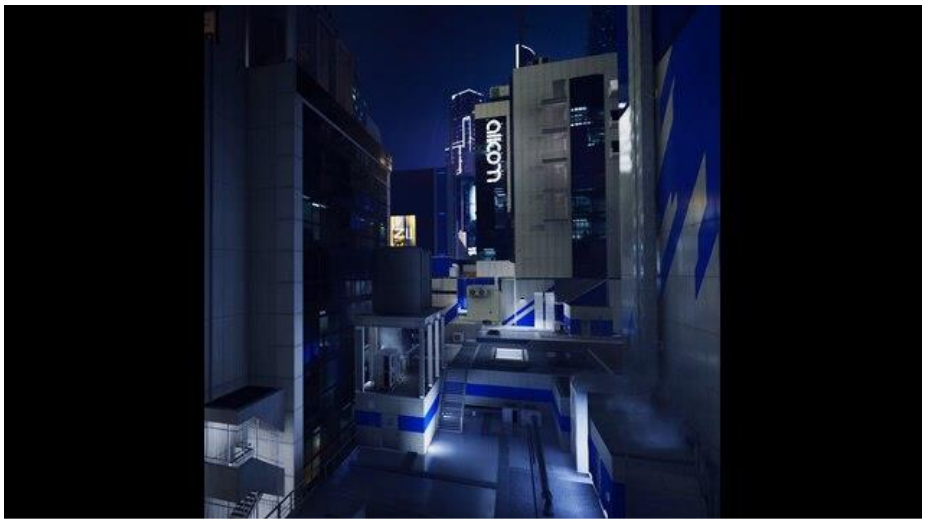


Figura 5: Escenari utilitzant la il·luminació per a guiar al jugador al objectiu

- **Ús de contrasts de colors:** En jocs on predominen una paleta de color durant tot el nivell solen utilitzar colors d'una altra paleta per generar un contrast i fer que els

objectes pintats d'aquesta manera siguin molt més cridaners, fent que hi sigui més fàcil per al jugador el poder veure aquests objectes i guiar-se per a ells. Un exemple clar d'aquest recurs és el Ghostrunner (Figura 6)

Figura 6: Escenari utilitzant el contrast de colors per guiar al jugador. En aquest cas podem observar com el color groc fa contrast amb el colors freds del escenari

Un altre concepte que es molt important per tal d'entendre la intenció del videojoc és la necessitat d'assaig i error per poder superar els nivells. Això a causa de que l'objectiu principal del joc consisteix en superar els obstacles d'una manera ràpida i fluida. Per fer això, s'ha de tenir una completa comprensió dels controls i del nivell que només el proporcionarà si s'intenta reiteradament els mateixos salts i maniobres, penalitzant al jugador amb tornar-ho a intentar si falla un dels obstacles posats. Aquest concepte es troba en varis jocs de parkour mencionats anteriorment, com el Mirror's Edge Catalyst o el GhostRunner. Jocs que t'obliguen a repetir el nivell cada cop que es cometi un error per així poder millorar a l'hora de moure's per l'escenari.

El concepte final del joc va lligat a l'anterior, que es la incorporació de punts de guardat en zones on les maniobres son complicades. Això fa que, un cop superades, si es falla més endavant, no s'hagi de tornar-ho a fer tot des del principi. Aquest concepte es molt important, ja que de no ser així, perdríem un gran nombre de jugadors a causa de la gran penalització que rebrien si fallessin un obstacle. Amb els punts de guardats automàtics afegim un cert balanceig a un joc difícil però no impossible.

5. Disseny del videojoc

5.1 Espai de joc

A causa del temps de realització i desenvolupament del videojoc, el projecte és un prototipus a on crearem un espai de joc amb l'objectiu principal de poder dissenyar les mecàniques que pot tenir el jugador a l'abast. Per fer-ho hem creat 3 nivells diferents, cadascú amb els seus nivells de dificultat i obstacles diferents, però on comparteixen el mateix ambient i estructura, que és una zona de simulació virtual.
Els tres escenaris dissenyats coincideixen amb els nivells implementats dintre del videojoc. El primer escenari compta com a tutorial i consisteix en un escenari obert amb un camí establert. Un cop s'arriba al final del nivell, el joc dona l'opció a l'usuari de tornar a jugar el mateix nivell o bé passar al següent. Aquests primer serveix per explicar al jugador les mecàniques principals de tot el joc.

Al segon escenari el jugador torna a una simulació però amb obstacles més complicats que el nivell anterior. La zona destaca per un nivell amb un camí bastant definit, on es posa en pràctica la mecànica de saltar y córrer per les parets, mecàniques explicades en el tutorial, mitjançant obstacles i estructures que requeriran un domini més avançat d'aquestes mecàniques per tal de superar-ho. Semblant al primer nivell, el joc preguntarà a l'usuari si vol tornar a fer el nivell o avançar cap al últim.

A l'últim nivell hi ha un canvi lleuger en l'estètica. Torna a ser dintre d'un simulador, però en aquest cas hi ha un sostre que impedeix al jugador arribar a altures grans. Al ser l'últim nivell del joc, l'usuari ha de fer ús de totes les mecàniques apreses en els nivells anteriors per poder superar els reptes establerts. Una altre característica que hi posseeix aquest nivell és la possibilitat pel jugador de triar diferents camins per superar el nivell, cadascun d'ells amb obstacles diferents però donant l'opció al jugador d'escollir el camí que més còmode es senti. Al final del nivell hi ha una porta amb un rètol on indica la sortida del nivell. Si el jugador el creua, el joc preguntarà un últim cop si vol repetir el nivell o si vol sortir de la simulació. En cas de sortir de la simulació, el jugador arribarà a l'escena final.

5.2 Mecàniques del joc

El joc està pensat en donar al jugador una sensació de moviment lliure, ràpid i fluid. Amb aquesta idea en el cap, hem creat una gran nombre de mecàniques pel moviment del jugador que proporcionin les sensacions buscades i un parell d'interaccions amb alguns obstacles per facilitat les maniobres.

La mecànica principal es la del moviment, que permet al jugador completa llibertat pel mapa en totes les direccions, facilitat l'exploració i avanç pels nivells. A continuació, també tenim la mecànica de salt bàsic, que ofereix al jugador la manera més fàcil de sortejar obstacles que estan per sobre d'ell.

A continuació començarem a parlar de mecàniques que, junt amb les de moviment bàsic, permeten al jugador moure's pels nivells amb rapidesa i soltesa. Aquestes mecàniques les anomenarem: carrera de paret, lliscar, ajupir, gronxar i enganxar.

5.2.1 Carrera de paret

Aquesta mecànica consisteix en utilitzar la paret per poder arribar a llocs on amb un salt no es suficient. Això permet al jugador realitzar salts sense haver de dependre del terra i dona més profunditat a les mecàniques de moviment bàsiques (veure Figura 7).

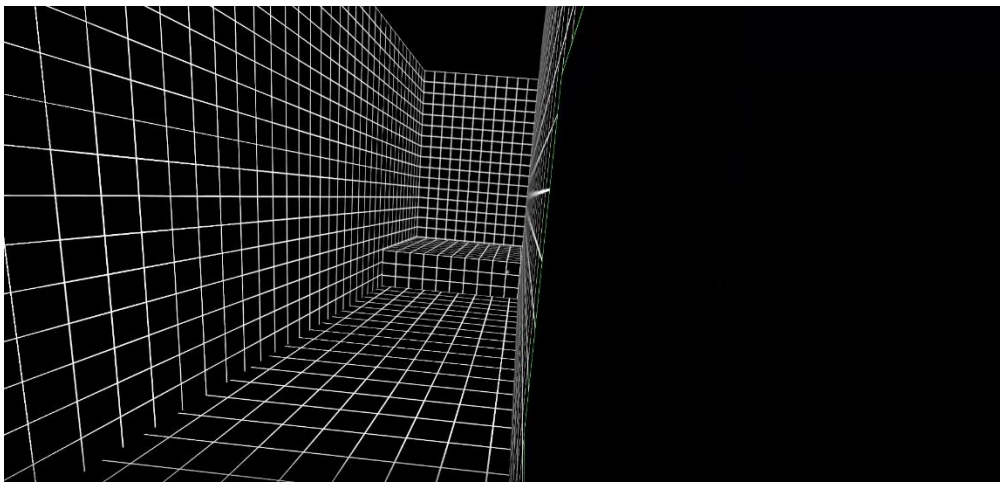


Figura 7: Exemple del jugador realitzant una carrera de paret per poder arribar a la següent plataforma

Cal destacar que aquesta mecànica està limitada pel tipus de paret que hi hagi al nivell, ja que només podrà realitzar-ho a parets que estiguin resseguides pel color verd (veure Figura 8). Això ens dona una manera de balancejar la mecànica, ja que de no ser així, el jugador podria fer ús de qualsevol paret per superar els obstacles amb facilitat.

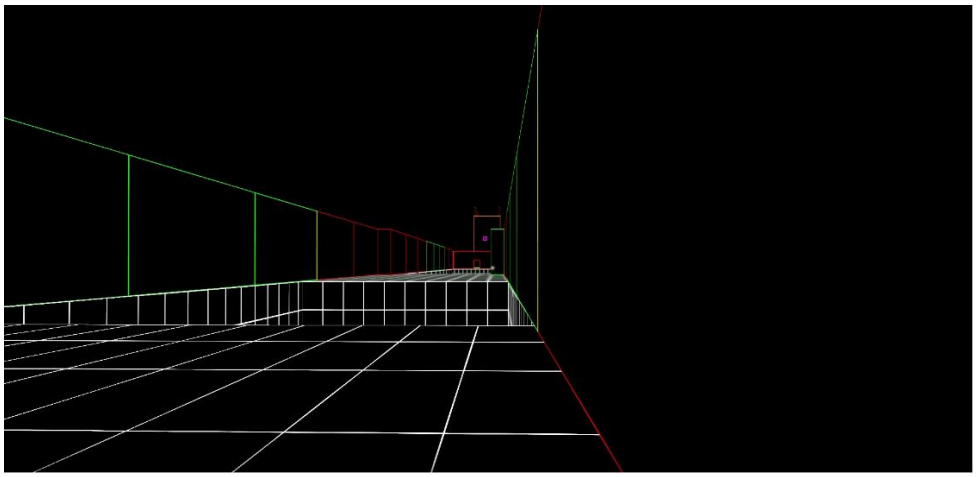


Figura 8: Escenari amb parets marcades amb el color verd indicant la possibilitat de realitzar una carrera de paret. També es pot apreciar variés parets marcat amb un altre color, donant a entendre que no s'hi pot aplicar la mateixa mecànica.

Aquesta mecànica no només es pot utilitzar per arribar més lluny amb els salts (encara que és una de les seves principals característiques), també serveix per arribar a superfícies que es troben fora de rang del salt bàsic del personatge (Veure Figura 9), permetent certa verticalitat en els nivells.

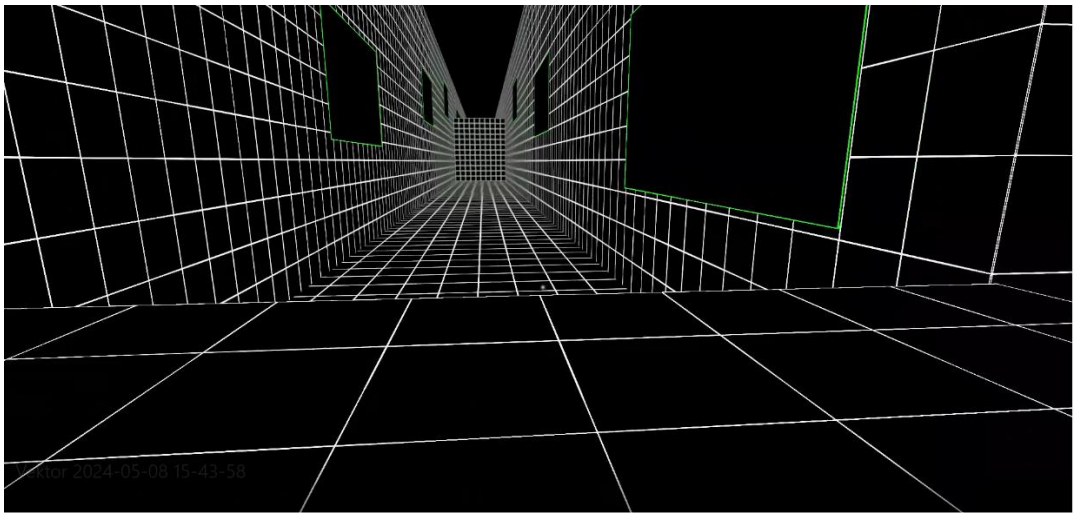


Figura 9: Escenari on el jugador ha de recórrer a la carrera de paret per poder-hi arribar a la plataforma més elevada

Tenir un bon control d'aquesta mecànica, conjuntament amb les de moviment, permet al jugador moure's sense perdre cap mena de velocitat, ja que la carrera de paret manté la mateixa velocitat que el personatge tenia abans de fer-la. Això acaba resultant en el tipus de moviment fluït que es busca en aquest videojoc.

5.2.2 Lliscar

Aquesta mecànica també està pensada per utilitzar-se conjuntament amb les de moviments bàsics. Permet al jugador baixar ràpidament de pendents, guanyant impuls per poder-hi saltar més lluny o guanyar més velocitat (veure Figura 10).

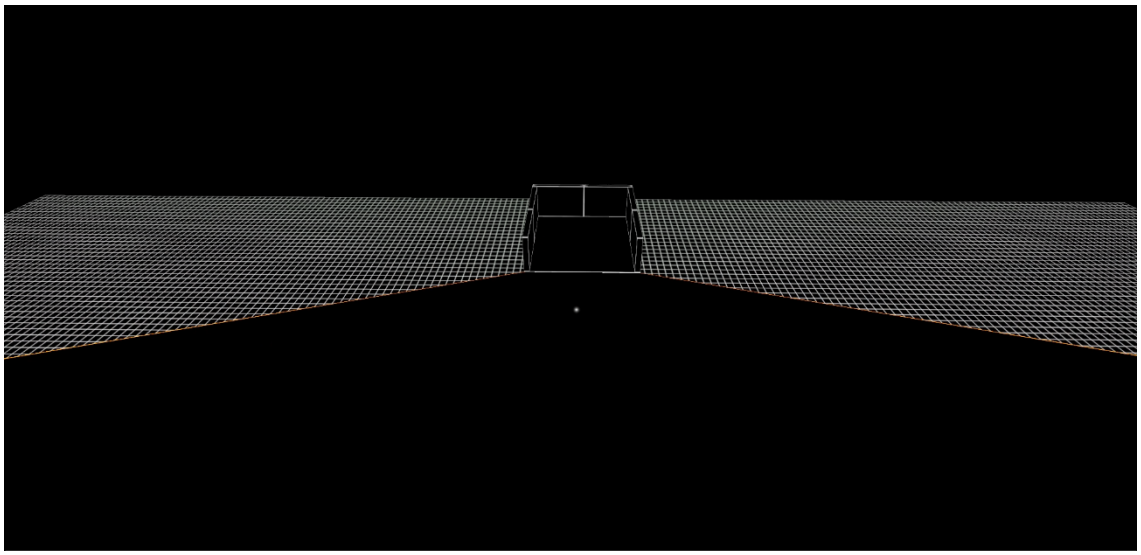


Figura 10: Jugador realitzant el lliscament per arribar ràpidament al final del nivell

Apart de la funció inicial, aquesta mecànica també compta amb dues funcions addicionals que, si bé no son obligatòries d'aprendre per superar els nivells, poden resultar útils per mantenir la fluïdesa en el moviment. Aquesta mecànica permet al jugador guanyar més distància quan s'està saltant a canvi de l'altitud. És especialment útil per evitat possibles errors a l'hora de realitzar salts llargs. Addicionalment, també es pot utilitzar per passar per llocs que requereixen ajupir-se (més informació en l'Apartat 5.2.3) sense haver de perdre la velocitat de moviment que es portava.

5.2.3 Ajupir

Mecànica bastant simple i bastant auto explicativa. Permet al jugador passar per zones on es requereix una estatura menor a la predeterminada (veure Figura 11). Realitzar aquesta mecànica comporta una penalització al moviment a canvi de una reducció de grandària, fent que el jugador pugui passar amb seguretat pels obstacles.

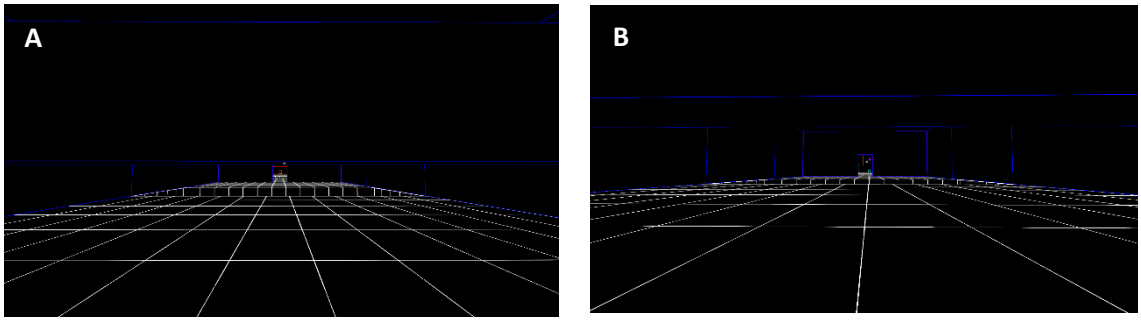


Figura 11: En la imatge "A" es mostra el punt de vista del jugador sense ajupir-se i la imatge "B" podem veure la vista del jugador ajupit.

Per afrontar aquests obstacles, també està disponible l'idea de lliscar (vist en l'Apartat 5.2.2) que manté la velocitat del personatge, però pot ser arriscat segons l'obstacle que vingui després.

5.2.4 Gronxar

Aquesta funció permet al jugador llençar una corda a una plataforma/objecte i gronxar-se per realitzar vàries coses: donar-se impuls per arribar més amunt, canviar de direcció mentre s'està en el aire o tenir més control sobre el personatge quan es mou a velocitats massa grans.

Com la mecànica de córrer per la paret (apartat 5.2.1), gronxar-se està limitat pel tipus d'objecte que hi hagi. El jugador només pot gronxar-se amb objectes que estiguin resseguits pel color violeta (veure Figura 12).

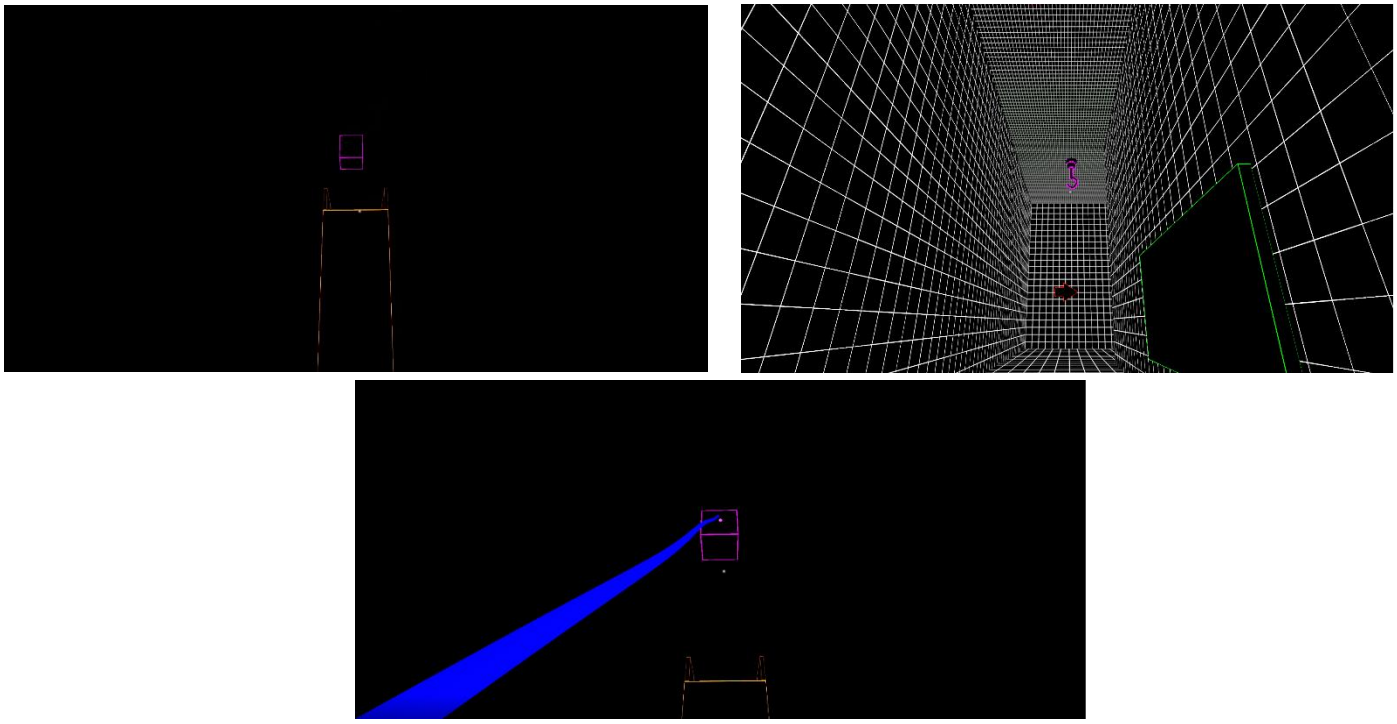


Figura 12: Diferents objectes els quals es poden utilitzar per gronxar-se i una mostra del que el jugador veu quan s'utilitzen.

5.2.5 Enganxar

Mecànica semblant a la de gronxar-se. A l'utilitzar-la, permet al jugador impulsar-se cap a l'objecte desitjat sense haver de realitzar cap mena de moviment. Aquesta mecànica s'utilitza per arribar a plataformes amb una altura molt superior a la que hi pot arribar el jugador. També es útil a l'hora d'evitar les caigudes, ja que també es pot utilitzar estant en l'aire (veure Figura 13). En aquest cas, els objectes han d'estar resseguits pel color groc per a que el jugador pugui utilitzar-ho.

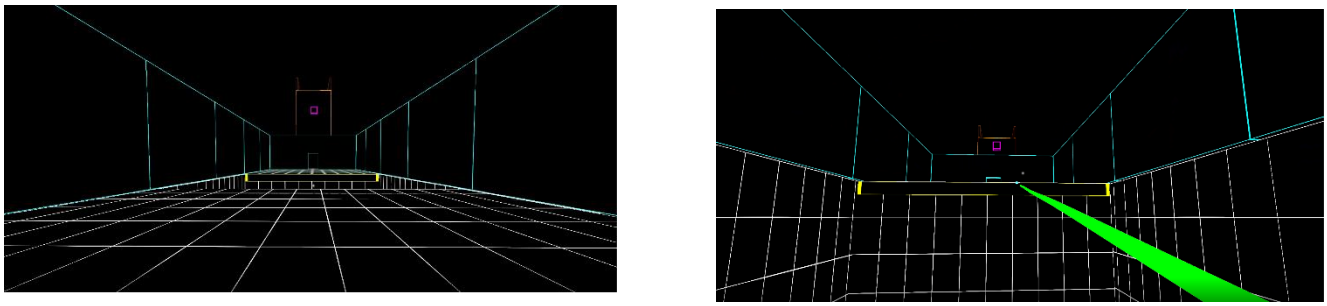


Figura 13: El jugador utilitza l'habilitat d'enganxar-se per evitar la mort. En la imatge es pot apreciar el color distintiu dels objectes els quals poden ser enganxats.

5.3 Interaccions del jugador

En el cas del nostre videojoc, no restringim les interaccions que pot realitzar el jugador amb l'entorn., ja que un dels principals aspectes del videojoc és que l'usuari pugui experimentar amb els moviment i mecàniques establertes amb la finalitat d'arribar al final del nivell, independentment del camí inicial que es triï. L'única limitació que establim en el joc és per evitar que el jugador pugui sortir del nivell, fent que no el pugui superar i la seva experiència es vegi arruïnada.

A més de les interaccions esmentades en l'apartat anterior, hi ha altres que el jugador pot realitzar per facilitar-li els nivells. En primer lloc, el jugador pot saltar a una paret normal i mantenir-se adherit a aquesta, mantenint el control de moviment premut, per aprofitar i calcular el seu següent moviment que permeti superar els obstacles.

La segona interacció té lloc quan el jugador es veu en la situació que el seu personatge es queda enganxat a la vorera d'una plataforma. En aquest cas, l'usuari podrà utilitzar la habilitat de lliscar per poder recuperar-se i així evitar caure.

5.4 Estudi de nivells

Els nivells del videojoc estan dissenyats per a que el jugador es vagi acostumant de manera gradual a les habilitats que té a l'abast, ja que, des de un principi, el personatge té totes les habilitats disponibles però, sense un bon entrenament, el jugador pot tenir dificultats per dominar-les. El prototip del joc consta de 3 nivells, amb un escenari per nivell i d'estructura lineal. Encara que hi ha l'opció d'escollir nivells concrets:

- **Nivell principal:** Aquest nivell, més comunament anomenat "Nivell tutorial", està dissenyat per a que el jugador sigui conscient dels controls i les mecàniques que té disponibles en tot moment.

- **Nivell senzill:** Un cop s'ha superat el nivell principal, passem al primer nivell "de veritat". En aquest es centra en ensenyar al jugador a fer ús de la mecànica de carrera de paret (mecànica vista en l'apartat 5.2.1), oferint diversos obstacles i plataformes on el jugador podrà experimentar amb aquesta mecànica. Encara que el jugador pot fer ús d'altres habilitats per superar el nivell, tot i que només són necessaris les habilitats de moviment i la carrera de paret.

- **Nivell final:** Aquest nivell es considera l'últim nivell del joc. En aquest s'haurà de posar en pràctica tots els coneixements dels nivell anteriors per poder assolir els obstacles presentats. És el nivell amb la dificultat més alta del joc, per tant serà necessari que el jugador hagi dominat fins a un cert nivell totes les mecàniques del joc per sortir victoriós.

Tot i que en el joc prototip només es mostrin tres nivells a jugar, el disseny inicial compta amb un total de 10 nivells que es centren en totes les mecàniques, tant de manera individual com en conjunt.

5.5 Interfícies

En el cas d'un joc de parkour, és vital mantenir la pantalla tan buida com sigui possible per a que el jugador tingui la màxima visió dels escenaris. A més, el jugador no necessita una barra de vida ni altres possibles dades que no hi guardin relació amb els obstacles del mapa. Per aquesta raó, hem decidit deixar la interfície de joc lliure. Això no vol dir que no es pugui introduir un petit element d'interfície en el joc de manera puntual. Per exemple, al nivell inicial podem trobar-hi una petita interfície abans de cada obstacle indicant els controls o combinacions de mecàniques que han d'utilitzar per superar-ho. Per realitzar-ho, hem aprofitat el color blanc amb contrast amb el color negre dels escenaris per poder mostrar un petit text just en el centre de la pantalla, per a que el jugador ho pugui veure (Veure Figura 14). Un cop ha passat cert temps, el text desapareix. El jugador té la capacitat de moure's pel nivell mentre llegeix els consells.

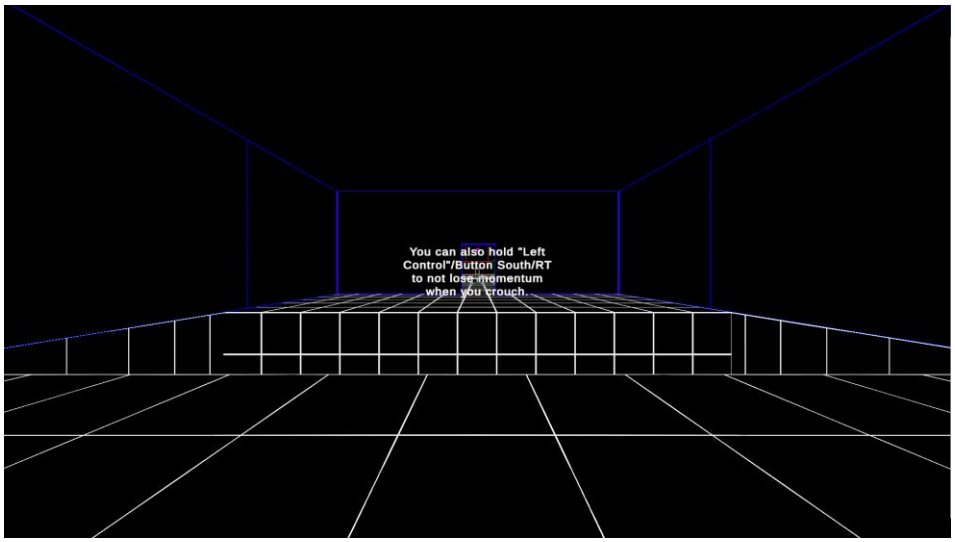


Figura 14: Exemple d'interfície amb informació sobre com superar els obstacles

5.6 Estil artístic i disseny

5.6.1 Mapa

El joc està enfocat en una simulació digital. Per aquesta raó, ens hem centrat en incorporar al disseny formes geomètriques senzilles agafant d'inspiració a videojocs en primera persona dels anys 70 per donar al jugador una sensació de retro-futurisme.

En un principi, el jugador ja pot veure que es troba en una mena de espai buit on els únics objectes son rectangles i quadrats de diversos colors. En arribar més endavant podem veure buits on s'ha de saltar per evitar caure i, conseqüentment, tornar a començar, o ajudar-se dels colors distintius dels objectes per avançar (Veure Figura 15).

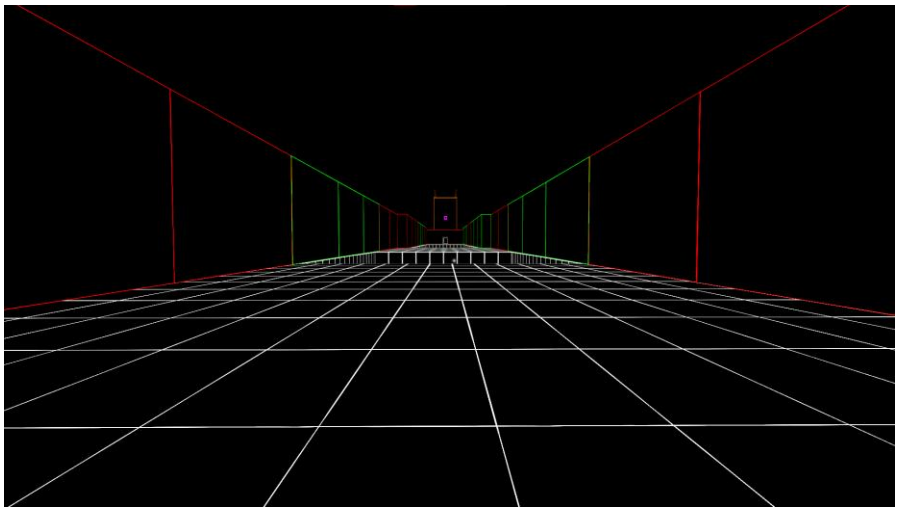


Figura 15: Escenari governat per rectangles minimalistes i amb diferent relleus per guiar al jugador.

Cap al final del nivell, es podrà apreciar una zona molt oberta on el jugador té l'oportunitat d'observar que no hi ha res més enllà del nivell, donant una sensació encara més obvia de estar en una mena de simulació generada només per a ell (Veure Figura 16).

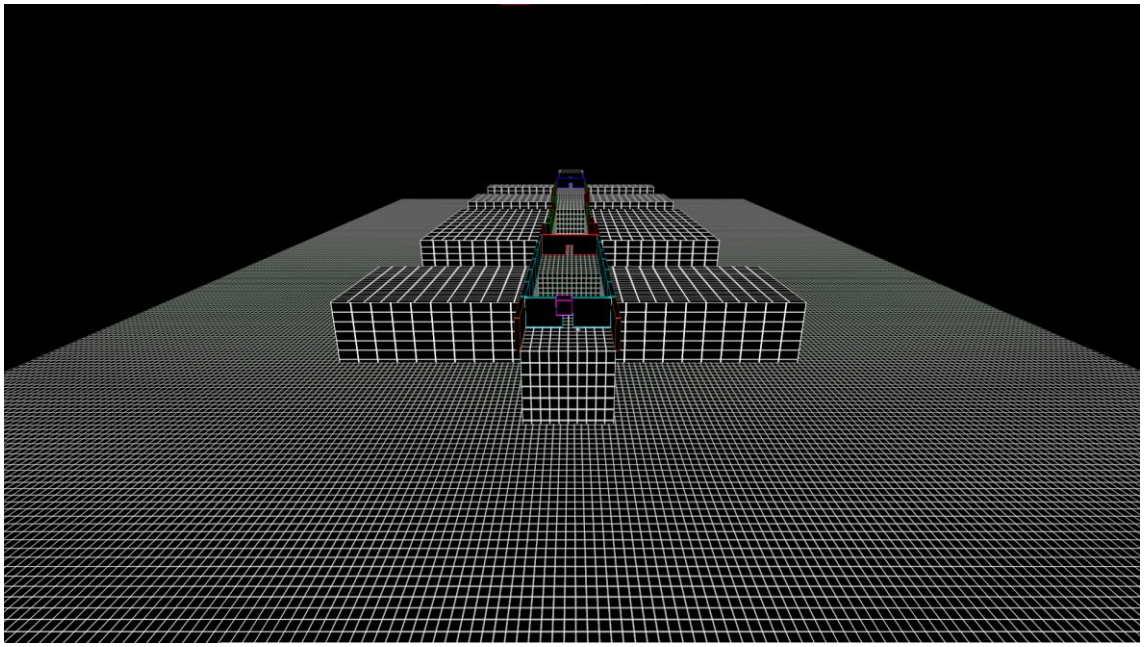


Figura 16: Paisatge completament buit, donant la sensació d'estar atrapat en una simulació.

En arribar al final del nivell, es podrà veure una plataforma sense cap tipus de camí per seguir endavant. Un cop trepitjada, el sistema li donarà l'opció al jugador d'avançar al següent nivell (Veure Figura 17).

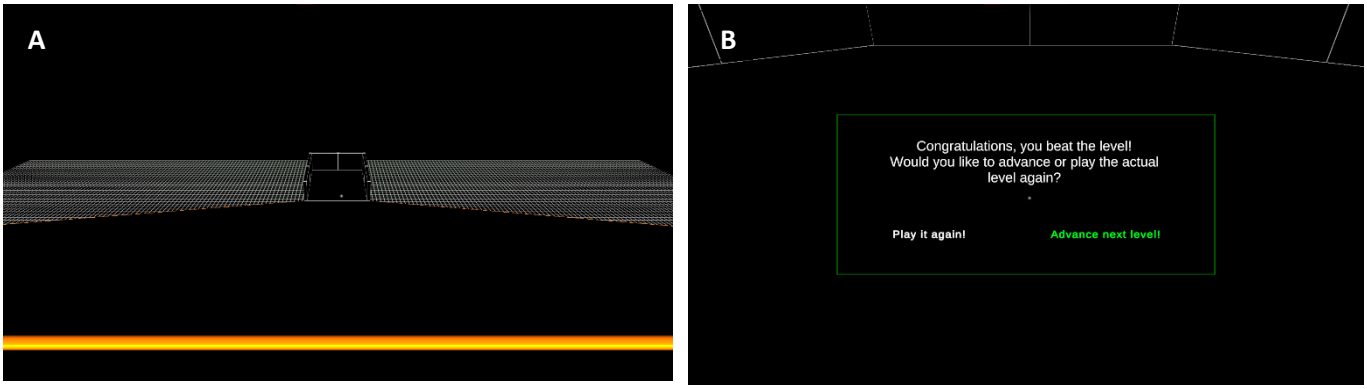


Figura 17: A la imatge A veiem la plataforma sense cap mena de sortida mentre que a la imatge B es mostra el que surt quan es trepitja, donant a entendre que algú ens està observant.

Un cop avancem a la següent zona podem observar un petit canvi en l'escenari. Les parets han agafat el mateix patró que el terra, donant la sensació encara més de que es una simulació que s'està construint. A mesura que es va avançant pel nivell, es va veient com cada cop i ha menys superfície per caminar, obligant al jugador a haver de saltar o utilitzat la paret per poder seguir cap al final (veure Figura 18).

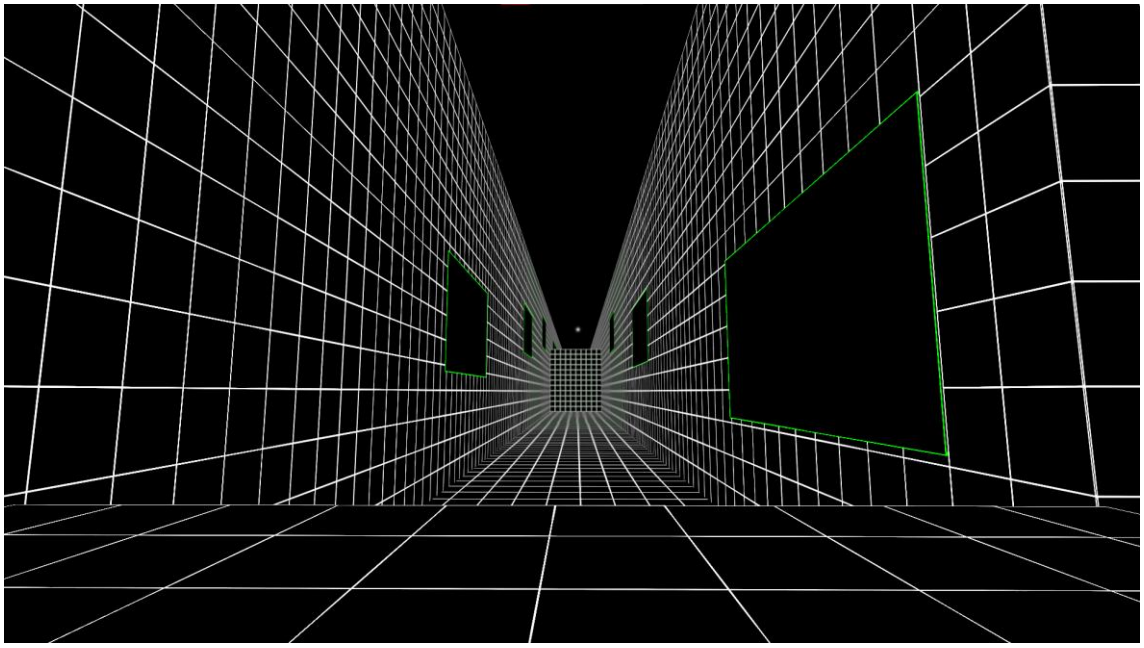


Figura 18: Les parets han canviat respecte al nivell anterior així com la disminució de superfície per caminar.

També podem trobar plataformes mòbils que el jugador haurà d'agafar per arribar a la última zona del nivell. Aquestes plataformes no estan unides a cap lloc, ja que al ser una simulació no requereix seguir un sentit lògic a l'hora de crear els obstacles (Veure Figura 19).

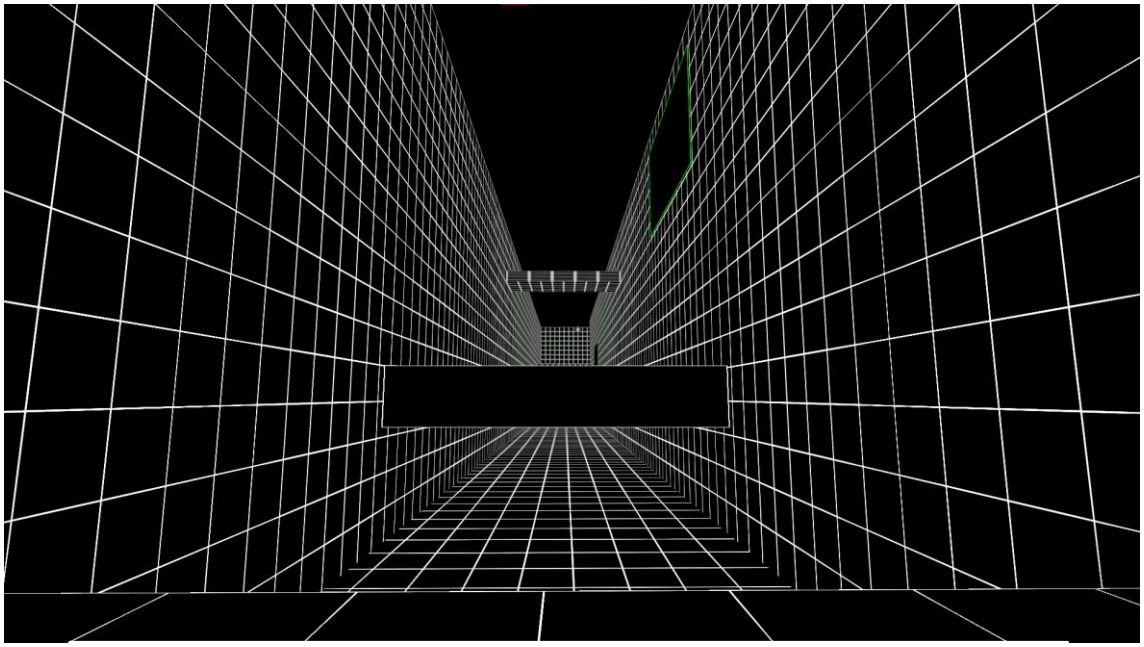


Figura 19: Plataformes mòbils sense seguir cap lògica espacial.

Cap al final del nivell, ens trobem amb habitació amb parets molt altes i cap sortida a la vista menys cap amunt, donant el missatge al jugador d'utilitzar les seves habilitats per poder sortir-ne (veure Figura 20).

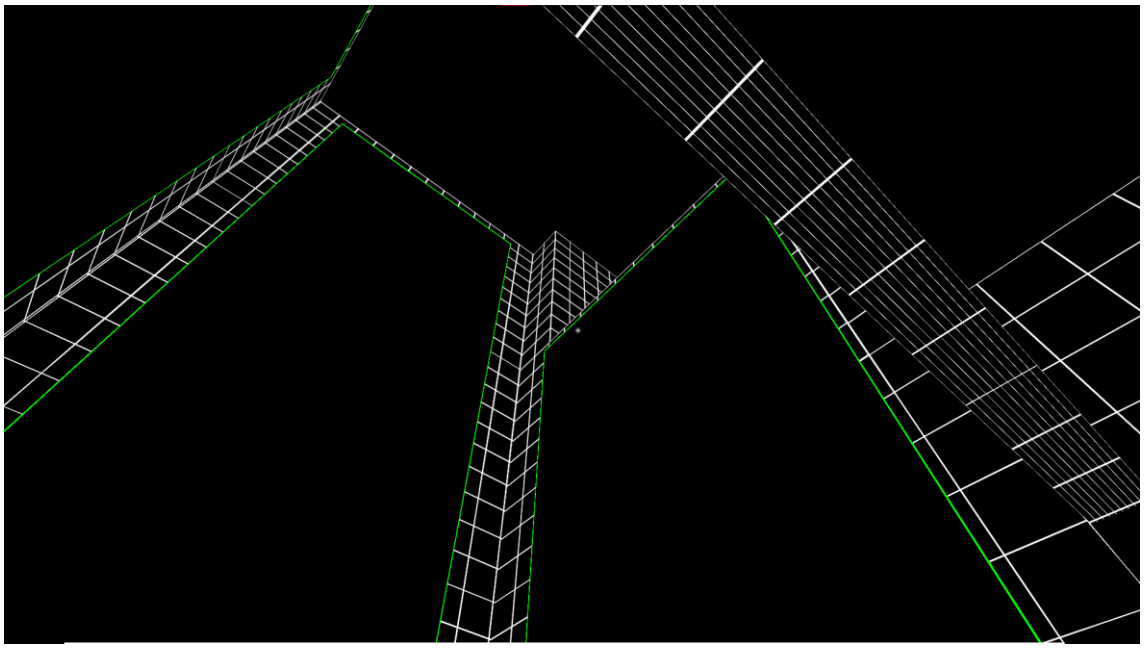


Figura 20: Camí sense sortida fàcil. El jugador haurà d'utilitzar el seu coneixement adquirit en aquest nivell si vol tenir èxit

Un cop superat aquesta zona ens trobem amb l'obstacle final, el qual acaba en una plataforma semblant a la del nivell anterior i el mateix missatge.

Finalment, el joc porta al jugador a una zona on, un altre cop, ha estat modificada per donar pas a un sostre amb el mateix patró que les parets i terra. Al ser l'últim nivell del joc, es on es mostra encara més la simulació en la que està atrapat el jugador. Un dels aspectes a destacar és l'aparició de senyals indicant el camí que el joc vol que es segueixi (veure Figura 21).

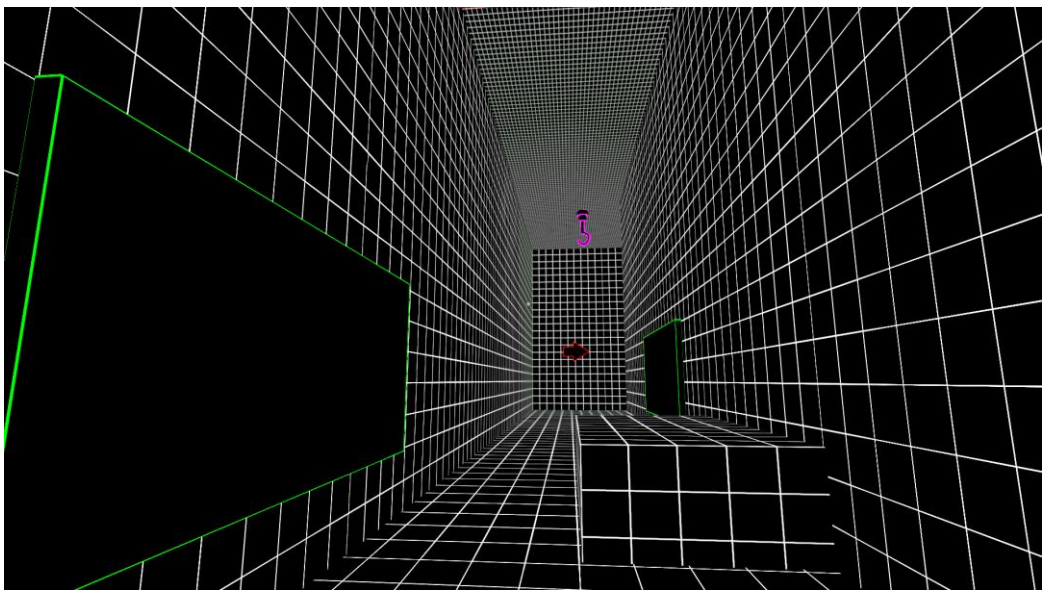


Figura 21: El sostre a agafat el mateix to que la resta de superfícies, a més de l'aparició de senyals d'indicació.

En avançar pels obstacles, podem veure que els objectes interactius estan molt més desordenats que a la resta de nivells, a més d'haver-hi molts més camins a escollir pel jugador. Aquests nivells van des de més fàcil accés fins a més difícil (veure Figura 22).

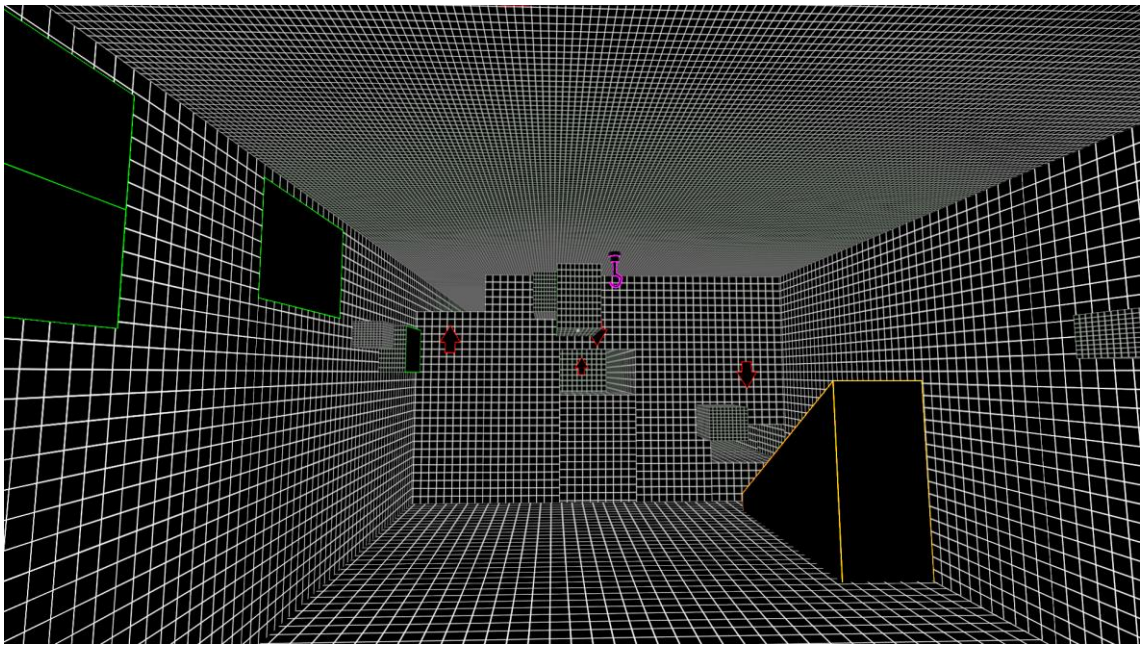


Figura 22: Zona on el camí es divideix.

Un cop escollit el camí, tots convergeixen en al mateix lloc, un espai molt gran amb moltes senyals guiant al jugador cap a la zona final del joc. A partir d'aquí el joc fa un intent de fer casi impossible el superar el nivell, indicant que no vol que s'arribi al final del nivell (veure figures 23, 24 i 25).

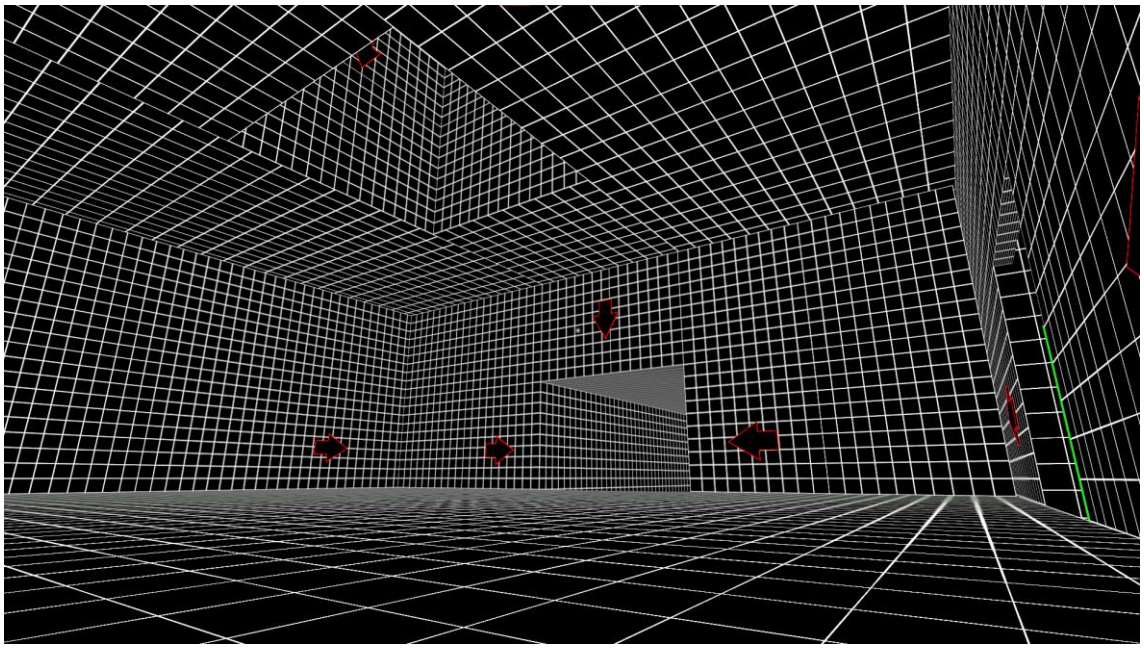


Figura 23: Zona ample indican el camí a seguir.

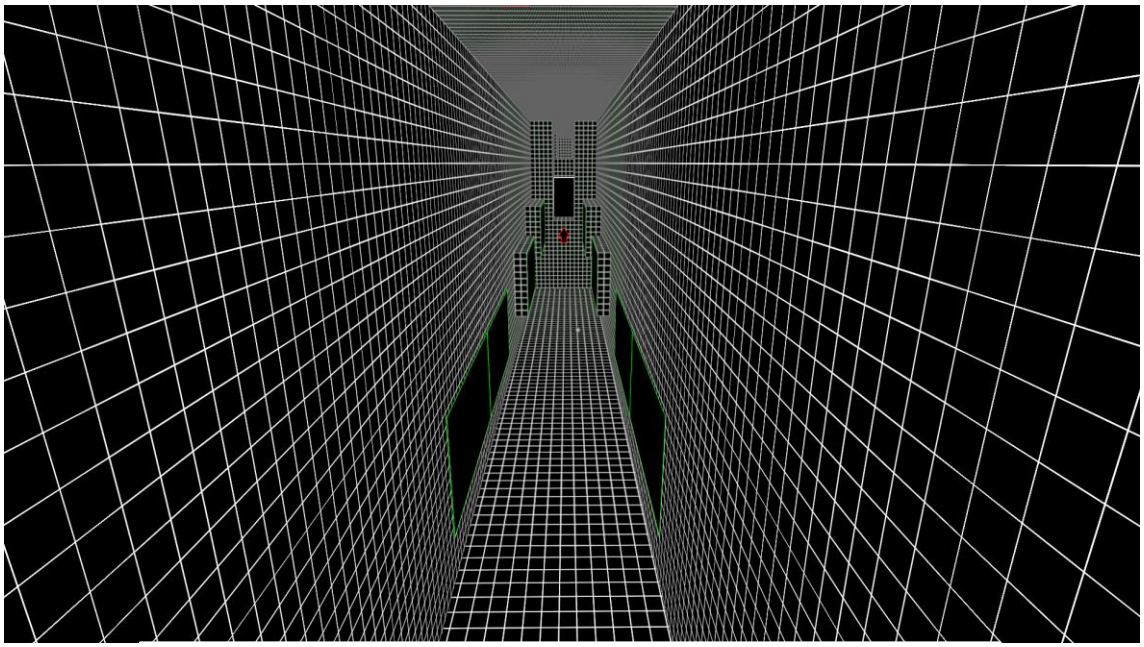


Figura 24: Obstacles d’alta dificultat que el jugador haurà de superar.

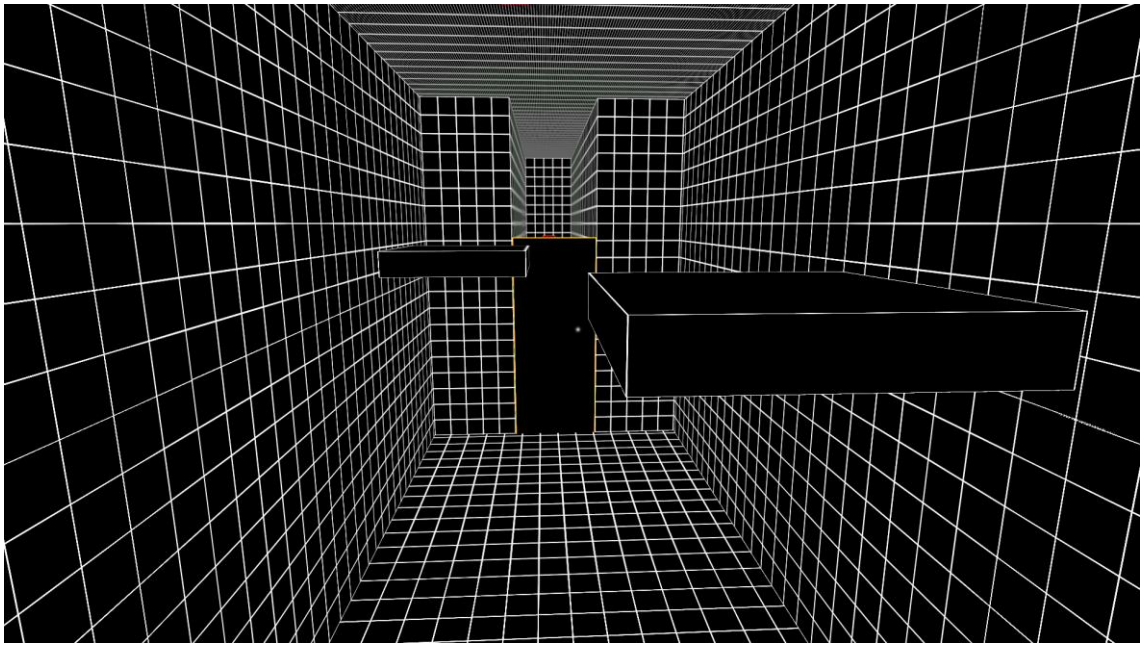


Figura 25: Zones amb plataformes mòbils.

Finalment, l’últim obstacle que veu el jugador es una paret amb un senyal que indica la sortida de la simulació. Al travessar-lo, el joc li donarà l’ultima decisió: Tornar a fer el nivell o ser lliure (veure figures 26 i 27).

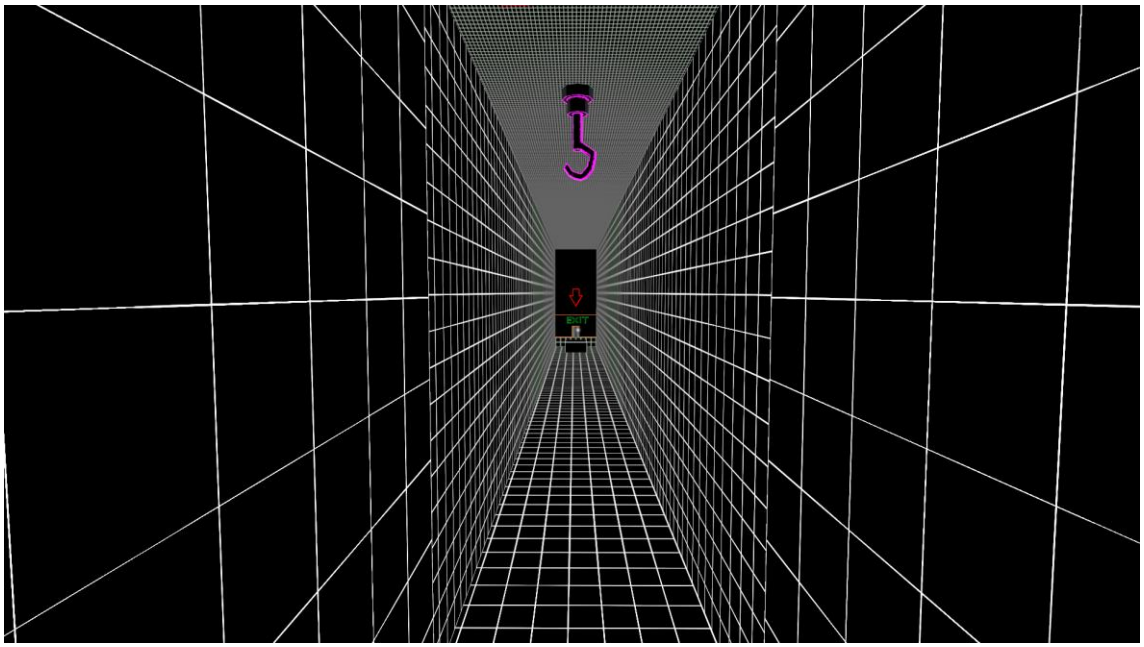


Figura 26: La sortida de la simulació.

Figura 27: Missatge del nivell final.

El disseny i l'estil del joc ha estat construït mitjançant el Software Blender, creant les estructures i textures adequades per anar donant forma als nivells segons el nostre criteri.

5.6.2 Personatge

Al joc, al ser en primera persona, hem prescindit de realitzar el disseny per al personatge. També una de les principals raons es a causa de la narrativa que es presenta en el videojoc. El protagonista l'hem pensat com una referència directa al jugador, fent que tota la situació en la que es troba en el joc sigui més personal i ajudi en la immersió.

5.7 Narrativa

A causa de que el joc es un prototip, hem decidit no fer l'apartat narratiu massa rellevant, encara que hem volgut afegir unes píndoles de pistes o recursos per afegir algun tipus de coherència al joc.

El protagonista, que es la representació directa del jugador, es veu atrapat en el que sembla un món virtual, al qual alguna cosa o persona el forçarà a superar els obstacles que hi ha al nivell per poder-hi sortir, però no sense abans donar-li vàries habilitats de parkour per poder tenir alguna oportunitat en els nivells que ha creat. A mesura que el jugador vagi superant nivells, podrà veure com cada cop aquest esser es va comportant més erràtic, frustrat amb el jugador per superar els seus obstacles, fet que es va mostrant a mesura que es va avançant pels nivells, cada cop amb objectes més desordenats i amb una estètica més virtual.

En el prototipus, hem creat només tres dels nivells que compondran el joc, però que donen una imatge general del estil i missatge del videojoc. Finalment, cal remarcar que la narrativa que es presenta té com a objectiu dotar d'un sentit i coherència tant a les mecàniques del jugador com a l'estètica del món.

5.8 Mons de joc

5.8.1 Dimensió física

Com s'ha esmentat en el apartat anterior, la dimensió física correspon a un espai generat virtualment. La grandària d'aquest espai en relatiu a la del mon real es podrien considerar equivalents, ja que al ser un joc de parkour, és important que el jugador pugui agafar referències clares respecte a les distàncies. El límits del mapa solen estar delimitats per parets invisibles i, en alguns nivells, parets sense cap tipus de relleu definit o sostres que impedeixin al jugador saltar molt alt.

Al ser un món virtual, hi destaquen molt les geometries bàsiques. Les que més s'utilitzen són les rectangulars i cúbiques, encara que poden haver-hi de més complexes. EL repte que té el jugador es la d'explorar aquestes geometries i utilitzar-les a seu favor per aconseguir superar els nivells.

Aquesta dimensió física va molt lligada a la narrativa ja que. Apart de ser un dels reptes del protagonista, també s'utilitza com a recurs per reforçar l'historia que es vol explicar al videojoc.

5.8.2 Dimensió temporal

El temps jugaria un rol relativament importat a causa que aquest joc ens situaria en un futur distòpic on seria possible atrapar a persones dintre de realitats virtuals. Pel que fa al transcurs del temps en el joc, aquest funciona de manera natural: el que es tarda en superar un nivell és el temps que el jugador inverteix en superar els obstacles.

5.8.3 Dimensió ambiental i emocional

Ens trobem en que el nivells son quasi tots contrasts entre el negre i blanc amb alguns tocs de colors primaris pel que respecta els objectes interactius. Volem que l'ambient transmeti emoció i una mica d'incertesa. Per una part els contrastos de colors respecte al paisatge donen aquell sentiment impulsiu de seguir-los mentre que la absència d'altres colors o de l'ambient donaran aquella sensació de inquietud i desconeixement.

Pel que fa a les emocions del jugador, volem que senti el repte que suposa superar els nivells establers i aquella necessitat primer de descobrir els misteris, a més de donar una mica de sensació d'angoixa de voler superar els nivell per poder sortir d'allà.

5.8.4 Referents estètics

Al llarg de la creació del videojoc hem agafat diferents referents estètics per al projecte. En l'estic gràfic de l'ambient i l'entorn, ens hems inspirat en el videojoc Maze War (1973), un dels primers jocs en primera persona i que utilitzava gràfics de vectors per aconseguir dibuixar formes geomètriques consistents. En el videojoc hem volgut realitzar una mena d'homenatge volent simular aquestes estructures més modernitzades (Veure Figura 28).

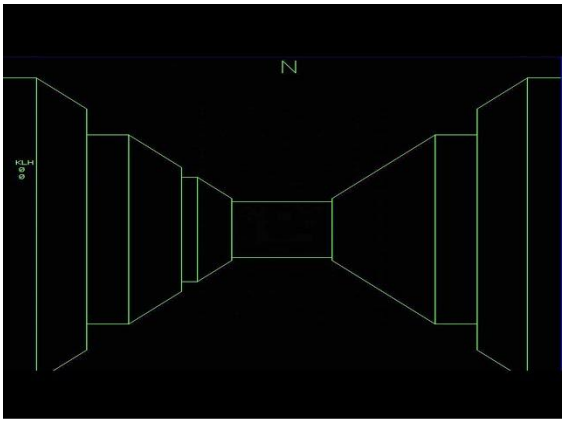


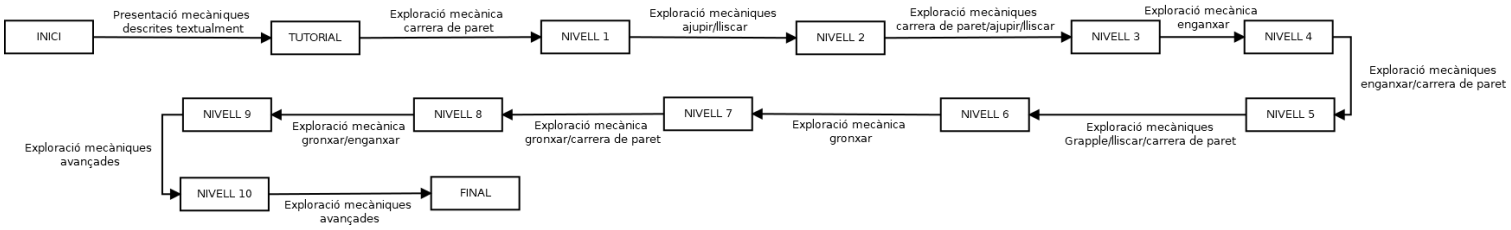
Figura 28: Imatge del gameplay Maze Wars.

Apart d'aquest, també hem agafat inspiració tant estètica com de disseny de nivells el videojoc Ghostrunner (2020), ambientat en un món distopic i en el que la humanitat està atrapada en una simulació. Aquest videojoc ens ha ajudat molt a entendre com fer nivells de parkour en

primera persona que fossin un repte però no injustos. També l'us dels colors per indicar el camí ens va servir d'inspiració per unir les dues estètiques dels jocs esmentats i millorar el producte final (veure Figura 29).

Figura 29: Imatge d'un nivell del Ghostrunner.

5.9 Estructura de nivells



Cada nivell requereix explorar les diverses mecàniques específiques que s'han presentat al començament del joc. Cada nivell requerirà un repte més avançat que l'anterior, ja que la dificultat anirà augmentat progressivament a mesura que el jugador avanci. Com hem esmentat anteriorment, en el prototip només hem creat el tutorial, el nivell 1 i el nivell 10 per donar al jugador una tast del videojoc.

5.10 Estructura del projecte

5.10.1 Estructura d'escenes

Tenint clar que el projecte creat és un prototip, el joc consta de tres escenes interconnectades entre si (veure Figura 30):

- Menú principal: És la primera escena que ens trobem al iniciar el videojoc. Apart del títol, inclou varies opcions que pot triar el jugador:
  - "New game": Comença el joc des del nivell tutorial.
  - "Level select": Permet escollir el nivell que es vulgui jugar.
  - "Settings": Varies opcions que permeten al jugador modificar varis aspectes del joc com el so, la sensibilitat, resolució...
  - "Quit": Surt del joc
- Joc: Escena principal on es troben tots els elements del videojoc. Permet tornar al menú principal sempre que es vulgui.
- Menú final: Escena final del projecte, indica que s'ha arribat al final i inclou opcions per tornar al menú, o bé sortir.

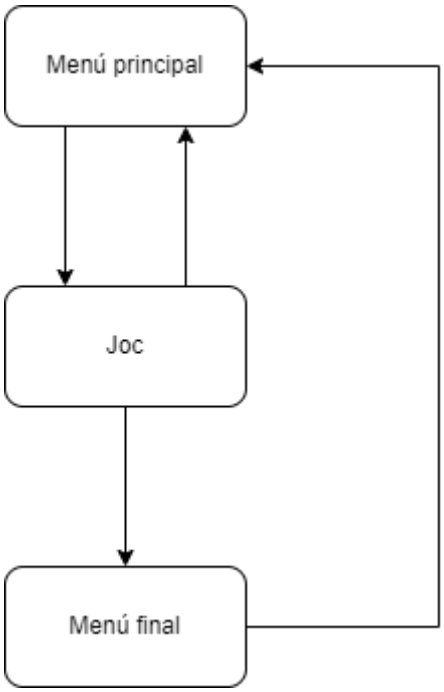


Figura 30: Diagrama que mostra les escenes i les seves connexions.

5.10.2 Menús

En aquest apartat, explicarem en detall els diferents menús que el jugador podrà trobar en el joc i interactuar entre ells, sempre mantenint una coherència. En el videojoc en podem trobar 5 tipus diferents de menús (amb els respectius sub-menús): el menú inicial, el menú d'opcions, el menú de pausa, el menú de final de nivell i el menú de final del joc.

Menú inicial:

En el menú inicial es presenta un fons animat negre amb dues llums de neons que formen un rectangle on es troben les opcions a escollir esmentades anteriorment (veure l'Apartat 5.10.1). Just a sobre de les opcions, podem veure el títol del videojoc amb les lletres del mateix color que les llums de les línies de fons. Hem utilitzat la font "Ethnocentric" al titol per donar-li un to futurista, mentre que a les opcions hem utilitzat la "Liberation sans" que sol ser la predeterminada en el motor Unity.

Al passar el cursor per qualsevol de les opcions, el jugador veurà que canviaran de color per indicar que han estat seleccionats. Al prémer qualsevols dels botons, s'escoltarà un so indicant que han estat seleccionats.
Si s'utilitza control per jugar, sempre s'il·luminarà la primera opció indicant a on es troba el jugador amb la possibilitat de triar un altre botó amb els controls. El so de selecció és el mateix que si es prem amb el cursor (Veure Figura 31).

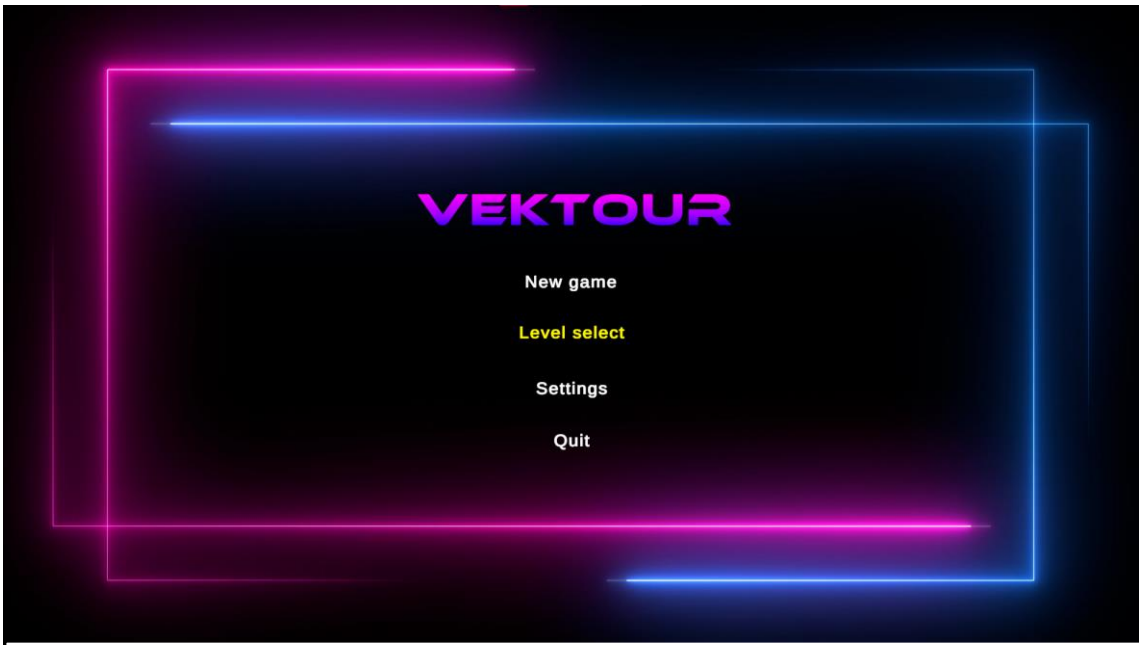


Figura 31: Menú principal.

Al prémer el botó de "New game", desapareixeran totes les opcions anteriors i apareixerà una pestanya confirmant la selecció. Aquesta pestanya és un rectangle negre amb un bordat verd viu, mantenint l'estil neó futurista del joc. La font del text es manté igual en tots els menús. (veure Figura 32).

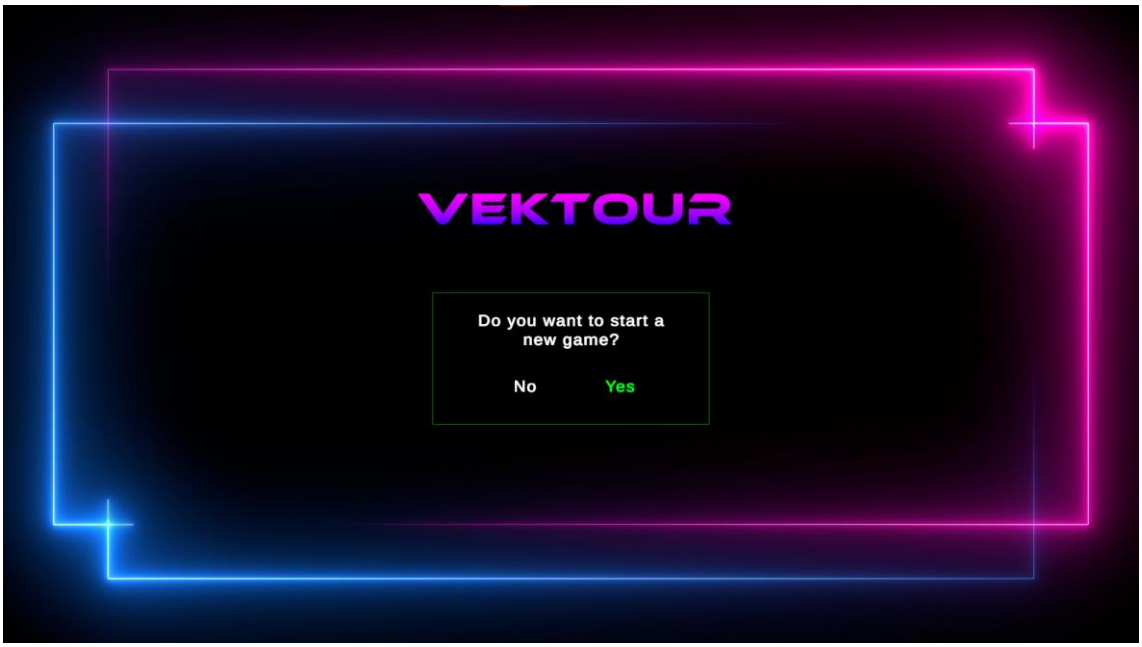


Figura 32: Confirmació “New game”.

L’opció “Level select” fa que aparegui una pestanya semblant a la selecció anterior però, en aquest cas, el rectangle es una mica més gran i està resseguit pel color groc. En aquesta finestra hi ha varies opcions dels nivells que hi ha en el joc per a que el jugador pugui seleccionar els que més li agradi (veure Figura 33). A l’haver fer només 3 nivells, només apareixien disponibles els nivells 1 i 10, deixant la resta amb interrogants. El nivell tutorial només pot ser jugat quan es comença una partida nova, ja que assumim que el jugador que vulgui triar un nivell ha entès els conceptes bàsiques de les mecàniques.

Figura 33: Finestra al prémer “Level select”.

Al seleccionar l’opció “Settings” s’obrirà la pestanya de les opcions del joc, on el jugador podrà modificar algunes variables per fer l’experiència com a jugador més fàcil. Més endavant es parlarà més en profunditat sobre el menú opcions.

Finalment, si es selecciona el botó “Quit”, apareixerà una finestra semblant a la de “New game” demanant confirmació amb la diferència de que el relleu del rectangle serà vermell (veure Figura 34). Aquest botó permet al jugador sortir del videojoc l’escriptori de l’ordinador.

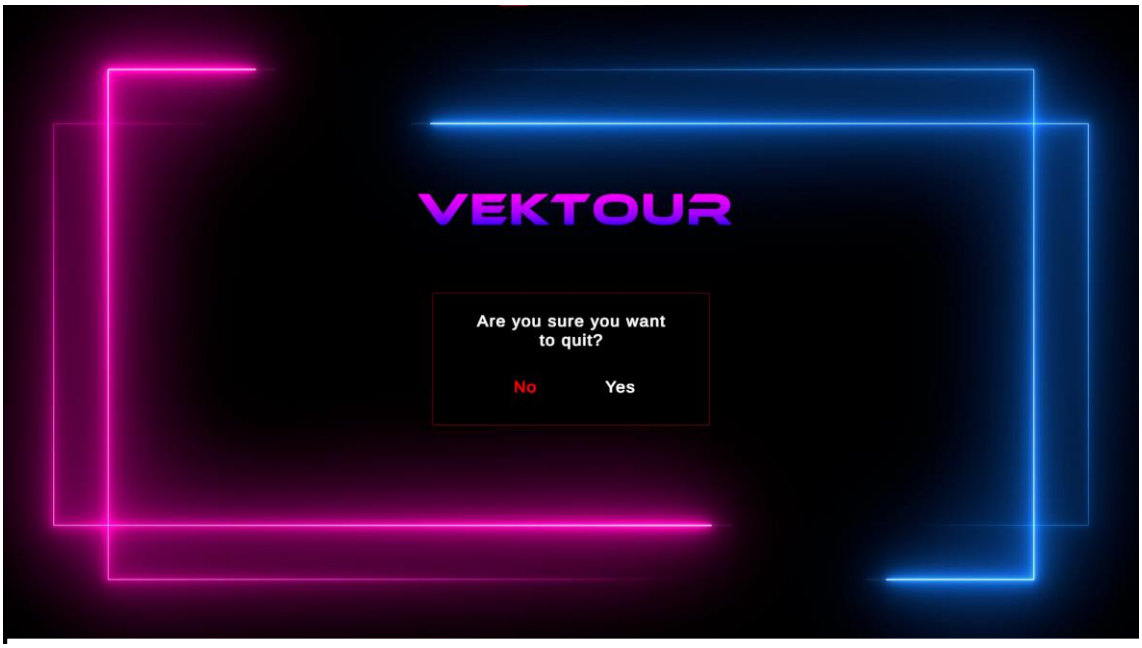


Figura 34: Confirmació “Quit”.

Menú d’opcions:

Aquest menú es troba dintre de dos altres menús del joc: el menú inicial i el menú de pausa, encara que l’estil i el contingut és exactament el mateix pels dos.

La pestanya “Settings” està estructurada igual que el menú d’inici, cada botó s’il·lumina amb un color cada cop que es passa el cursor o es selecciona amb el control, al prémer un d’ells fa un so de confirmació com a la resta de menús (veure Figura 35).

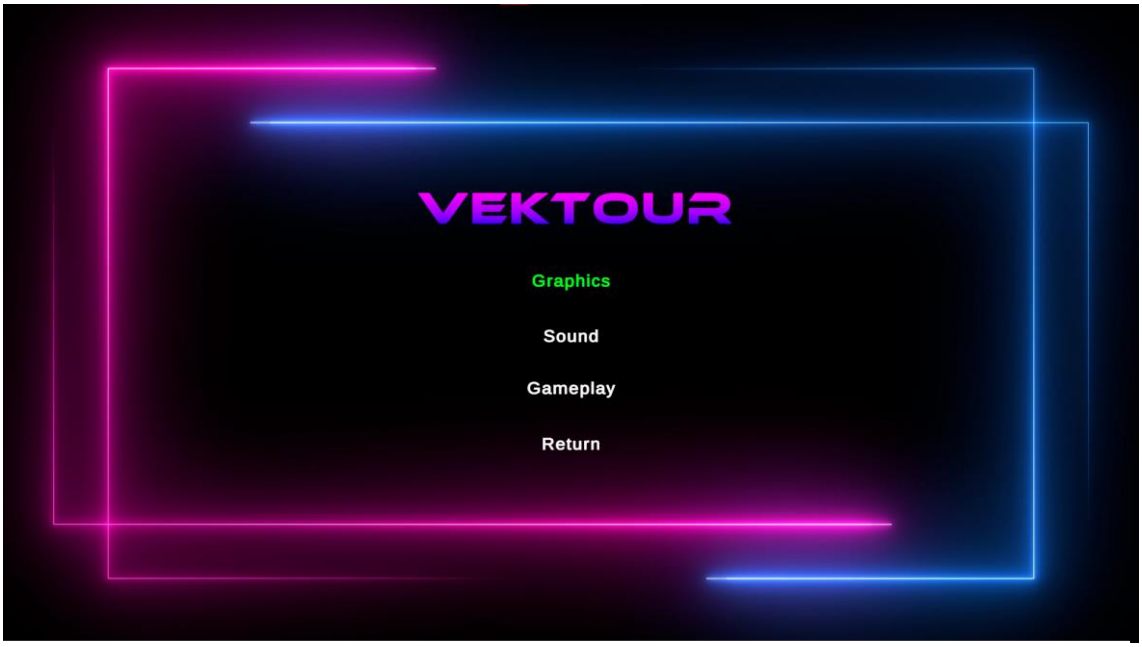


Figura 35: Menú “Settings” dintre del menú inicial

Cal destacar que el fons del menú de “settings” és el mateix que el del menú inicial perquè només hem afegit els botons mentre que hem amagat els de l’inici. En el menú pausa no hi surt ja que estem en un altre menú (veure Figura 36).

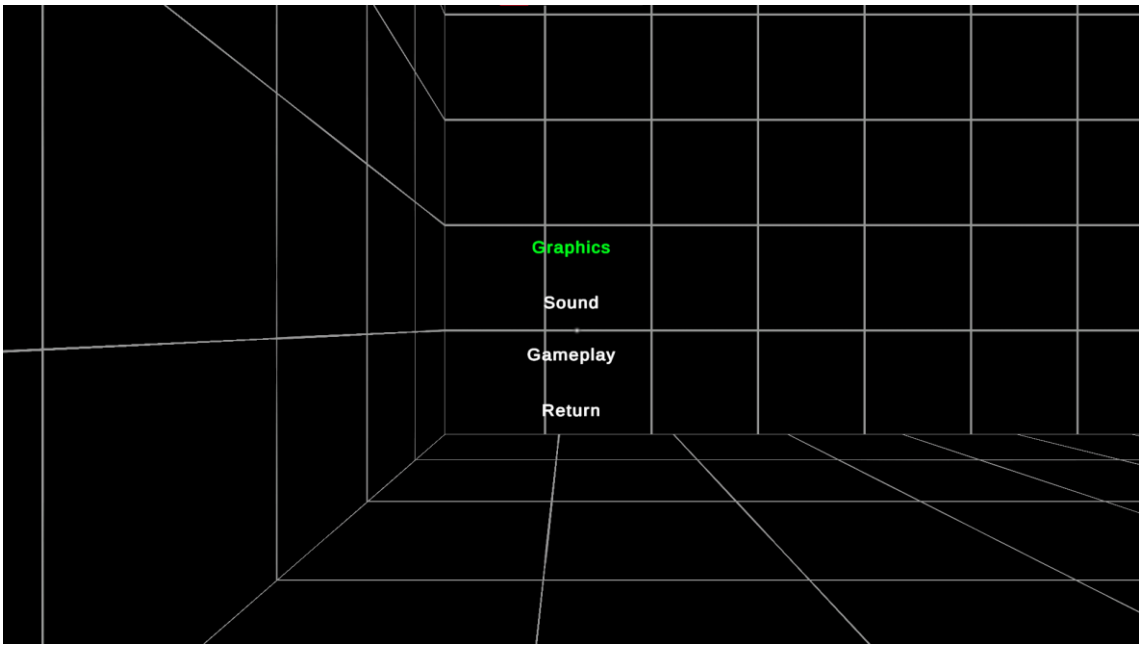


Figura 36: Menú “Settings” dintre del menú pausa.

Al seleccionar l’opció de “Graphics”, apareix una finestra rectangular de fons negre amb relleu blau. Dintre d’aquest, podrem trobar dos desplegables els quals serveixen tant per establir la resolució del joc com la qualitat dels gràfics. Al fer clic en els desplegables, hi haurà una llista d’opcions a triar, al passar el cursor canviaran de color a un taronja fort i, al seleccionar una d’aquestes, farà un so. També ens trobem amb una caixa la qual podrem activar per decidir si volem el nostre joc en pantalla completa o no. Al prémer, canviarà de color a un verd suau (veure Figura 37).

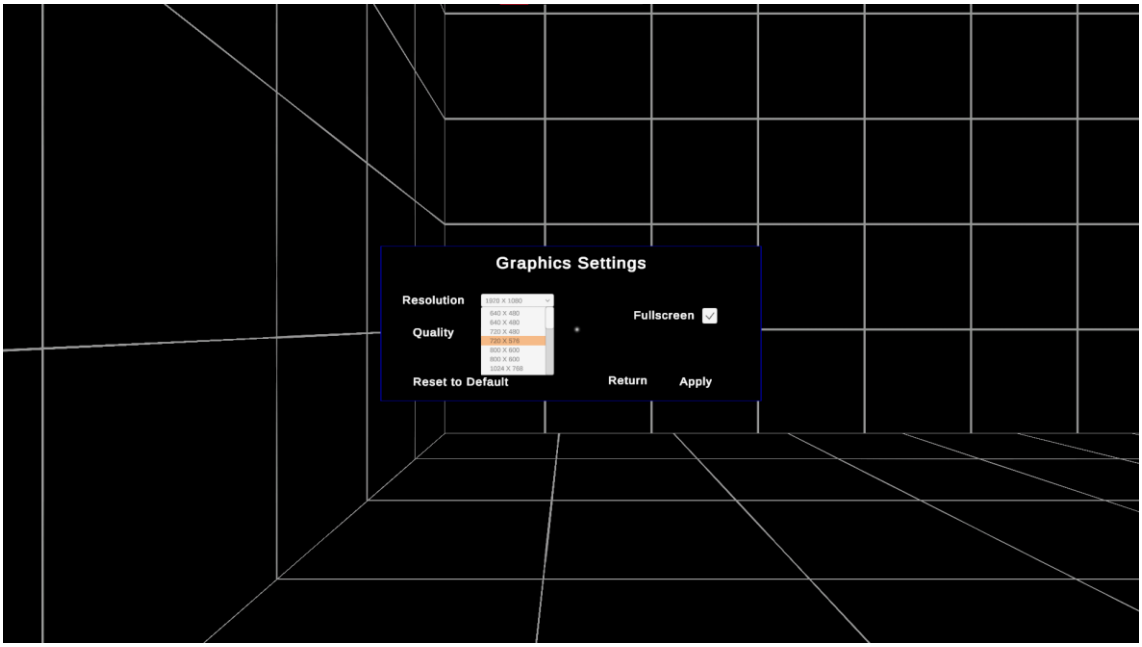


Figura 37: Pestanya “Graphics” dintre del menú opcions.

En l'opció de "Sound", apareixerà una finestra semblant a les altres però amb el relleu de color groc. Dintre d'aquests podem veure tres Sliders que modifiquen els diferents sons del joc. A l'interactuar amb ells, veiem que el valor de la dreta va canviant, indicant el volum a més d'anar afegint el color violeta al Slider per indicar visualment a quin nivell està el so (veure Figura 38).

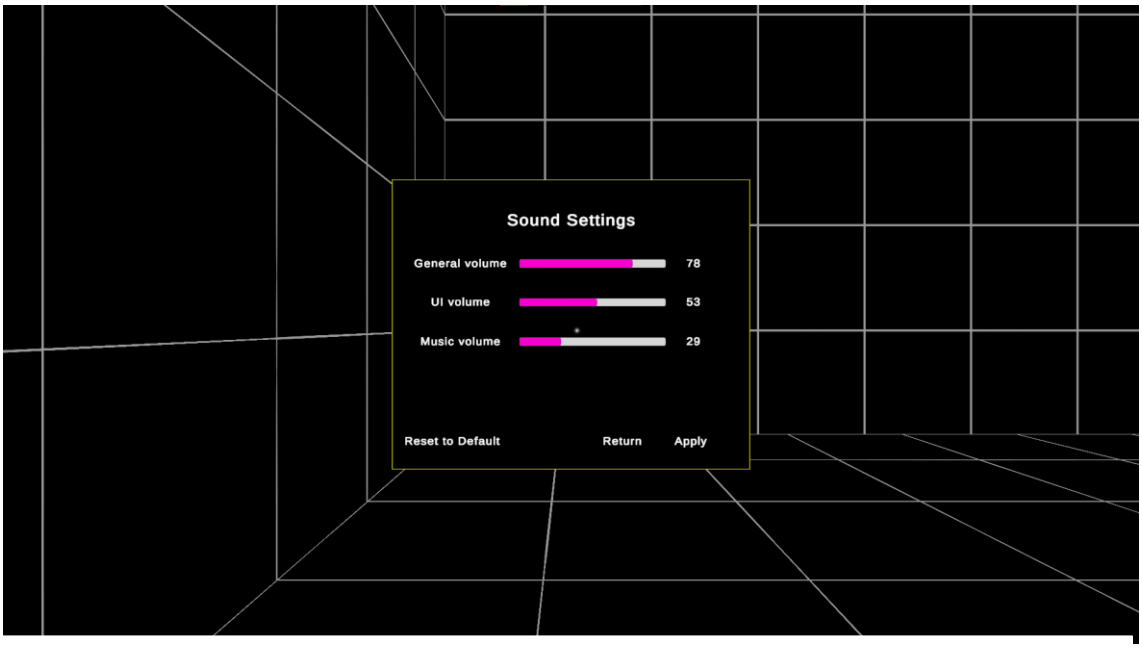


Figura 38: Pestanya "Sound" dintre del menú opcions.

A l'opció de Gameplay apareix una finestra amb el relleu blau, dintre hi trobem un Slider (semblant al de la finestra "Sound") per indicar la sensibilitat de la càmera, una caixa que pot ser activada per invertir el control vertical de la càmera i un botó que, al prémer, es mostra una imatge amb els controls tant de teclat i ratolí com de consola (veure figures 39 i 40).

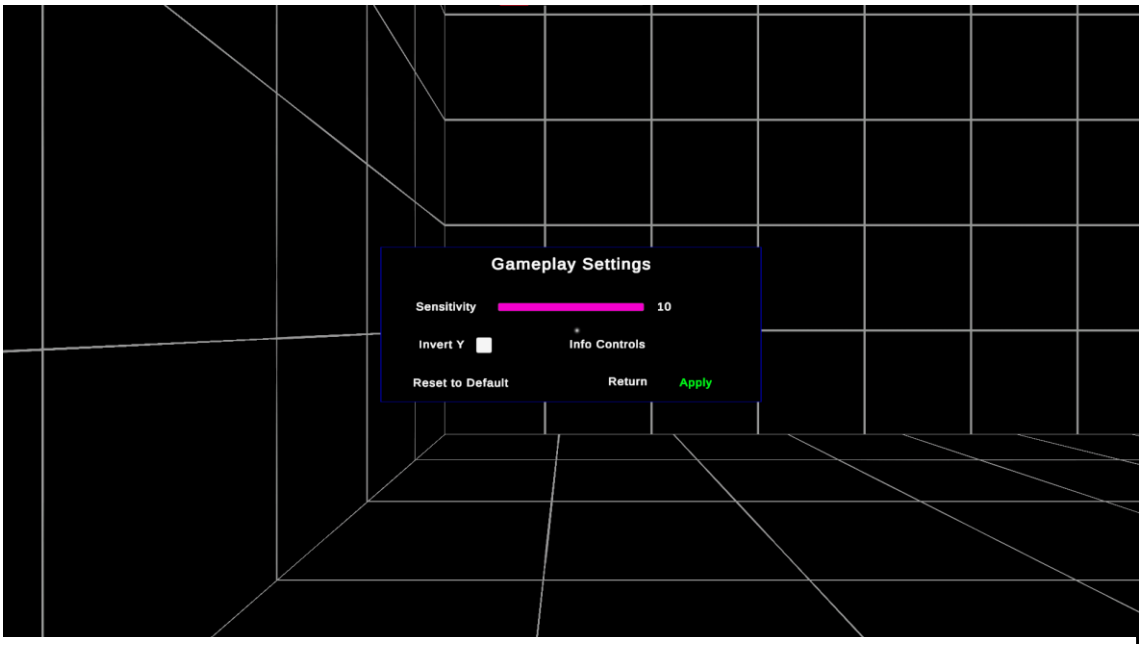


Figura 39: Pestanya "Gameplay" dintre del menú opcions.

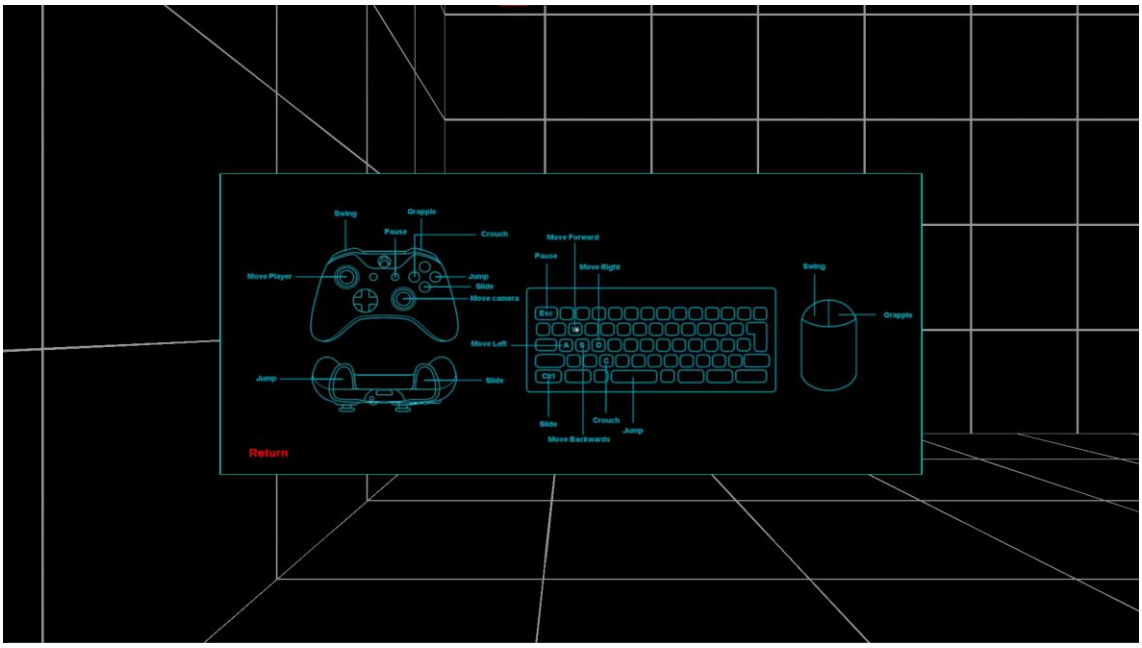


Figura 40: Imatge dels controls al prémer el botó “Info Controls”.

Menú de pausa:

El menú de pausa, que el trobem quan volem pausar el joc, no té establer un fons fix, ja que el jugador en tot moment ha de poder veure al personatge per així tenir una referència sobre la situació en la que es troba el joc. Hi podem trobar tres botons diferents, un per continuar, un altre per sortir (amb una pestanya de confirmació) i un altre per anar al menú d’opcions. El disseny dels botons són els mateixos que en els del menú principal i el d’opcions, simplement hem agafat el botó predeterminat de Unity i l’hem tret el requadre per deixar només el text amb la tipografia predeterminada (veure Figura 41). Com la resta de menús, canvien de color quan es passar el cursor per sobre o quan es selecciona amb el controlador de consola i fa un soroll de confirmació al clicar en ells.

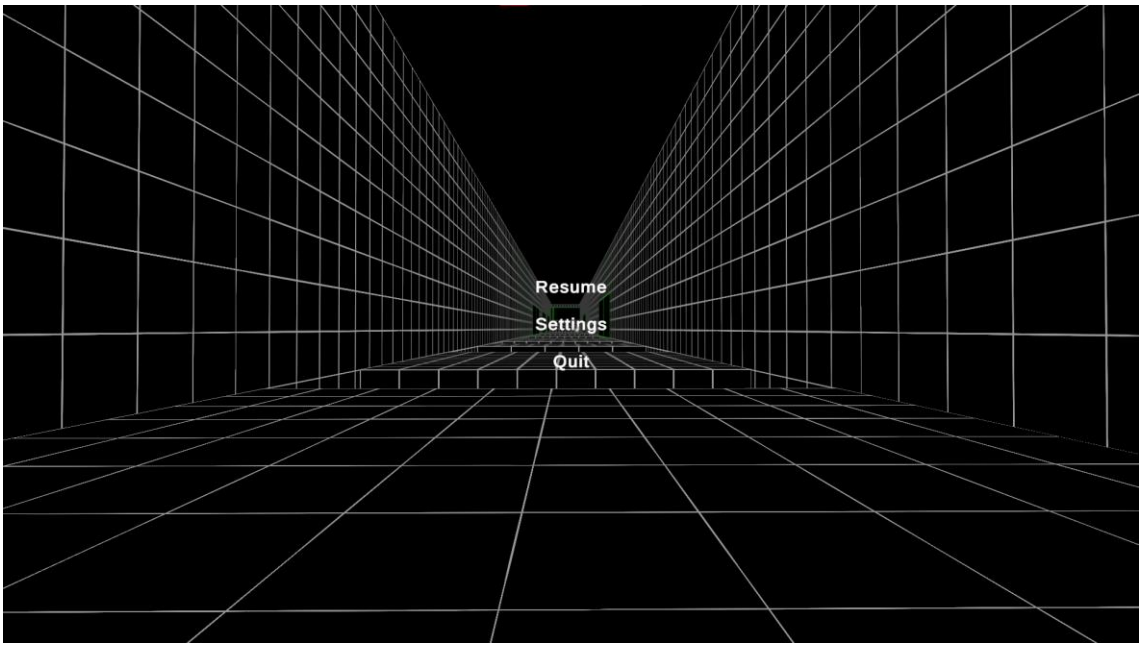


Figura 41: Menú de pausa.

Menú final de nivell:

A l'arribar al final d'un nivell, apareix una finestra preguntant al jugador si vol repetir el nivell o avançar al següent (o en el cas de l'últim nivell si vol arribar a l'escena final). La finestra es un rectangle negre amb un relleu de color verd, molt semblant a les finestres del menú d'opcions (veure Figura 42).

Figura 42: Menú de final de nivell.

Menú final de joc:

Per últim, ens trobem el menú final que surt quan hem finalitzat el joc. En aquest cas les úniques coses que hi ha a l'escena són: un fons completament negre, un text preguntant si vol sortir del jo o anar al menú i dos botons iguals que la resta que indiquen si anar al menú inicial o sortir a l'escriptori de l'ordinador (veure Figura 43). Tant la tipografia, com els botons, com el so de confirmació, son exactament iguals que la resta de menús. L'única cosa que el diferencia de la resta es el fons completament obscur i un so que s'escolta de fons que afegeix una mica de misteri a l'escena.

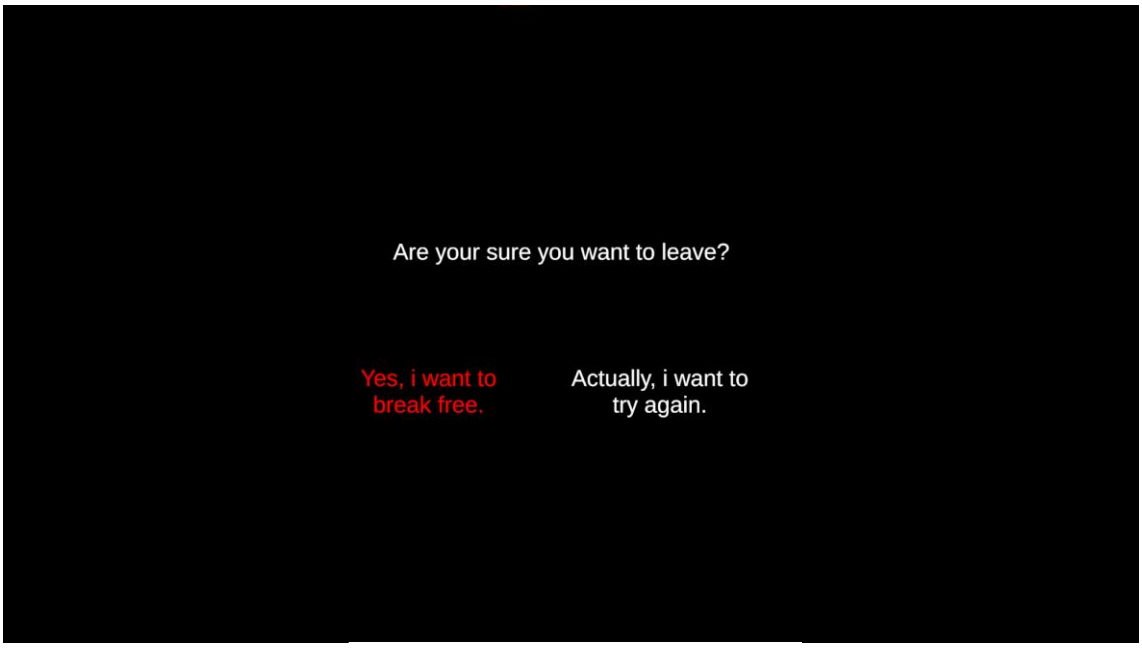


Figura 43: Menú de final de joc.

5.10.3 Il·luminació

Pel que fa la il·luminació, en el nostre cas hem volgut prescindir de fonts de llum convencionals, ja que volíem que el nostre joc fos el contrasts de negre amb els colors del objectes. Per fer-ho, vam utilitzar una llibreria anomenada "Post Processing" (veure Figura 44) que ens va permetre utilitzar el contrast dels colors en les textures dels objectes amb el fons per fer que el jugador pogués veure sense necessitat d'una font de llum. El resultat final ofereix una il·luminació estil neó pels objectes molt característica i acord amb l'estil que estàvem buscant.

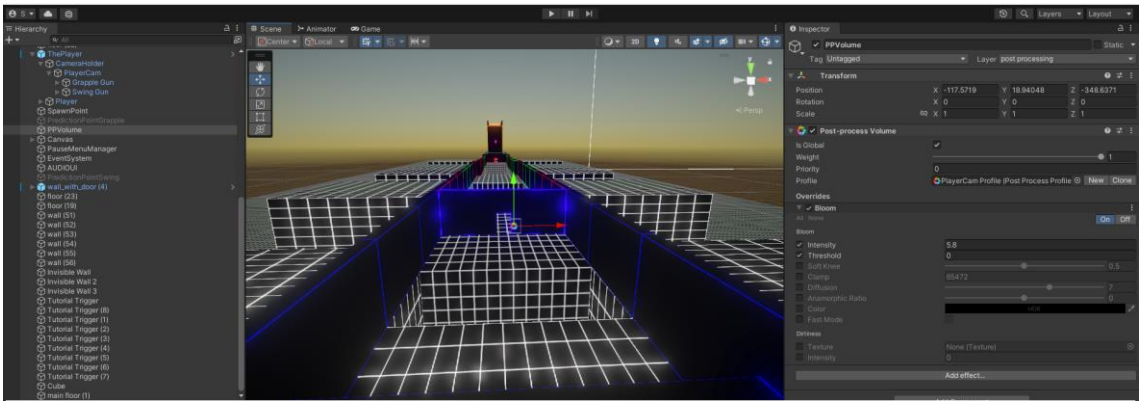


Figura 44: Element Post-Processing i la vista general de l'escena.

5.10.4 Disseny de so

Encara que el disseny de so no l'hem considerat un dels aspectes centrals del projecte, ha estat implementat amb l'objectiu de millorar l'experiència del jugador. La música de fons s'ha incorporat amb la intenció d'oferir al jugador la millor experiència immersiva a més d'ajudar-lo a centrar-se en els reptes que es trobi. Cada nivell conté una pista de música diferent, que intenta mantenir al jugador centrat i animat en tot moment.

A més de la música, hem incorporat sons per als botons del menús creats. Aquesta elecció proporciona una retroalimentació auditiva, així com millorar la interactivitat i experiència de l'usuari al navegar per les diferents interfícies.

5.10.5 Personatge i càmera

El personatge està compost per varis components: El "CameraHolder" i l'objecte "Player"

CameraHolder:

Aquest component es el que s'encarrega del moviment de la càmera del jugador. Dintre d'aquest component hi podem trobar d'altres, com per exemple els objectes per on surten els efectes de les mecàniques de gronxar-se i enganxar-se. Aquest component està col·locat just a sobre del collider del Player

Player:

Aquest component correspon a tot el corresponent amb el moviment del jugador i interacció d'aquest amb l'entorn. Al ser un joc en primera persona, ens va semblar innecessari la creació i disseny d'un personatge, ja que mai el podrà veure el jugador. Per aquesta raó, hem utilitzat la forma geomètrica de càpsula (la predeterminada de Unity) per a utilitzar com a referència (Veure Figura 45).

Aquest component està format per:

- "PlayerObj": Component on hi ha el collider.
- "Orientation": Component que indica cap a on està orientat el jugador.

- “POV”: Component que utilitza el “CameraHolder” per saber on apuntar la càmera.

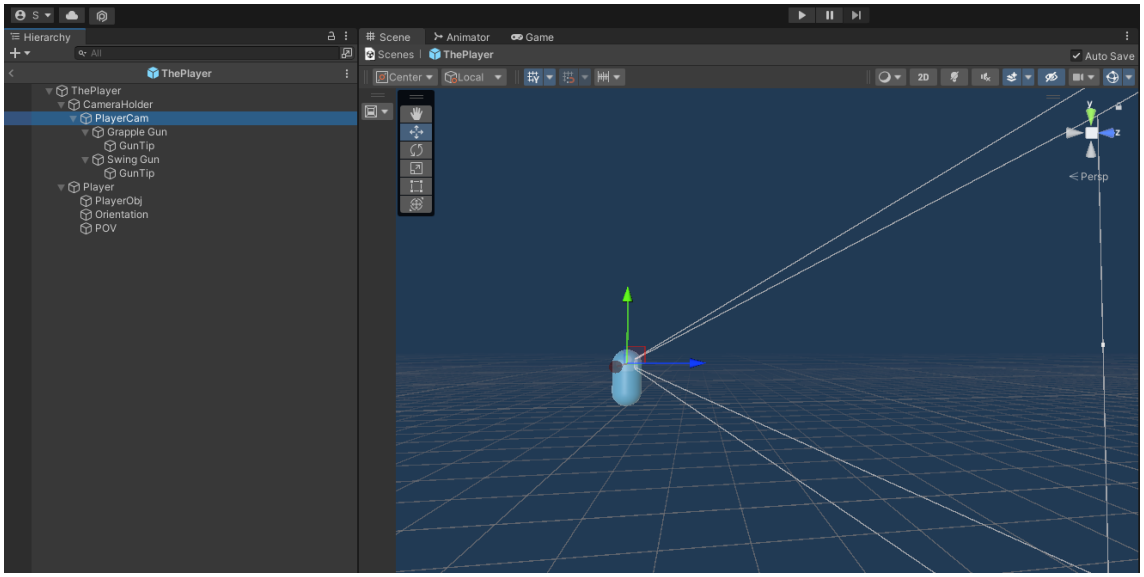


Figura 45: Personatge amb els component que el formen.

6. Implementació i proves

6.1 Implementació de nivells

6.1.1 Estructura de les escenes

El joc està format per un conjunt d'escenes independents, on cada una d'elles representa un nivell o pantalla de joc. A mesura que el jugador va avançant en el joc, anem canviant d'escena i carregant els element que formen cada una. En el projecte hi ha un total de 4 escenes que formen el joc, ordenades per ordre cronològic, tot i que en alguns casos tornem a escenes anteriors (veure Figura 46).

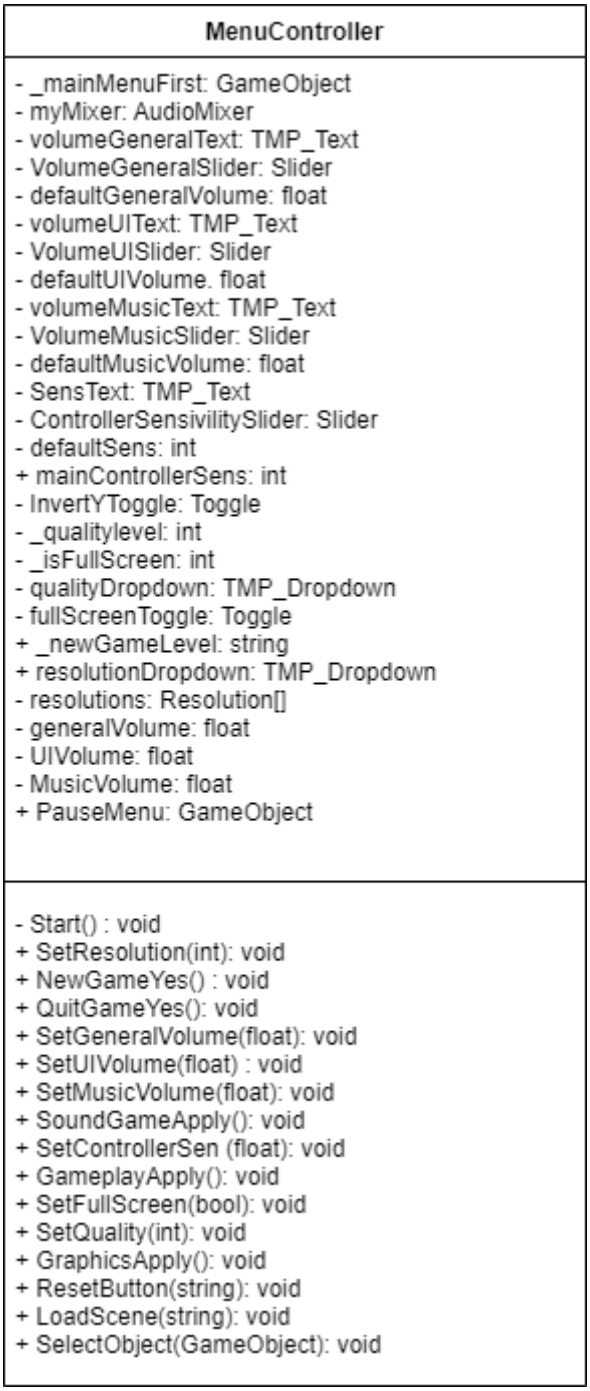
Figura 46: Estructura de les escenes del videojoc.

L'escena Menu és la primera escena que es carrega quan s'inicia el joc. Aquesta escena es l'encarregada de mostrar per pantalla el títol i els botons amb el que el jugador hi podrà interactuar. A més, també es l'encarregada de carregar els valors que s'han inserit anteriorment en el apartat de Settings (en cas que s'hagi accedit al joc anteriorment) i de guardar-les en cas que s'hagin modificat. Els dos objectes encarregats de realitzar aquestes tasques són el MenuController i el LoadPrefs.

6.1.2 MenuController

La classe MenuController és l'encarregada de gestionar tot el menú principal i els botons que hi apareixen: Ajustar volum, resolucions, sensibilitat dels controls, començar partida nova, sortir del joc, etc.

Model de la classe MenuController:



Atributs de la classe MenuController:

- **_mainMenuFirst:** GameObject que serà seleccionat tal qual s'iniciï l'escena. Pensat per navegar amb els controls de comandament de consola.

- **myMixer:** El component AudioMixer que utilitzarem per administrar els diferents efectes de so.

- **volumeGeneralText:** Text que indica el nivell del volum general del menú de Settings. Va des del 0 al 100.

- **VolumeGeneralSlider:** Correspon al Slider del volum general en el menú de Settings. És el que el jugador utilitzarà per ajustar el volum del so.

- **defaultGeneralVolume:** El valor per defecte del volum general. En aquest cas correspon al número 100.

- **volumeUIText:** Text que indica el nivell del volum dels efectes de so del joc (quan es clica un botó, al utilitzar la mecànica de gronxar-se o de enganxar-se) del menú de Settings. Va des del 0 al 100.

- **VolumeUISlider:** Correspon al Slider del volum dels efectes de so del joc. És el que el jugador utilitzarà per ajustar el volum del so.

- **defaultUIVolume:** El valor per defecte del volum dels efectes de so del joc. En aquest cas correspon al número 100.

- **volumeMusicText:** Text que indica el nivell del volum de la música dels nivells. Va des del 0 al 100.

- **VolumeMusicSlider:** Correspon al Slider del volum de la música dels nivells. És el que el jugador utilitzarà per ajustar el volum del so.

- **defaultMusicVolume:** El valor per defecte del volum de la música dels nivells. En aquest cas correspon al número 100.

- **SensText:** Text que indica el nivell de la sensibilitat del moviment de la càmera del jugador. Va des del 1 al 10.

- **ControllerSensivilitySlider:** Correspon al Slider de la sensibilitat de la càmera. És el que el jugador utilitzarà per ajustar la sensibilitat.

- **defaultSens:** Valor per defecte de la sensibilitat del control de la càmera amb el mouse. En aquest cas correspon al numero 10.

+ **mainControllerSens:** Valor per defecte de la sensibilitat del control de la càmera amb els controls de comandament de la consola. En aquest cas correspon també al numero 10, poden modificar aquest valor des de l'inspector del Unity per a fer proves.

- **InvertYToggle:** Component Toggle que serveix per indicar si el jugador vol el moviment de la càmera vertical invertit o no.

- **_qualitylevel:** Valor que estableix la qualitat dels gràfics del videojoc. Sent 0 els "low" gràfics i 3 els "ultra".

- **_isFullScreen:** Booleà que indica si el joc està en pantalla completa o en mode finestra.

- **qualityDropdown:** Component tipus Dropdown on hi estan guardats els diferents nivells de qualitat (low, medium, high, ultra).

- **fullScreenToggle:** Component Toggle que serveix si el jugador vol el videojoc en pantalla completa o en mode finestra.

+ **_newGameLevel:** String on hi anirà el nom del primer nivell que hi jugarà el jugador, podent-lo modificar des de l'inspector del Unity per a fer proves.

+ **resolutionDropdown:** Component tipus dropdown on és mostraran les diferents resolucions que hi pot escollir el jugador. Poden ser modificat des de l'inspector de Unity per facilitar la realització de proves.

- **resolutions:** Un array on es guardaran les resolucions que el jugador hi podrà triar depenent del monitor.

- **generalVolume:** float on es guarda el valor actual del volum general.

- **UIVolume:** float on es guarda el valor actual dels efectes de so del joc.

- **MusicVolume:** float on es guarda el valor actual del volum de la música dels nivells.

+ **PauseMenu:** Objecte on hi guardarem tot el menú de pausa.

En la Figura 47 podem veure les propietat públiques assignades al component de la classe MenuController.

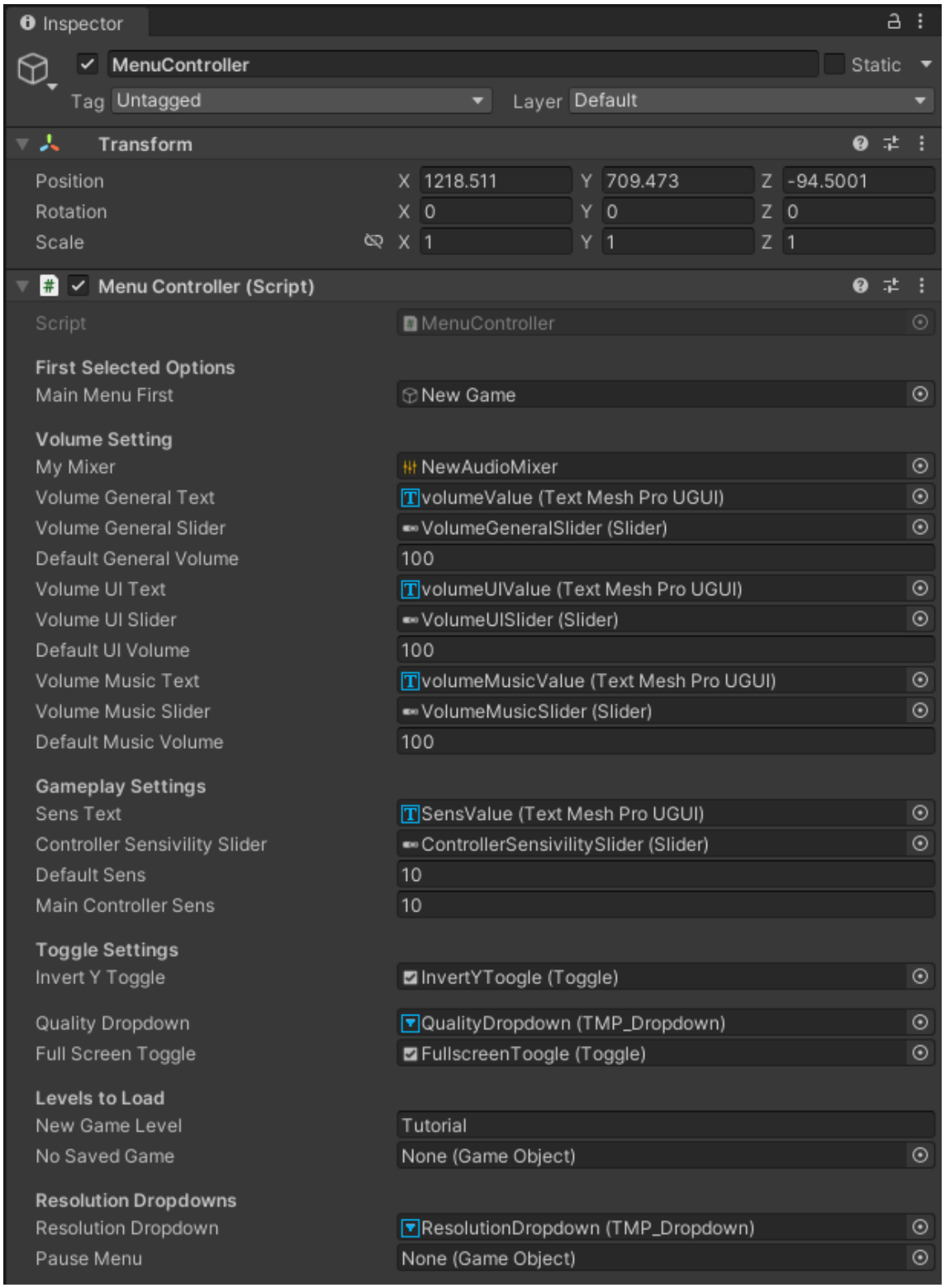


Figura 47: Propietats del component de la classe MenuController. L'atribut Pause Menu figura buit per que hem agafat l'objecte del Menú principal.

Mètodes de MenuController:

- **Start():** Funció cridada al primer frame del motor de Unity. Estableix com a objecte seleccionat el _mainMenuFirst (per poder utilitzar els controls de comandament de consola), mostra el cursor en la pantalla si no hi estava abans, estableix els gràfics del videojoc i s'encarrega d'omplir el Dropdown de les resolucions que pot triar el jugador segons el monitor.

```
private void Start()
    {
        EventSystem.current.SetSelectedGameObject(_mainMenuFirst);
        Cursor.lockState = CursorLockMode.None;
        Cursor.visible = true;
        SetQuality(QualitySettings.GetQualityLevel());
        resolutions = Screen.resolutions;
        resolutionDropdown.ClearOptions();

        List<string> options = new List<string>();

        int currentResolutionIndex = 0;

        for(int i=0;i<resolutions.Length;i++)
        {
            string option = resolutions[i].width +" X " + resolutions[i].height;
            options.Add(option);

            if(resolutions[i].width == Screen.width && resolutions[i].height == Screen.height)
            {
                currentResolutionIndex = i;
            }
        }

        resolutionDropdown.AddOptions(options);
        resolutionDropdown.value = currentResolutionIndex;
        resolutionDropdown.RefreshShownValue();


    }
```

- **SetResolution(int resolutionIndex):** Estableix la resolució del videojoc donada pel jugador al triar-la del Dropdown resolutions.

```
    public void SetResolution(int resolutionIndex)
    {
        Resolution resolution = resolutions[resolutionIndex];
        Screen.SetResolution(resolution.width, resolution.height, Screen.fullScreen);
    }
```

- **NewGameyes():** Inicia una nova partida carregant el nivell definit en l'atribut _newGameLevel i activa de nou els texts que hi apareixen en el tutorial.

```
public void NewGameyes()
{
    PlayerPrefs.SetFloat("TutorialCompleted", 0);
    SceneManager.LoadScene(_newGameLevel);
    EventSystem.current.SetSelectedGameObject(null);

}
```

- **QuitGameYes():** Funció que tanca la aplicació.

```
public void QuitGameyes()
{
    Application.Quit();
}
```

- **QuitGameMenu():** Envia al jugador al menú principal des del menú de pausa.

```
public void QuitGameMenu()
{
    PauseMenu.GetComponent<PauseMenu>().Resume();
    SceneManager.LoadScene("Menu");
}
```

- **SetGeneralVolume(float volume):** Estableix el volum del grup "master" dintre del AudioMixer en el valor donat.

```
public void SetGeneralVolume(float volume)
{
    generalVolume = volume;
    myMixer.SetFloat("master", Mathf.Log10(volume) * 20);
    float volumeToText = volume * 100;
    volumeGeneralText.text = volumeToText.ToString("0");
}
```

- **SetUIVolume(float volume):** Estableix el volum del grup "sfx" dintre del AudioMixer en el valor donat.

```
public void SetUIVolume(float volume)
{
    UIVolume = volume;
    myMixer.SetFloat("sfx", Mathf.Log10(volume) * 20);
    float volumeToText = volume * 100;
    volumeUIText.text = volumeToText.ToString("0");
}
```

- **SetMusicVolume(float volume):** Estableix el volum del grup “music” dintre del AudioMixer en el valor donat.

```
public void SetMusicVolume(float volume)
{
    MusicVolume = volume;
    myMixer.SetFloat("music", Mathf.Log10(volume)*20);
    float volumeToText = volume * 100;
    volumeMusicText.text = volumeToText.ToString("0");
}
```

- **SoundGameApply():** Guarda els valors de so dintre del PlayerPrefs.

```
public void SoundGameApply()
{
    PlayerPrefs.SetFloat("masterVolume", generalVolume);
    PlayerPrefs.SetFloat("uiVolume", UIVolume);
    PlayerPrefs.SetFloat("musicVolume", MusicVolume);
}
```

- **SetControllerSen(float sensitivity):** Donant un valor estableix la sensibilitat de la càmera del jugador (apart de canviar el text amb el valor establert).

```
public void SetControllerSen(float sensitivity)
{
    mainControllerSens = Mathf.RoundToInt(sensitivity);
    SensText.text = sensitivity.ToString("0");

}
```

- **GameplayApply():** Guarda el valor de la sensibilitat de la càmera i el booleà de si s’ha invertir la Y de la càmera dintre del PlayerPrefs.

```
public void GameplayApply()
{
    if(InvertYToggle.isOn)
    {
        PlayerPrefs.SetInt("masterInvertY", 1);
    }
    else
    {
        PlayerPrefs.SetInt("masterInvertY", 0);
    }

    PlayerPrefs.SetFloat("masterSen", mainControllerSens);
}
```

- **SetFullScreen(bool isFullscreen):** Aquesta funció guarda el valor booleà que el jugador ha establert per decidir si es vol pantalla completa o no dintre de _isFullScreen.

```
public void SetFullScreen(bool isFullscreen)
{
    _isFullScreen = isFullscreen;
}
```

- **SetQuality(int qualityIndex):** Donat el index seleccionat del dropdown dels gràfics, aquesta funció el guarda en l'atribut _qualitylevel.

```
public void SetQuality(int qualityIndex)
{
    _qualitylevel = qualityIndex;
}
```

- **GraphicsApply():** Guarda els atributs _qualitylevel i _isFullScreen dintre del PlayerPrefs.

```
public void GraphicsApply()
{

    PlayerPrefs.SetInt("masterQuality",_qualitylevel);
    QualitySettings.SetQualityLevel(_qualitylevel);

    PlayerPrefs.SetInt("masterFullscreen", (_isFullScreen ? 1 : 0));
    Screen.fullScreen = _isFullScreen;
}
```

- **ResetButton (string MenuType):** Segons el tipus de setting en el que el jugador estigui (graphics, volume o gameplay), es restableixen els valors modificats pel jugador als valors predeterminats inicialment. Acte seguit es crida la funció corresponent al tipus de setting.

```
    public void ResetButton(string MenuType)
    {
        if(MenuType == "Graphics")
        {
            qualityDropdown.value = 3;
            QualitySettings.SetQualityLevel(3);

            fullScreenToggle.isOn = true;
            Screen.fullScreen = true;

            Resolution currentResolution = Screen.currentResolution;
            Screen.SetResolution(currentResolution.width,
currentResolution.height, Screen.fullScreen);
            resolutionDropdown.value = resolutions.Length;
            GraphicsApply();
        }

        if (MenuType == "Audio")
        {
            myMixer.SetFloat("master", Mathf.Log10(defaultGeneralVolume/100) *
20);
            VolumeGeneralSlider.value = defaultGeneralVolume;
            volumeGeneralText.text = defaultGeneralVolume.ToString("0");

            myMixer.SetFloat("sfx", Mathf.Log10(defaultUIVolume/100) * 20);
            VolumeUISlider.value = defaultUIVolume;
            volumeUIText.text = defaultUIVolume.ToString("0");

            myMixer.SetFloat("music", Mathf.Log10(defaultMusicVolume/100) *
20);
            VolumeMusicSlider.value = defaultMusicVolume;
            volumeMusicText.text = defaultMusicVolume.ToString("0");

            SoundGameApply();
        }
        if (MenuType == "Gameplay")
        {
            SensText.text = defaultSens.ToString("0");
            ControllerSensivilitySlider.value = defaultSens;
            mainControllerSens = defaultSens;
            InvertYToggle.isOn = false;
            GameplayApply();
        }
    }
```

- **LoadScene(string Scene):** carrega l'escena que tingui el mateix nom que l'atribut donat

```
    public void LoadScene(string Scene)
    {
        SceneManager.LoadScene(Scene);
    }
```

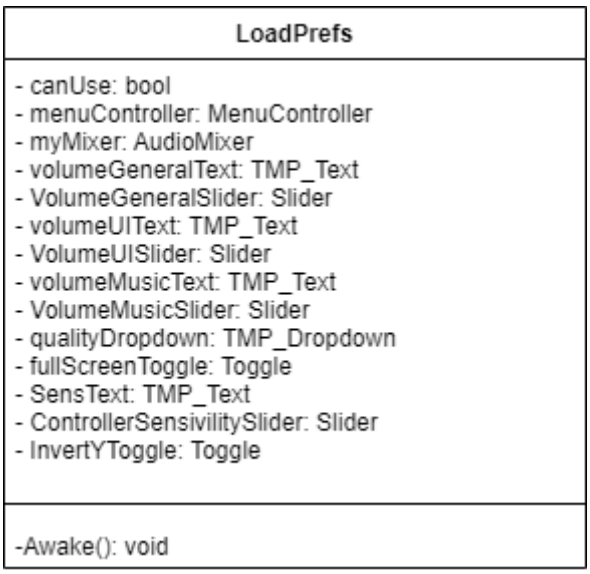
- **SelectObject(GameObject menu):** Selecciona el botó donat per a que el jugador es pugui moure fent ús dels controls de comandament de consola.

```
    public void SelectObject(GameObject menu)
    {
        EventSystem.current.SetSelectedGameObject(menu);
    }
```

6.1.3 LoadPrefs

La classe LoadPrefs és l'encarregada de carregar la configuració del menú de Settings que el jugador ha establert una vegada s'encén el joc. Això permet al jugador no haver-hi d'ajustar el volum, els gràfics o la jugabilitat cada cop que inici el videojoc.

Model de la classe LoadPrefs:



Atributs de la classe LoadPrefs:

- **canUse:** Booleà que serveix per detectar si es carreguen els valors guardats a PlayerPrefab al videojoc o no.

- **menuController:** Referència al script MenuController que utilitzem en el menú principal per poder-hi cridar vàries funcions des de la classe LoadPrefs.

- **myMixer:** El component AudioMixer que utilitzem per administrar els diferents efectes de so.

- **volumeGeneralText:** Text que indica el nivell del volum general del menú de Settings. Va des del 0 al 100.

- **VolumeGeneralSlider:** Correspon al Slider del volum general en el menú de Settings. És el que el jugador utilitzarà per ajustar el volum del so.

- **volumeUIText:** Text que indica el nivell del volum dels efectes de so del joc (quan es clica un botó, al utilitzar la mecànica de gronxar-se o de enganxar-se) del menú de Settings. Va des del 0 al 100.

- **VolumeUISlider:** Correspon al Slider del volum dels efectes de so del joc. És el que el jugador utilitzarà per ajustar el volum del so.

- **volumeMusicText:** Text que indica el nivell del volum de la música dels nivells. Va des del 0 al 100.

- **VolumeMusicSlider:** Correspon al Slider del volum de la música dels nivells. És el que el jugador utilitzarà per ajustar el volum del so.

- **qualityDropdown:** Component tipus Dropdown on hi estan guardats els diferents nivells de qualitat (low, medium, high, ultra).

- **fullScreenToggle:** Component Toggle que serveix si el jugador vol el videojoc en pantalla completa o en mode finestra.

- **SensText:** Text que indica el nivell de la sensibilitat del moviment de la càmera del jugador. Va des del 1 al 10.

- **ControllerSensivilitySlider:** Correspon al Slider de la sensibilitat de la càmera. És el que el jugador utilitzarà per ajustar la sensibilitat.

- **InvertYToggle:** Component Toggle que serveix per indicar si el jugador vol el moviment de la càmera vertical invertit o no.

En la Figura 48 podem veure les propietats públiques assignades al componen de la classe LoadPrefs.

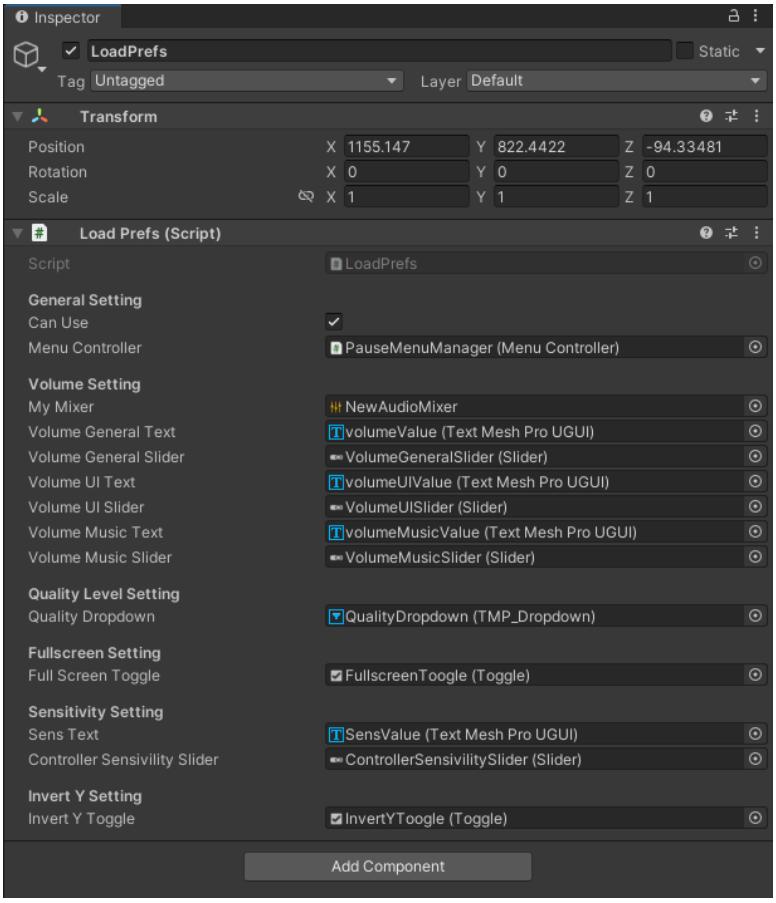


Figura 48: Propietats del component de la classe LoadPrefs

Mètodes de LoadPrefs:

- **Awake():** Funció que s'executa un cop al carregar la instància on es troba. En el nostre cas, el que fa és agafar les variables dels Settings guardades en PlayerPrefs (masterVolume, uiVolume, musicVolume, masterQuality, masterFullscreen, masterSen i masterInvertY) i les carrega en els atributs dels Settings que hem agafat anteriorment per així que el jugador no hagi de modificar constantment les opcions cada cop que inicia el videojoc. En cas de no hi haver cap atribut en el PlayerPrefs, el script MenuController agafarà els valors per defecte (comentats en la classe anterior)

```
private void Awake()
    {
        if (canUse)
        {
            if (PlayerPrefs.HasKey("masterVolume"))
            {
                float localGeneralVolume = PlayerPrefs.GetFloat("masterVolume");
                myMixer.SetFloat("master", Mathf.Log10(localGeneralVolume) * 20);
                float volumeGeneralToText = localGeneralVolume * 100;
                volumeGeneralText.text = volumeGeneralToText.ToString("0");
                VolumeGeneralSlider.value = localGeneralVolume;

                if (PlayerPrefs.HasKey("uiVolume")){
                    float localUIVolume = PlayerPrefs.GetFloat("uiVolume");
                    myMixer.SetFloat("sfx", Mathf.Log10(localUIVolume) * 20);
                    float volumeUIToText = localUIVolume * 100;
                    volumeUIText.text = volumeUIToText.ToString("0");
                    VolumeUISlider.value = localUIVolume;
                }


                if (PlayerPrefs.HasKey("musicVolume")){
                    float localMusicVolume = PlayerPrefs.GetFloat("musicVolume");
                    myMixer.SetFloat("music", Mathf.Log10(localMusicVolume) * 20);
                    float volumeMusicToText = localMusicVolume * 100;
                    volumeMusicText.text = volumeMusicToText.ToString("0");
                    VolumeMusicSlider.value = localMusicVolume;
                }


            }
            else
            {
                menuController.ResetButton("Audio");
            }

            if (PlayerPrefs.HasKey("masterQuality"))
            {
                int localQuality = PlayerPrefs.GetInt("masterQuality");
                qualityDropdown.value = localQuality;
                QualitySettings.SetQualityLevel(localQuality);
            }
            if (PlayerPrefs.HasKey("masterFullscreen"))
            {
                int localFullscreen = PlayerPrefs.GetInt("masterFullscreen");
                if (localFullscreen == 1)
                {
                    Screen.fullScreen = true;
                    fullScreenToggle.isOn = true;
                }
                else
                {
                    Screen.fullScreen = false;
                    fullScreenToggle.isOn = false;
                }
```

```
        }
        if (PlayerPrefs.HasKey("masterSen"))
        {
            float localSensitivity = PlayerPrefs.GetFloat("masterSen");

            ControllerSensivilitySlider.value = localSensitivity;
            menuController.mainControllerSens = Mathf.RoundToInt(localSensitivity);
        }

        if (PlayerPrefs.HasKey("masterInvertY"))
        {
            if (PlayerPrefs.GetInt("masterInvertY") == 1)
            {
                InvertYToggle.isOn = true;
            }
            else
            {
                InvertYToggle.isOn = false;
            }
        }

    }
}
```

6.2 Implementació de les interfícies

En el motor Unity, totes les interfícies d'usuari venen definides dins d'un component anomenat Canvas, on hi col·loquem tots els elements que vulguem mostrar per pantalla. En el videojoc hem creat una sèrie de canvas diferents que corresponen a diferents interfícies.

6.2.1 Menú principal

Aquest Canvas correspon els elements que s'hi mostren en el menú principal. Com podem observar en la Figura 49, dintre d'aquests ens trobem amb varis Canvas que hi corresponen a diferent elements del menú principal i del menú d'opcions. Apart també hi podem trobar un Video Player que es el que s'encarrega de mostrar el vídeo de fons i el Text que hi correspon al títol del videojoc.

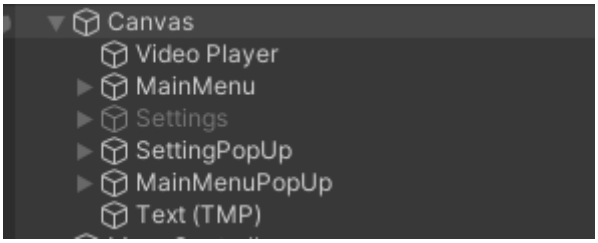


Figura 49: Elements que es troben en el contenidor Canvas.

Tota aquesta interfície està gestionada per l'objecte MenuController comentat anteriorment (veure Apartat 6.1.2), per tant no veiem necessari explicar-la també en aquest apartat.

MainMenu:

Aquest Canvas està compost per nombrosos elements que hi corresponen als botons que el jugador veu res més iniciar el videojoc (veure Figura 50). Cadascun d'aquests botons està compost pels mateixos components: al prémer el botó, activa el so de confirmació i torna visible una de les altres interfícies del component Canvas principal (veure Figura 51).

Figura 50: Botons que es troben dintre del Canvas MainMenu.

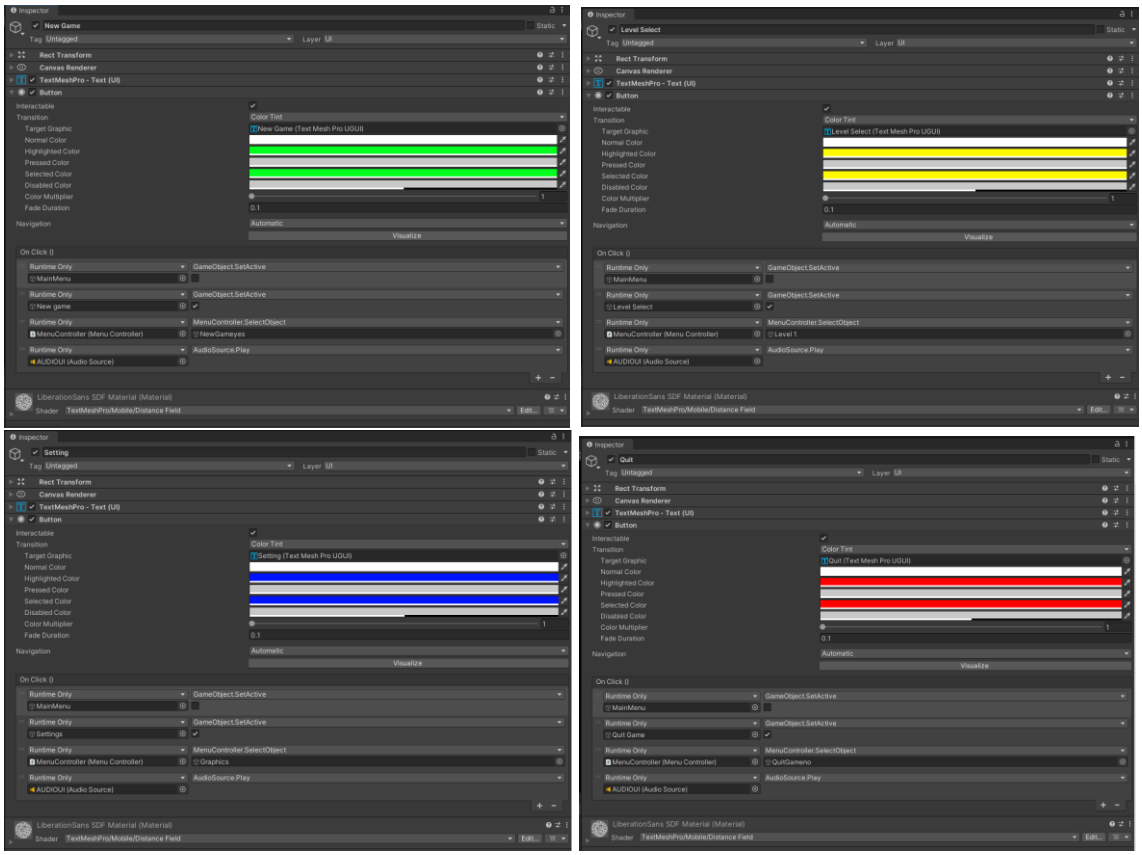


Figura 51: Els components que hi tenen els botons del Menú principal.

Settings:

Semblant al Canvas MainMenu, aquest correspon al menú de configuració del videojoc, on el jugador pot modificar diferents aspectes del videojoc com el volum, els gràfics o diferents elements de la jugabilitat (veure Figura 52).

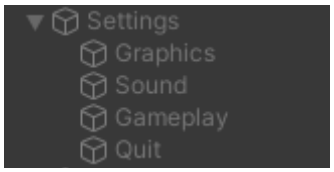


Figura 52: Botons que es troben dintre del canvas Settings. Per defecte està ocult fins que el jugador prem el botó de Settings en el canvas MainMenu.

De la mateixa manera que en el MainMenu, els botons tenes els mateixos components, només canviant les crides de funcions (veure Figura 53).

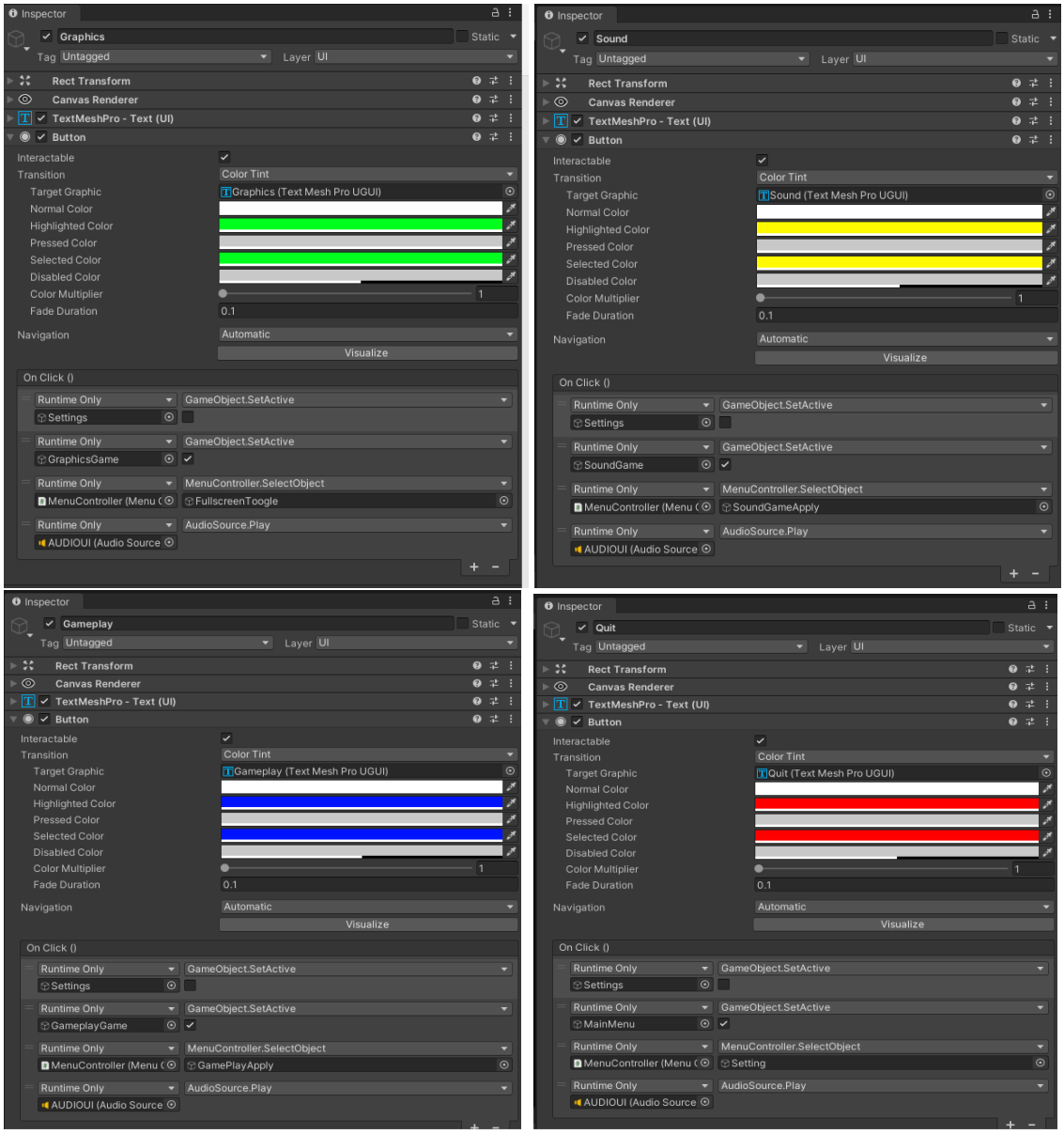


Figura 53: Els components que tenen els botons del Setting.

SettingPopUp

Dintre d'aquest Canvas hi podem trobar uns altres 4 que hi corresponen a les finestres que hi apareixen segons el tipus de botó que es seleccioni en el Canvas de Settings (veure Figura 54). Aquestes finestres són les que el jugador utilitzarà per canviar diferents aspectes del joc.

Figura 54: Les diferents finestres (i, per conseqüència, també Canvas) que ens podem trobar dintre del Canvas SettingPopUp.

- **SoundGame:** Finestra on el jugador modifica el so del videojoc. Si en fixem en la Figura 55, conté 3 Sliders per canviar els diferents tipus de so que hi ha i 3 botons: el botó SoundGameApply guarda els canvis efectuats als sliders, el SoundGameReturn torna a la finestra anterior i el SoundGameReset torna els valors dels Sliders per defecte. (veure Figura 56)

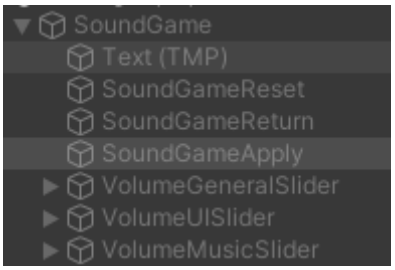


Figura 55: Els elements que conté el Canvas SoundGame.

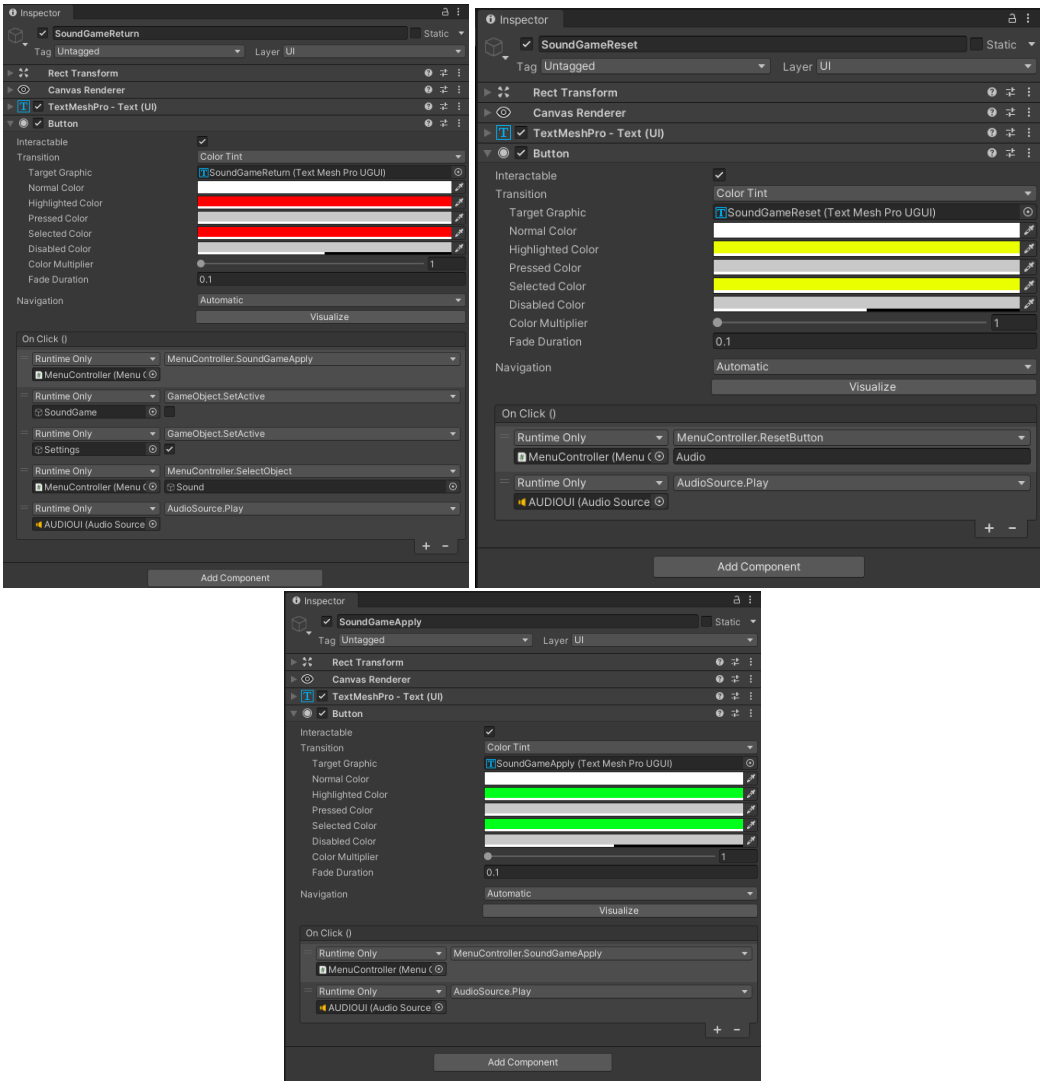


Figura 56: Components que formen els botons de SoundGame.

- **GameplayGame:** Finestra on el jugador pot canviar diferents aspectes de la jugabilitat. Aquest Canvas està compost per un Toggle on pot invertir la càmera i un Slider per modificar la sensibilitat de la càmera (veure Figura 57). Apart dels 3 botons que fan la mateixa funció que els botons de SoundGame amb la inclusió un botó addicional que correspon a mostrar els controls del videojoc (veure Figura 58).

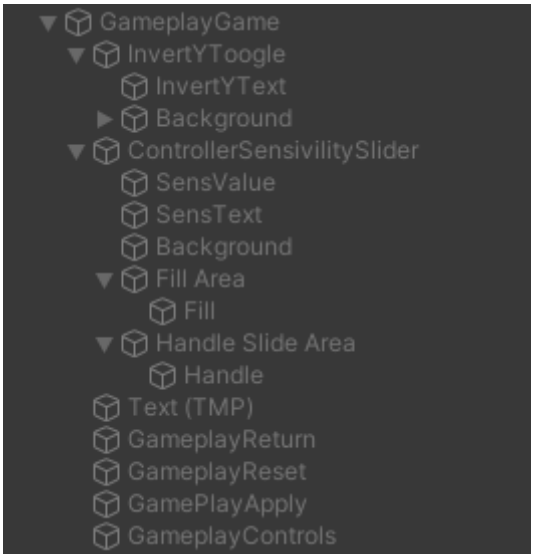


Figura 57: Elements que conté el Canvas GameplayGame

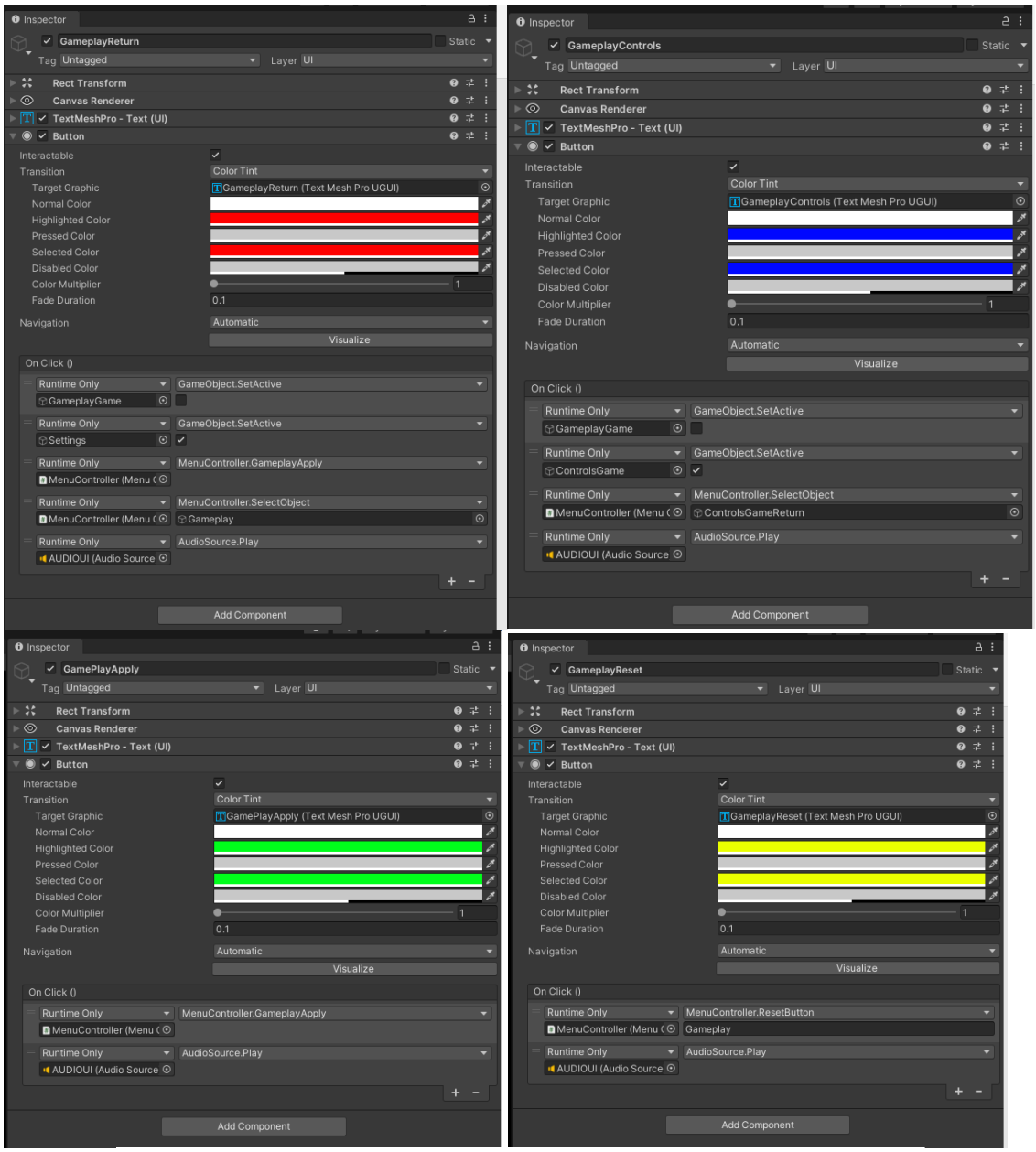


Figura 58: Components que formen els botons de GameplayGame.

- **GraphicsGame:** Aquesta finestra permet al jugador modificar les opcions gràfiques del videojoc. El Canvas està compost per un Toggle que determina si el videojoc està en pantalla completa o no i dos Dropdowns que corresponen a les diferents resolucions de la pantalla i els diferents nivells de gràfics del videjoc (veure Figura 59). També té els tres botons idèntics als altres Canvas de SettingPopUp (veure Figura 60).

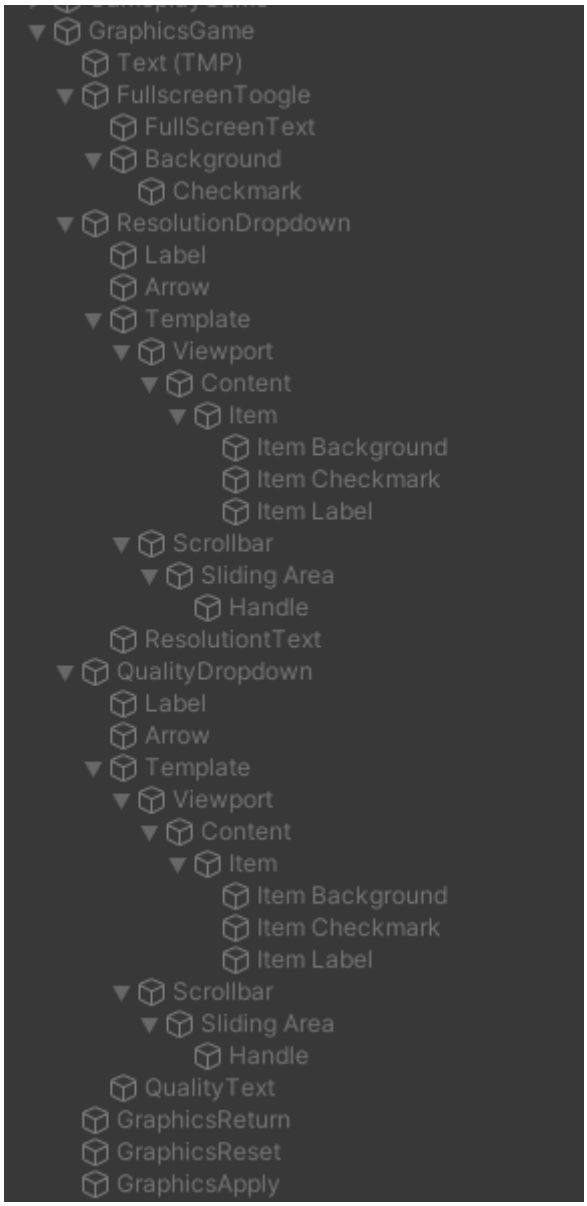


Figura 59: Elements que conté el Canvas GraphicsGame.

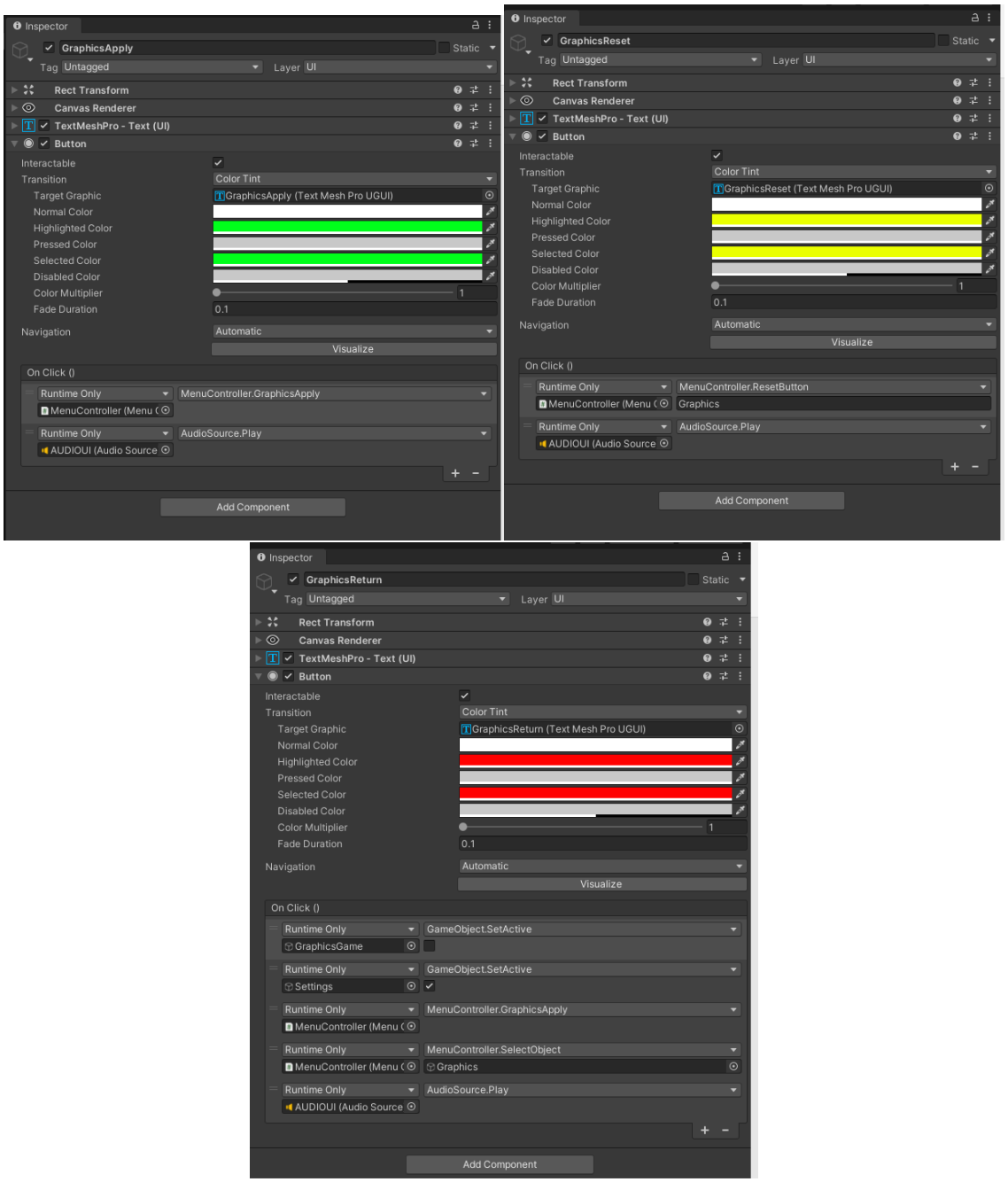


Figura 60: Components que formen els botons de GraphicsGame.

- **ControlsGame:** Aquesta finestra només es compon de dos objectes, una imatge amb els controls del joc i un botó per retornar a la finestra anterior (veure Figura 61).

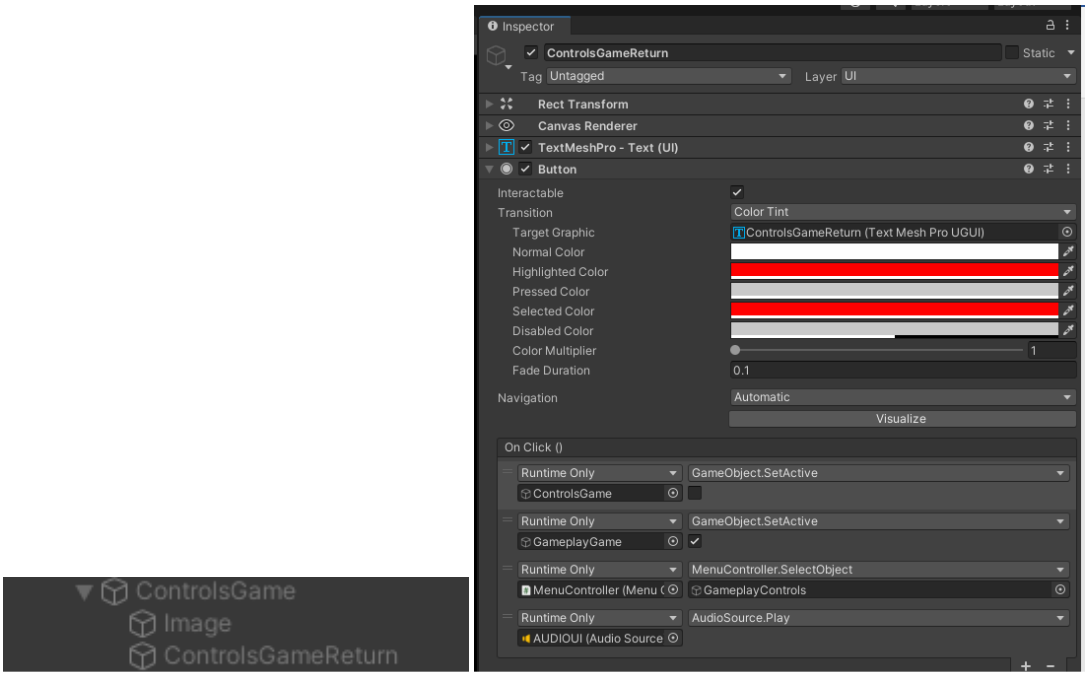


Figura 61: Elements que formen el Canvas ControlsGame, juntament amb els components que hi formen el botó de ControlsGameReturn.

MainMenuPopUp:

Aquest Canvas correspon a les finestres que hi surten un cop es selecciona una de les opcions del menú principal que no sigui el botó Settings (veure Figura 62).

Figura 62: Diferents elements que formen el canvas MainMenuPopUp.

- **New game:** Correspon a una petita finestra que confirma la selecció de començar una partida nova. Apart del text de confirmació hi ha dos botons que el jugador pot interactuar (veure Figura 63): el NewGameyes farà que el jugador sigui transportat al primer nivell mentre que el NewGameno traurà aquesta finestra per mostrar un altre cop el Canvas de MainMenu (veure Figura 64).

Figura 63: Diferents elements que hi formen el Canvas New game.

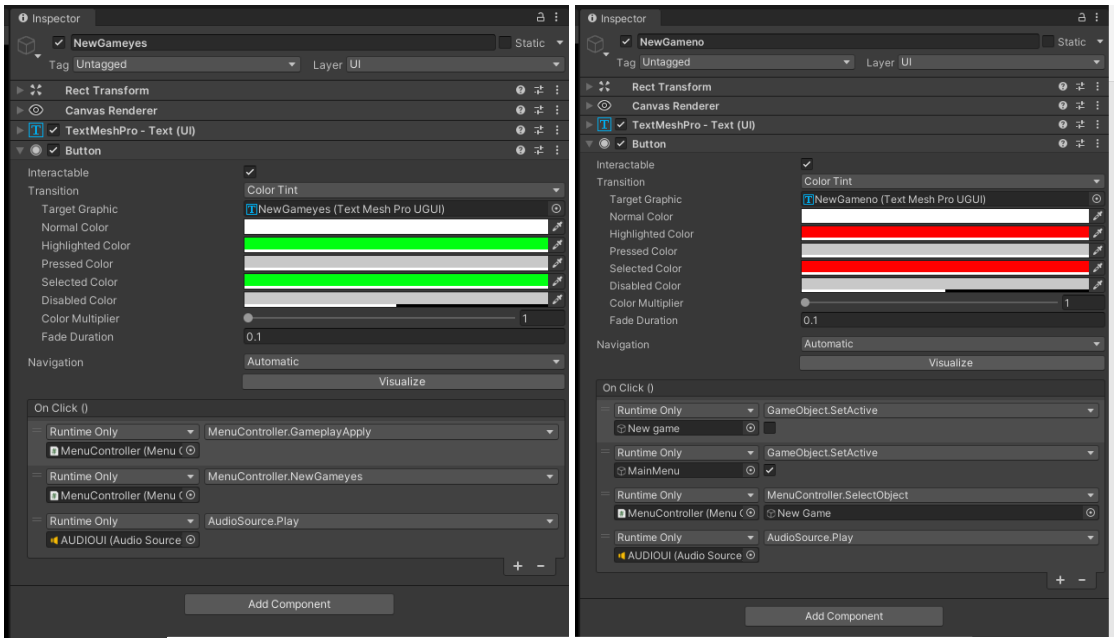


Figura 64: Components que formen els botons de New game.

- **Level Select:** Finestra on el jugador podrà seleccionar manualment quin nivell vol jugar sense necessitat d'haver de començar una partida nova (veure Figura 65). Apart del text tota la resta son botons, encara que només dos d'ells són funcionals, ja que són els nivells disponibles de moment. Per últim el botó de retorn serveix per tornar un altre cop al Canvas del menú principal (veure Figura 66).

Figura 65: Elements dintre del Canvas Level Select.

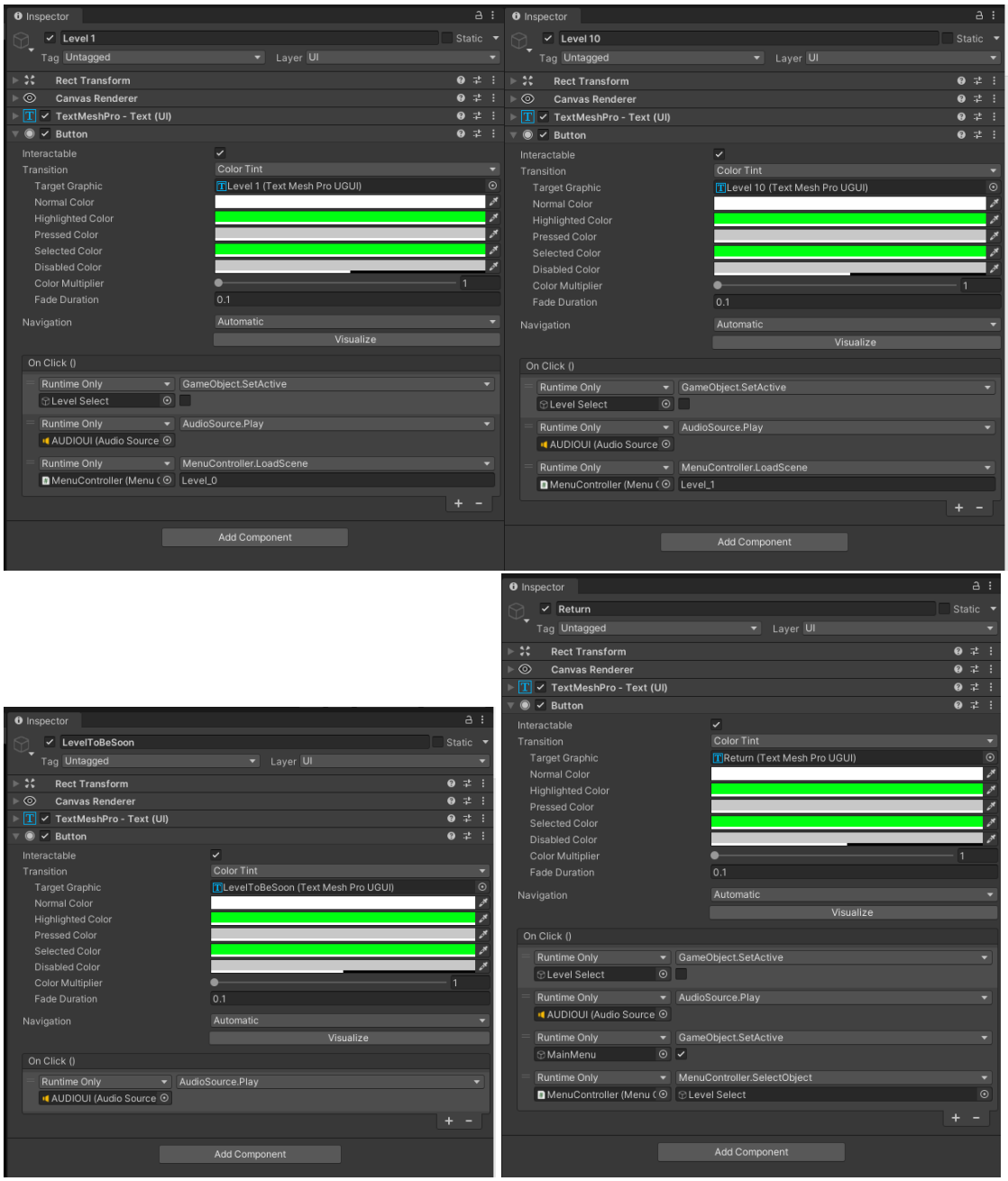


Figura 66: Components dintre del botons de Level Select. Inclòs un dels botons buits (la resta són copies).

- **Quit Game:** Finestra semblant a la de New game. En aquesta, pregunta al jugador si vol sortir del joc amb dos botons (veure Figura 67): QuitGameyes per sortir a l'escriptori del ordinador i QuitGameno per anar al Canvas del menú principal (veure Figura 68).

Figura 67: Elements dintre del Canvas Quit Game.

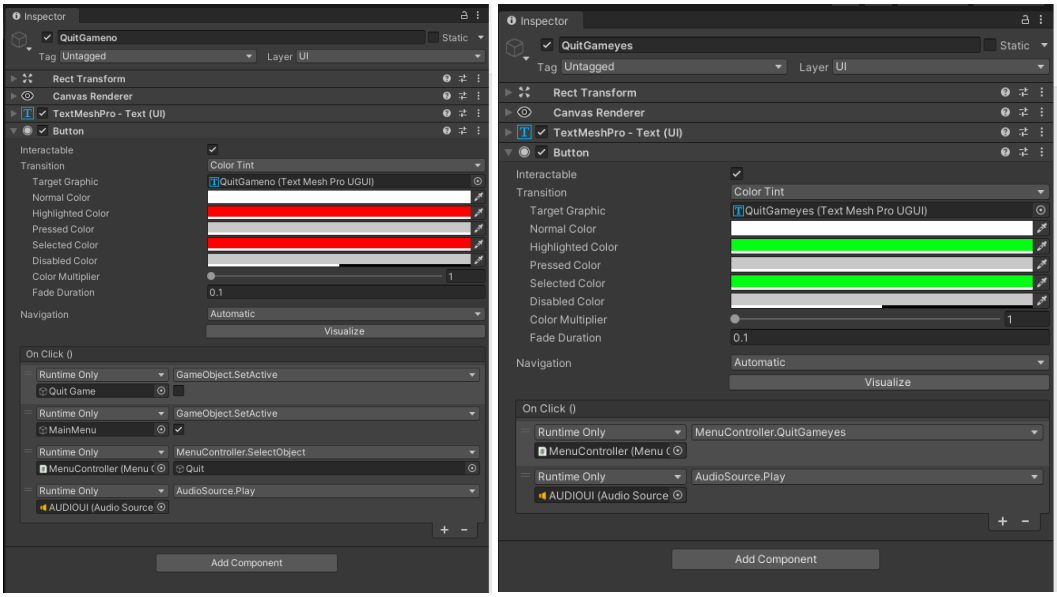


Figura 68:Components dintre dels botons de Quit Game.

6.2.2 Pause Menu

Aquests és el Canvas que hi ha inclòs en tots els nivells jugables pel jugador (sense comptar el menú principal i el final). Hi conté tres botons els quals el jugador pot interactuar amb el ratolí o el comandament de la consola (veure Figura 69). El botó de Setting realitza la mateixa funció que el botó amb el mateix nom en el Menú principal, mentre que el botó de Resume permet tornar al joc i el botó de quit executa una finestra addicional per assegurar que el jugador vol sortir del nivell (veure Figura 70). Aquest Canvas hi té adjunt una classe que és la que porta tot el funcionament de Pausa del videojoc.

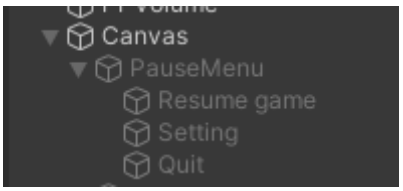


Figura 69: Elements dintre del Canvas PauseMenu

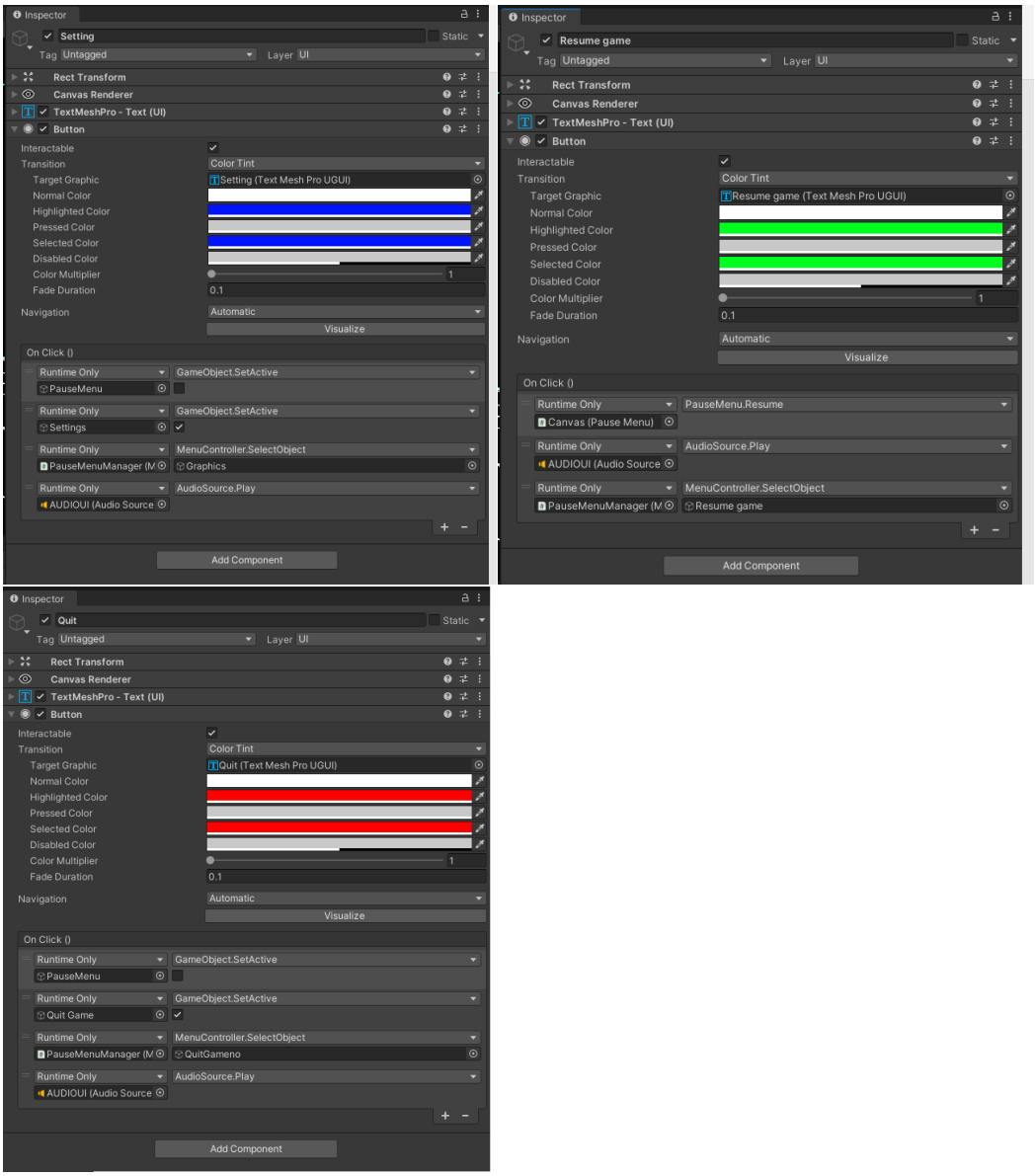
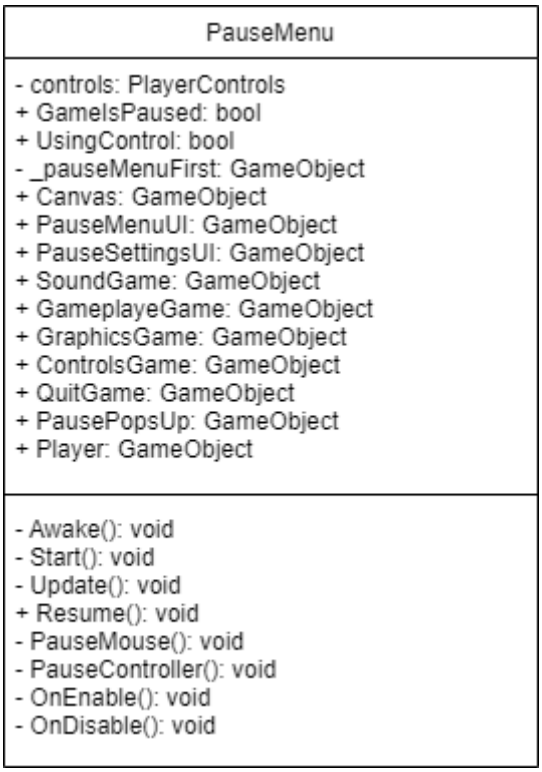


Figura 70: Components dintre dels botons de PauseMenu.

Model de la classe PauseMenu:



Atributs de PauseMenu:

- **controls:** Atribut que correspon al moviment del jugador. L'utilitzem per a que el jugador no ho pugui interactuar amb el videojoc quan estigui pausat.

+ **GameIsPaused:** Booleà que indica si el joc està pausat o no.

+ **UsingControl:** Atribut que utilitzarem per saber si el jugador utilitza un comandament de consola o no.

- **_pauseMenuFirst:** GameObject que serà seleccionat un cop es premi el botó de pausa al jugar amb comandament.

+ **Canvas:** Atribut on hi tenim el canvas de tot el menú de pausa.

+ **PauseMenuUI:** GameObject on hi trobem el menú de pausa principal.

+ **PauseSettingsUI:** GameObject on hi trobem el menú de Settings en menú de pausa (idèntic al del menú principal).

+ **SoundGame:** GameObject on hi trobem el menú de volum en el menú de pausa.

+ **GameplayGame:** GameObject on hi trobem el menú de jugabilitat en el menú de pausa.

+ **GraphicsGame:** GameObject on hi trobem el menú de gràfics en el menú de pausa.

+ **ControlsGame:** GameObject on hi trobem el menú de controls en el menú de pausa.

+ **QuitGame:** GameObject on hi trobem el menú de sortir del joc del menú de pausa.

+ **PausePopsUp:** GameObject on hi trobem les pestanyes que trobem un cop premem el botó de Settings o de sortir del joc (idèntic als dels menú principal).

+ **Player:** Atribut on hi guardem el GameObject del jugador que hi ha en el nivell.

En la Figura 71 podem veure les propietats públiques assignades al component de la classe PauseMenu.

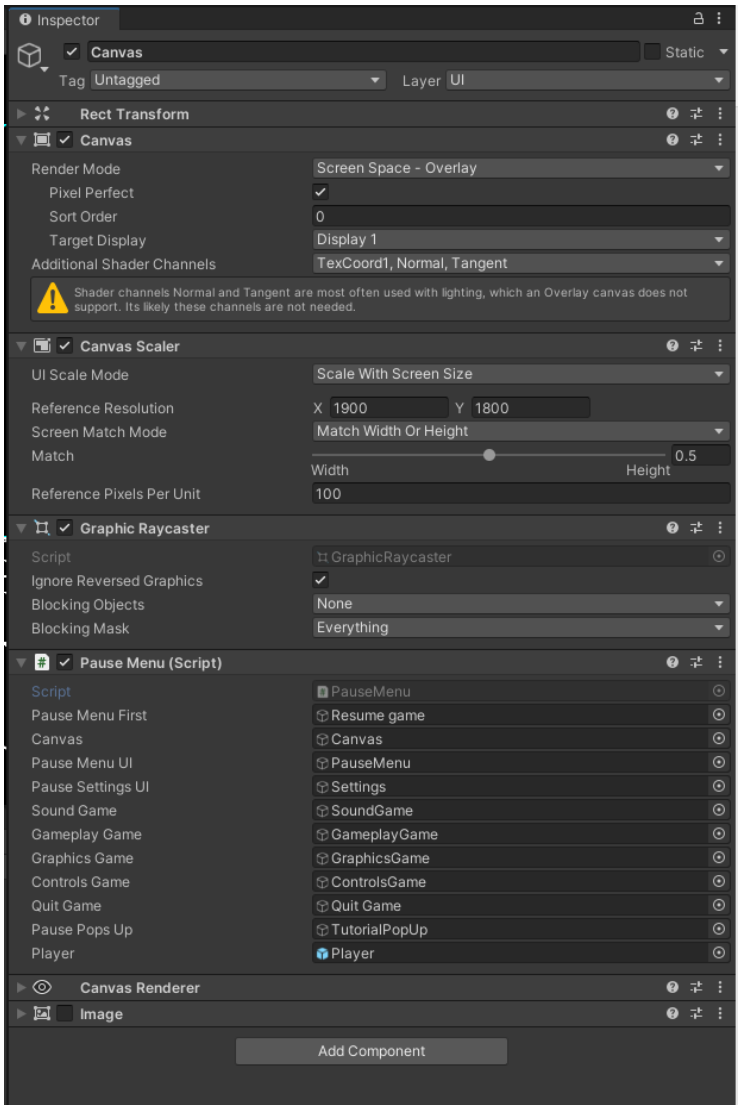


Figura 71: Propietats del component de la classe PauseMenu.

Métodes de PauseMenu

- **Awake():** Funció que s'executa un cop al iniciar l'escena. Simplement omple la variable controls.

```
    private void Awake()
    {
        controls = new PlayerControls();
    }
```

- **Start():** Funció d'inicialització de la classe. Desactivem tots els Canvas per a que no es mostrin al principi de l'escena. Apart d'inicialitzar les variables.

```
    private void Start()
    {
        UsingControl = false;
        Canvas.GetComponent<Image>().enabled = false;
        PauseMenuUI.SetActive(false);
        PauseSettingsUI.SetActive(false);
        SoundGame.SetActive(false);
        GameplayGame.SetActive(false);
        GraphicsGame.SetActive(false);
        ControlsGame.SetActive(false);
        QuitGame.SetActive(false);
    }
```

- **Update():** Funció que es crida en cada frame. Detecta si el jugador prem el botó de pausa del teclat o del comandament de consola en cas de que utilitzi un. En cas positiu (i el jugador no ha arribat al final del nivell), depenent si el joc ja hi estava pausat amb anterioritat o no, cridarà la funció de Resume() o bé la funció de pausa segons quin botó hagi premut.

```
    void Update()
    {
        if (Input.GetKeyDown(KeyCode.Escape) && !NextOrRepeatLevel.gameIsFinished)
        {
            if (GameIsPaused)
            {
                Resume();
            }
            else
            {
                PauseMouse();
            }
        }

        if (controls.Gameplay.Pause.WasPressedThisFrame() &&
!NextOrRepeatLevel.gameIsFinished)
        {
            if (GameIsPaused)
            {
                Resume();
                EventSystem.current.SetSelectedGameObject(null);
            }
            else
            {
                PauseController();

            }
        }
    }
```

- **Resume():** Funció que tanca el menú de pausa i torna a la partida.

```
public void Resume()
{
    Canvas.GetComponent<Image>().enabled = false;
    PauseMenuUI.SetActive(false);
    PauseSettingsUI.SetActive(false);
    SoundGame.SetActive(false);
    GameplayGame.SetActive(false);
    GraphicsGame.SetActive(false);
    ControlsGame.SetActive(false);
    PausePopsUp.SetActive(true);
    QuitGame.SetActive(false);
    Time.timeScale = 1f;
    GameIsPaused = false;
    EventSystem.current.SetSelectedGameObject(null);
    Player.GetComponent<PlayerMovementAdvanced>().enabled = true;
}
```

- **PauseMouse():** Funció que permet al jugador congelar el joc (i els controls del jugador) i mostrar el menú de pausa un cop es prem el botó de pausa del teclat.

```
void PauseMouse()
{
    Canvas.GetComponent<Image>().enabled = true;
    PauseMenuUI.SetActive(true);
    PausePopsUp.SetActive(false);
    Time.timeScale = 0f;
    UsingControl = false;
    GameIsPaused = true;
    Player.GetComponent<PlayerMovementAdvanced>().enabled = false;
}
```

- **PauseController():** Funció que permet al jugador congelar el joc i mostrar el menú de pausa un cop es prem el botó de pausa del comandament de consola. A més s'amaga el cursor i es selecciona la primera opció del menú per facilitar la visibilitat.

```
void PauseController()
{
    Canvas.GetComponent<Image>().enabled = true;
    PauseMenuUI.SetActive(true);
    PausePopsUp.SetActive(false);
    Time.timeScale = 0f;
    GameIsPaused = true;
    Cursor.lockState = CursorLockMode.Locked;
    Cursor.visible = false;
    UsingControl = true;
    EventSystem.current.SetSelectedGameObject(_pauseMenuFirst);
    Player.GetComponent<PlayerMovementAdvanced>().enabled = false;
}
```

- **OnEnable():** Funció creada pel comandament de consola. Permet activar els controls de moviments del comandament.

```
private void OnEnable()
{
    controls.Gameplay.Enable();
}
```

- **OnDisable():** Funció creada pel comandament de consola. Permet desactivar els controls de moviments del comandament.

```
private void OnDisable()
{
    controls.Gameplay.Disable();
}
```

6.2.3 TutorialPopUp

Durant el transcurs dels nivells del videojoc, hi hauran moments on el joc mostrarà consells o instruccions per ensenyar al jugador sobre les diferent mecàniques del joc. Tots aquest consells s'emmagatzemen dintre del Canvas TutorialPopUp vist en la Figura 72, on només consisteixen en un TextMeshPro amb les instruccions a donar.

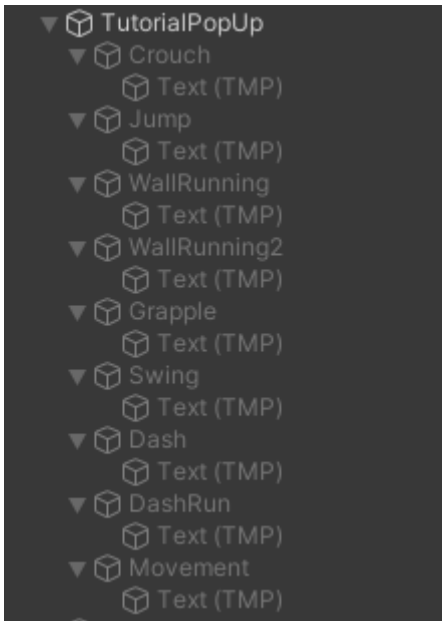


Figura 72: Contingut dintre del Canvas TutorialPopUp.

6.2.4 NextOrRepeatLevel

El Canvas conté el que es mostra un cop s'ha arribat al final del nivell. Consisteix en una petita finestra amb un text i dos botons que et permeten, o bé avançar de nivell, o tornar-lo a jugar (veure Figura 73). En el cas del nivell del tutorial, al prémer el botó de repetir el nivell, sortirà un altre finestra amb un text preguntant si es vol tornar a jugar amb el tutorial activat o no (veure Figura 74) i dos botons per triar la opció que el jugador vulgui.

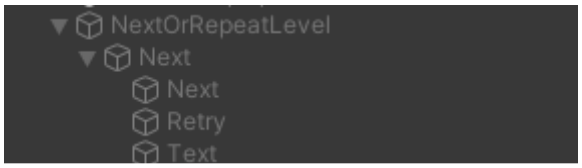


Figura 73: Contingut dintre del Canvas NextOrRepeatLevel.

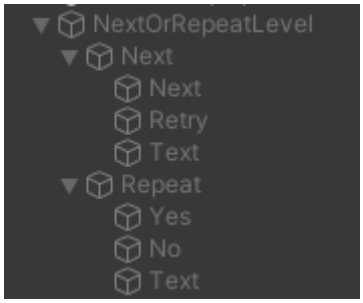


Figura 74: Contingut dintre del Canvas NextOrRepeatLevel en el nivell tutorial.

6.2.5 CoolDownRespawn

Aquest Canvas s'utilitza quan el jugador cau de les plataformes i ha de tornar a començar el nivell. El contingut es molt simple: només dos textos indicant que es prepari i una compte enrere que, al acabar-s'hi, el jugador podrà tornar a moure's (veure Figura 75).

Figura 75: Contingut dintre del Canvas CoolDownRespawn.

6.2.6 EndText

Per acabar, aquest últim Canvas és el que utilitzarem per l'escena final. Està compost per uns quants elements amb un Text que aniran apareixent i desapareixent a mesura que el jugador va llegint (veure Figura 76). Finalment també hi ha inclòs dos botons que el jugador podrà seleccionar y que permetrà tornar a començar o sortir del videojoc (veure Figura 77).

Figura 76: Contingut dintre del Canvas EndText.

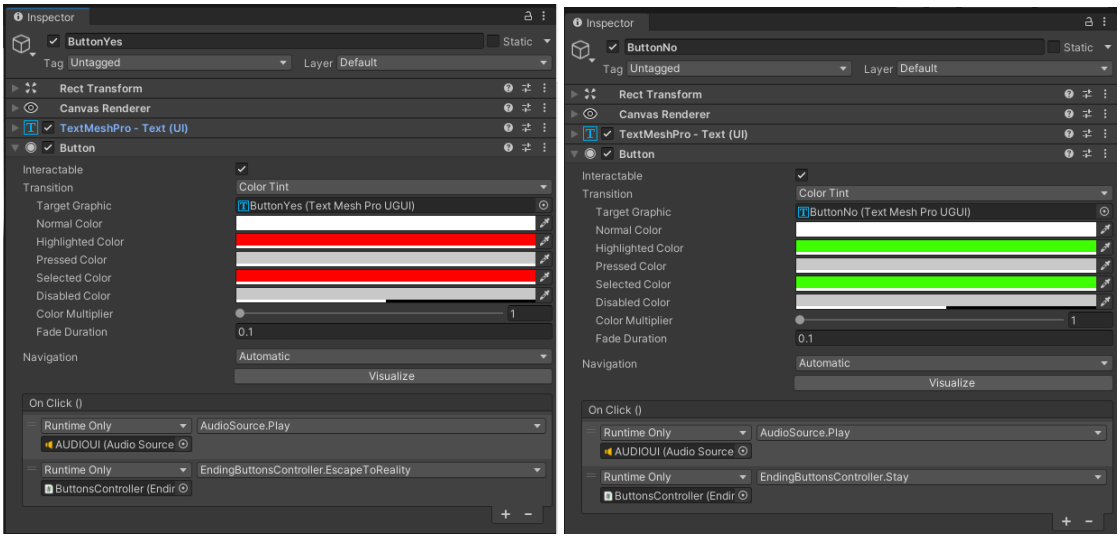


Figura 77: Components que hi tenen els botons dintre del Canvas EndText.

Tots els textos i botons tenen implementats un component de Animation, que és el que permet que hi vagin apareixent i desapareixent. Simplement modifiquen la opacitat del text del màxim al mínim, i viceversa (veure Figura 78).

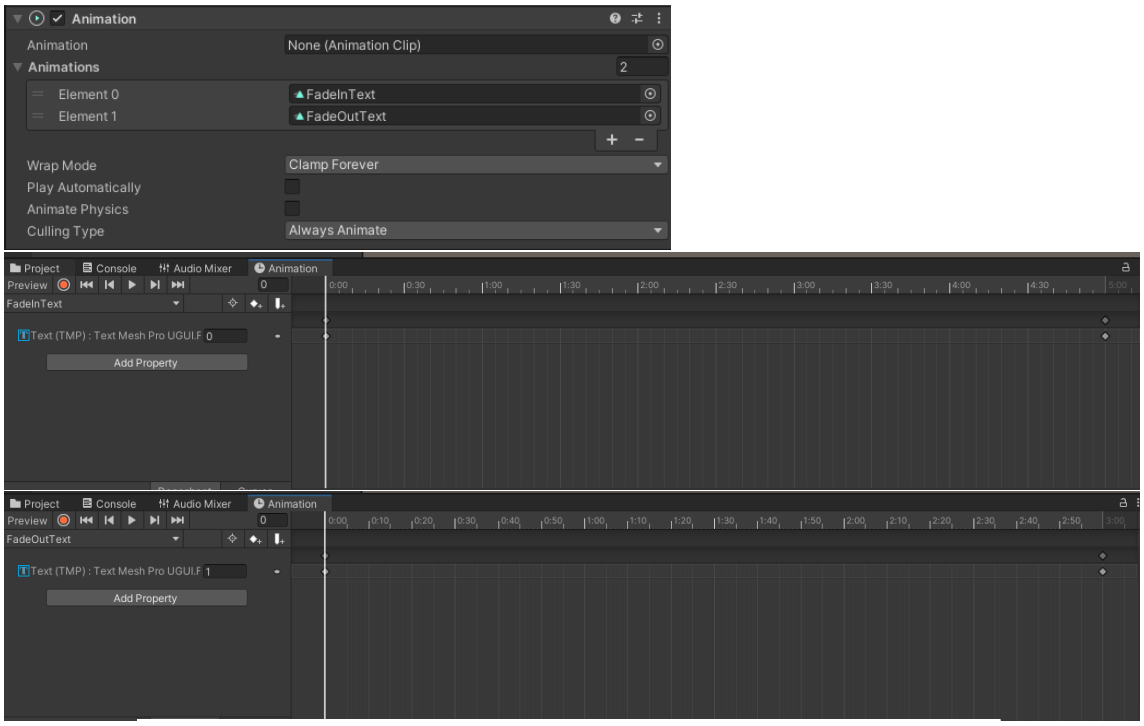
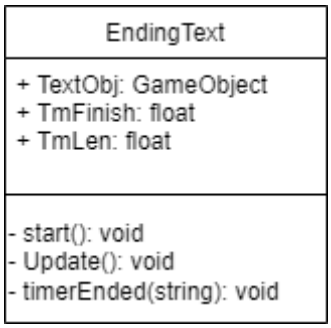


Figura 78: Component Animation que hi tenen tots els textos i botons en el Canvas EndText.

<u>EndingText:</u>

La classe EndingText és l'encarregada de gestionar els textos de l'escena final del joc. La funció es molt simple: La classe té dos comptadors, el primer és el temps que ha d'esperar per inicial l'animació de FadeIn del text. Quan aquesta arriba a zero, s'inicia l'altre comptador, el qual indica el temps que haurà d'esperar la funció per iniciar l'animació de FadeOut del text escollit.

Model de la classe EndingText:



Atributs de la classe EndingText:

+ **TextObj:** Atribut on hi guardem el text que volem utilitzar per a que es mostri o desaparegui.

+ **TmFinish:** Float que utilitzarem per detectar quan ha de desaparèixer el text en pantalla.

+ **TmLen:** Float on guardem el temps que ha d'esperar per a que el text es mostri.

En la Figura 79 podem veure les propietats públiques assignades al component de la classe EndingText. Aquesta classe s'ha utilitzat per a cada text de l'escena final del joc, però només mostrem un ja que seria una mica redundant.

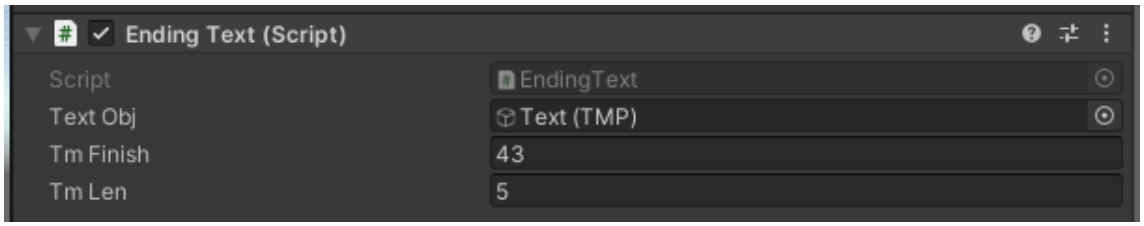


Figura 79: Propietats de la classe EndingText

Mètodes de la classe EndingText:

- **Start():** Funció que s'inicia al primer frame de l'escena. Mostra el cursor i s'assegura d'aturar les animacions del GameObject

```
void Start()
{
        Cursor.lockState = CursorLockMode.None;
        Cursor.visible = true;
        TextObj.GetComponent<Animation>().Stop("FadeInText");
        TextObj.GetComponent<Animation>().Stop("FadeOutText");
}
```

- **Update():** Funció que s'executa a cada frame de l'escena. Posa en funcionament dos comptadors, quan s'acaba el comptador TmLen crida la Funció TimeEnded amb el nom de l'animació d'aparició de text. Quan s'acaba el TmFinish, crida l'animació de desaparició.

```
    void Update()
    {
     if (TmLen > 0.0f)
     {
                TmLen -= Time.deltaTime;
     }
          if(TmFinish > 0.0f)
     {
                TmFinish -= Time.deltaTime;
     }

          if (TmLen<=0.0f && TmFinish > 0.0f)
          {
                timerEnded("FadeInText");
          }
          if(TmFinish<=0.0f)
     {
                TextObj.GetComponent<Animation>().Stop("FadeInText");
                timerEnded("FadeOutText");
          }

    }
```

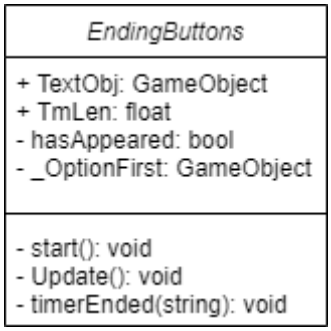
- **TimerEnded(string):** Funció que inicia la animació donada pel string

```
    void timerEnded(string name)
{
          TextObj.GetComponent<Animation>().Play(name);
}
```

EndingButtons:

Aquesta Classe es semblant a la de EndingTexts, però s'encarrega de fer aparèixer i desaparèixer els botons de l'escena final (i també els activa per a que el jugador els pugui utilitzar).

Model de la classe EndingButtons:



Atributs de la classe EndingButtons:

+ **TextObj:** Atribut on hi guardem el text o botó que volem utilitzar per que es mostri o desaparegui.

+ **TmLen:** Float on guardem el temps que ha d'esperar per a que el text o botó es mostri.

- **hasAppeared:** Booleà que indica que el text o botó ha aparegut en l'escena

- **_OptionFirst:** Atribut que selecciona un dels botons. Pensat pels jugadors que utilitzen comandament de consola.

En la Figura 80 podem veure les propietats públiques assignades al component de la classe EndingButtons.

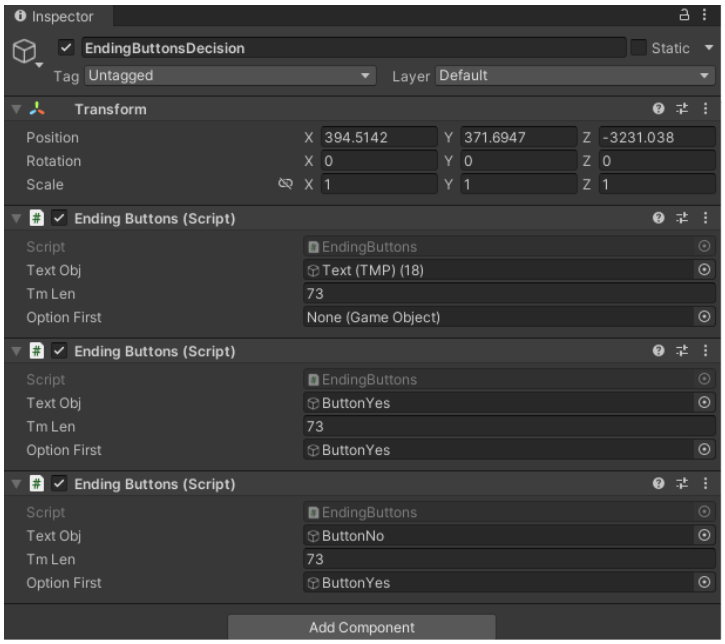


Figura 80: Propietats de la classe EndingButtons en els 3 casos que s'utilitza.

Mètodes de la classe EndingButtons:

- **Start():** Funció que s'inicia al primer frame. Estableix el booleà a la fals, per a que totes les animacions que hi poguessin estar activades al TextObj i desactiva el botó, en cas de que hi hagués un.

```
void Start()
{
	hasAppeared = false;
	TextObj.GetComponent<Animation>().Stop("FadeInText");
	TextObj.GetComponent<Animation>().Stop("FadeOutText");
	if (TextObj.GetComponent<Button>() != null)
	{
		TextObj.GetComponent<Button>().enabled = false;
	}
}
```

- **Update():** Funció que es crida a cada frame de l'escena. A diferència que la classe anterior, aquest només conté un comptador, ja que no es requereix que desaparegui al cap d'un temps. Un cop s'acaba aquest timer, es crida a la funció timerEnded amb el nom de l'animació de desaparició.

```
void Update()
{
	if (TmLen > 0.0f)
	{
		TmLen -= Time.deltaTime;
	}

	if (TmLen <= 0.0f && !hasAppeared)
	{
		timerEnded("FadeInText");
	}

}
```

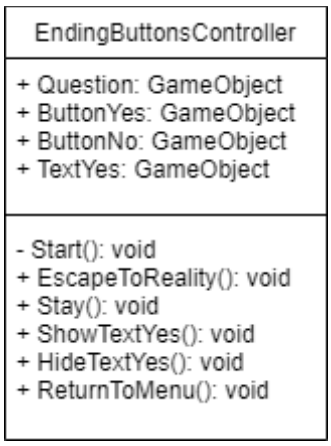
- **timerEnded():** Funció que salta quan el comptador s'acaba. Inicia l'animació donada pel name, activa el botó en cas de que el GameObject assignat ho sigui i estableix el booleà a cert.

```
void timerEnded(string name)
{

	TextObj.GetComponent<Animation>().Play(name);
	if (TextObj.GetComponent<Button>() != null)
	{
		EventSystem.current.SetSelectedGameObject(_OptionFirst);
		TextObj.GetComponent<Button>().enabled = true;
	}
	hasAppeared = true;
}
```

EndingButtonsController:

Per últim, aquesta classe es la que s'encarrega de gestionar el que passa un cop s'ha escollit un dels botons de l'escena final. Si es decideix sortir del joc, aquesta classe s'encarregarà de tornar al jugador a l'escriptori mentre que, si decideix tornar al menú principal, el canviarà d'escena, no sense abans mostrar els últims textos del joc.

Model de la classe EndingButtonsController:



Atributs de la classe EndingButtonsController:

+ **Question:** GameObject que hi conté la pregunta a la qual el jugador ha de respondre amb els botons en l'escena final.

+ **ButtonYes:** Assignem aquest GameObject al botó que diu que vol tornar al menú principal.

+ **ButtonNo:** Fa referència al botó que el jugador pot triar per sortir del joc.

+ **TextYes:** GameObject que conté el text final en cas de que el jugador triï el botó de tornar al menú principal.

En la Figura 81 podem veure les propietats públiques assignades al component de la classe EndingButtonsController.

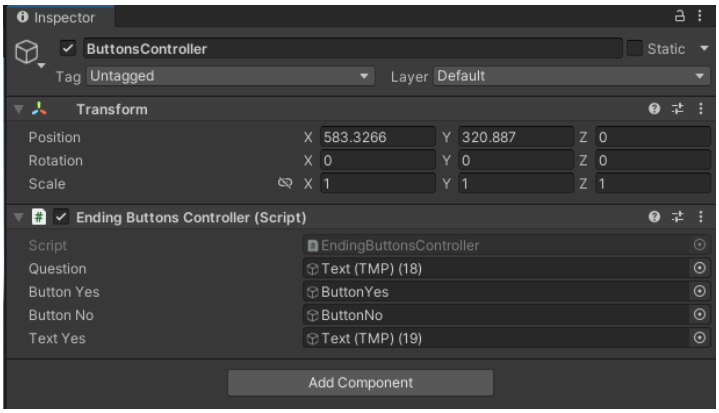


Figura 81: Propietats de la classe EndingButtonsController

Mètodes de la classe EndingButtonsController:

- **Start():** Funció que s'inicia al primer frame. Atura totes les animacions dels GameObject atributs.

```
void Start()
{
    Question.GetComponent<Animation>().Stop("FadeInText");
    Question.GetComponent<Animation>().Stop("FadeOutText");

    ButtonYes.GetComponent<Animation>().Stop("FadeInText");
    ButtonYes.GetComponent<Animation>().Stop("FadeOutText");

    ButtonNo.GetComponent<Animation>().Stop("FadeInText");
    ButtonNo.GetComponent<Animation>().Stop("FadeOutText");

    TextYes.GetComponent<Animation>().Stop("FadeInText");
    TextYes.GetComponent<Animation>().Stop("FadeOutText");
}
```

+ **EscapeToReality():** Funció que treu al jugador del joc i l'envia a l'escriptori.

```
public void EscapeToReality()
{
    Application.Quit();
}
```

+ **Stay():** En cas de que el jugador decideixi anar al menú principal, treu tant el text com els botons (a més de desactivar l'opció de seleccionar-los) i invoca la funció ShowTextYes al cap d'una estona.

```
public void Stay()
{
    Question.GetComponent<Animation>().Stop("FadeInText");
    ButtonYes.GetComponent<Animation>().Stop("FadeInText");
    ButtonNo.GetComponent<Animation>().Stop("FadeInText");
    Question.GetComponent<Animation>().Play("FadeOutText");
    ButtonYes.GetComponent<Animation>().Play("FadeOutText");
    ButtonNo.GetComponent<Animation>().Play("FadeOutText");
    ButtonYes.GetComponent<Button>().enabled = false;
    ButtonNo.GetComponent<Button>().enabled = false;
    Invoke(nameof(ShowTextYes), 3.0f);

}
```

+ **ShowTextYes():** Funció que mostra l'atribut TextYes durant uns segons. Acte seguit crida a la funció HideTextYes.

```
public void ShowTextYes()
{
    TextYes.GetComponent<Animation>().Play("FadeInText");
    Invoke(nameof(HideTextYes), 5.0f);
}
```

+ **HideTextYes():** Funció que oculta l'atribut TextYes i invoca la funció ReturnToMenu al cap d'una estona.

```
public void HideTextYes()
{
    TextYes.GetComponent<Animation>().Stop("FadeInText");
    TextYes.GetComponent<Animation>().Play("FadeOutText");
    Invoke(nameof(ReturnToMenu), 3.0f);
}
```

+ **ReturnToMenu():** Funció que carrega l'escena del Menú principal.

```
public void ReturnToMenu() {

    SceneManager.LoadScene("Menu");
}
```

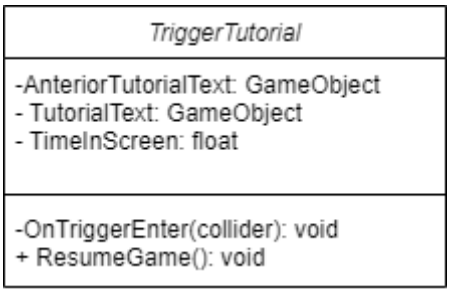
6.3 Implementació de l'entorn i objectes

En aquest apartat s'especifica la implementació i les classes d'elements que podem trobar en l'entorn dels nivells, mecàniques d'aquests elements i objectes amb els quals podem interactuar.

6.3.1 TriggerTutorial

En el primer nivell de tots hi ha elements invisibles per tota la zona que fan que, al passar a través d'ells, s'activi la interfície amb un missatge explicant una de les mecàniques que el jugador pot realitzar, ja sigui córrer, saltar, gronxar-se, etc. Després d'haver passat una estona, aquests missatge desapareixeria junt amb l'objecte invisible de l'escenari, per així evitar que hi tornés a sortir cada cop que es passés per aquella zona.

Model de la classe TriggerTutorial:



Atributs de TriggerTutorial:

- **AnteriorTutorialText:** Aquest GameObject l'utilitzarem per agafar l'anterior missatge tutorial que s'ha mostrat al jugador (si hi ha hagut un).

- **TutorialText:** GameObject que té assignat el Text que hi volem mostrar al jugador.

- **TimeInScreen:** Float que consisteix en el temps que volem que el text es mostri en la pantalla.

En la Figura 82 podem veure les propietats públiques assignades al component de la classe TutorialTrigger.

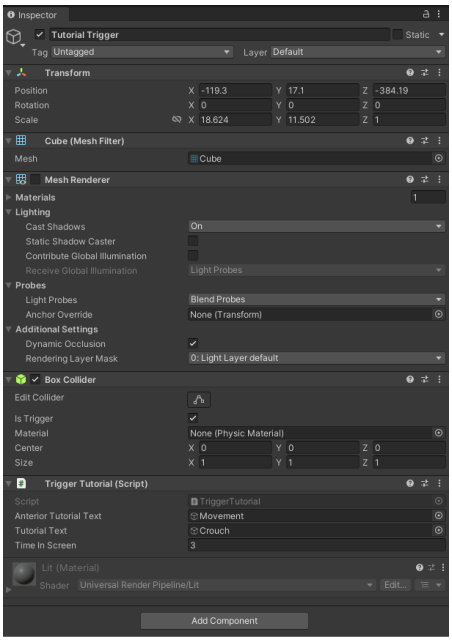


Figura 82: Propietats de la classe TutorialTrigger.

Mètodes de TutorialTrigger:

- **OnTriggerEnter(collider):** Aquesta funció només s'activa quan aquest objecte entra en contacte amb un altre objecte que tingui el component collider (en aquest cas, sempre serà el jugador). Primer comprova que el jugador no hagi completat el tutorial anteriorment. En cas de que no ho hagi fet, desactivarà el text del tutorial que hi havia anteriorment (en cas de que el jugador avanci segueixi avançant i no hagi passat el temps del tutorial anterior) i activa l'element TutorialText, acte seguit invoca la funció ResumeGame().

```
private void OnTriggerEnter(Collider other)
{
    if(PlayerPrefs.GetFloat("TutorialCompleted") == 1)
    {
        Destroy(gameObject);
    }
    else
    {
        AnteriorTutorialText.SetActive(false);
        TutorialText.SetActive(true);
        Invoke("ResumeGame", TimeInScreen);
    }

}
```

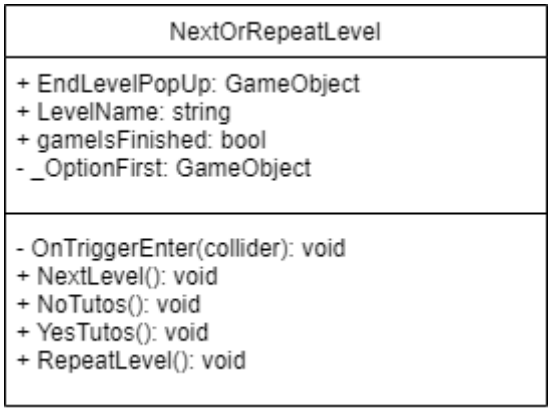
+ **ResumeGame():** Aquesta funció fa que es deixi de mostrar el text tutorial i destrueix el Objecte per evitar que es repeteixi el missatge cada cop que hi entri dintre del collider del GameObject.

```
public void ResumeGame()
{

    TutorialText.SetActive(false);
    Destroy(gameObject);

}
```

6.3.2 NextOrRepeatLevel

Element que apareix en tots els nivells del joc. Aquest objecte invisible és el que s'encarrega de detectar quan el jugador arriba al final del nivell. Un cop entra en contacte amb aquest, fa aparèixer el Canvas amb el mateix nom que la classe. També és l'encarregat de gestionar les funcions que s'activen quan el jugador selecciona un dels botons de la finestra de final de nivell.

Model de la classe NextOrRepeatLevel:



Atributs de la classe:

+ **EndLevelPopUp:** GameObject on hi guardem el Canvas que conté la finestra de final de nivell.

+ **LevelName:** String que utilitzem per guardar-hi el nom del següent nivell.

+ **gameIsFinished:** Booleà que utilitzarem per identificar si el jugador ha arribat al final del joc o no. Aquest atribut és importat ja que l'utilitzem en la classe PauseMenu explicada anteriorment (veure apartat 6.2.2).

- **_OptionFirst:** Atribut al qual assignem un dels botons per a que es seleccioni automàticament quan aparegui la finestra. Creat exclusivament pels que utilitzen comandament de consola.

En la Figura 83 podem veure les propietats públiques assignades al component de la classe NextOrRepeatLevel.

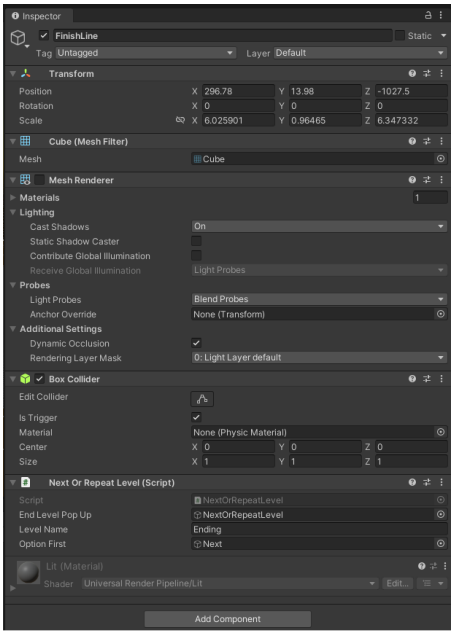


Figura 83: Propietats de la classe NextOrRepeatLevel.

Mètodes de NextOrRepeat

- **OnTriggerEnter(collider):** Aquesta funció només s'activa quan aquest objecte entra en contacte amb un altre objecte que tingui el component collider (en aquest cas, sempre serà el jugador). A l'entrar en contacte amb l'objecte, estableix el booleà a cert per indicar que s'ha arribat al final i acte seguit es marca com que el joc ha estat pausat. Un cop fet això, es mostra la finestra amb els botons per donar al jugador l'elecció de si tornar a jugar el mateix nivell o passar al següent. Addicionalment seleccionem un dels botons que hem establer com a _OptionFirst pels jugadors que utilitzen comandament de consola.

```
private void OnTriggerEnter(Collider other)
{
   gameIsFinished = true;
   Time.timeScale = 0f;
   PauseMenu.GameIsPaused = true;
   EndLevelPopUp.SetActive(true);
    EventSystem.current.SetSelectedGameObject(_OptionFirst);

}
```

+ **NextLevel():** Si el jugador escull el botó de passar al següent nivell, es desactiva la pausa del joc, marquem un altre cop el gameIsFinished a fals i apuntem en el PlayerPrefs que el tutorial no ha estat completat (ja que en els següents nivells hi podrien haver-hi més tutorials pel jugador). Acte seguit carreguem l'escena escollida amb el string.

```
public void NextLevel()
{

    Time.timeScale = 1f;
    PauseMenu.GameIsPaused = false;
    EndLevelPopUp.SetActive(false);
    gameIsFinished = false;
    PlayerPrefs.SetFloat("TutorialCompleted", 0);
    SceneManager.LoadScene(LevelName);
}
```

+ **NoTutos():** En el cas de que el jugador hagi decidit tornar a fer el nivell, si està en el tutorial, apareixerà un altre finestra preguntant si vol activar els tutorials o no. En el cas de no voler-los, els apuntem en el PlayerPrefs.

```
public void NoTutos()
{
    PlayerPrefs.SetFloat("TutorialCompleted",1);
}
```

+ **YesTutos():** En el cas de sí voler tornar a jugar el nivell amb els tutorials, aquesta funció apunta la decisió en el PlayerPrefs.

```
public void YesTutos()
{
    PlayerPrefs.SetFloat("TutorialCompleted", 0);
}
```

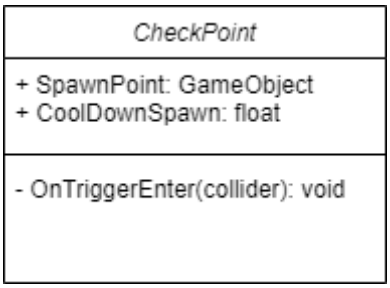
+ **RepeatLevel():** Similar al nivell de NextLevel(). Només varia en que l'escena que carrega es la mateixa en la que es troba i no apunta res al PlayerPrefs, ja que això ja ho ha realitzat alguna de les funcions anteriors.

```
public void RepeatLevel()
{
    Time.timeScale = 1f;
    PauseMenu.GameIsPaused = false;
    EndLevelPopUp.SetActive(false);
    gameIsFinished = false;

    SceneManager.LoadScene(SceneManager.GetActiveScene().name);
}
```

6.3.3 CheckPoint

Aquest objecte permet al jugador, un cop havent-lo creuat, canviar el punt d'aparició en el nivell. Per fer-ho simplement agafem el punt de reaparició del jugador i el desplacem al punt on es troba aquest objecte.

Model de la classe CheckPoint:



Atributs de Checkpoint:

+ **SpawnPoint:** Objecte que assignem al punt d'aparició del jugador.

+ **CoolDownSpawn:** Float que determina el temps que donem des que el jugador reapareix fins que el jugador por torna-hi a jugar.

Mètodes de Checkpoint:

- **OnTriggerEnter(collider):** funció que només s'activa quan aquest objecte entre en contacte amb un altre objecte que tingui el component collider (en aquest cas, sempre serà el jugador). A l'entrar en contacte amb un altre objecte, canvia la posició del GameObject SpawnPoint per la posició on es troba l'objecte assignat a aquesta classe. Acte seguit es destrueix per evitar que el jugador pugui perdre progrés (per exemple, té el punt d'aparició molt més endavant però torna enrere i sense voler entra en contacte amb un objecte que tenia assignada aquesta classe. Això faria que el punt d'aparició es canviés a un molt més enrere).

```
private void OnTriggerEnter(Collider other)
{
    SpawnPoint.transform.position = transform.position;
    Destroy(gameObject);
}
```

6.3.4 MovingPlatform

En alguns nivells, el jugador es trobarà amb algunes zones on hi pot haver plataformes amb moviment. L'objecte està estructurada de manera que, per una part, hi ha la plataforma amb un objecte Collision per detectar quan un jugador es situa a sobre de la plataforma, i per l'altre part trobem un objecte que hi guarda els punts per on la plataforma haurà de moure's (veure Figura 84). Per posar en funcionament l'objecte, utilitzarem 3 classes diferents: WaypointPath, PlayerFixSpeed i MovingPlatform.

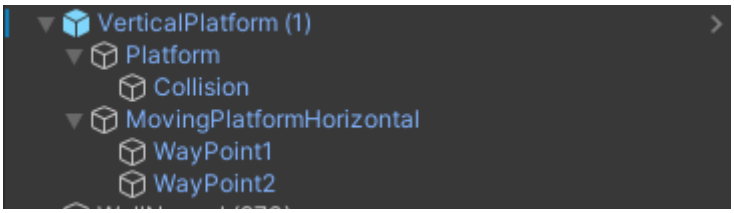
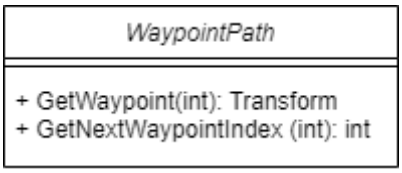


Figura 84: Elements que hi conté l'objecte de plataforma mòbil.

WaypointPath*:*

Aquesta classe auxiliar consisteix en gestionar els punts per on ha de passar la plataforma i proporcionar-los a la classe principal.

Model de la classe WaypointPath:



Mètodes de la classe WaypointPath:

+ **GetWaypoint(int):** Donat un número, retorna el fill del pare MovingPlatformHorizontal (en el cas de la Figura 84) que estigui en la posició indicada pel nombre.

```
public Transform GetWaypoint(int waypointIndex)
{
    return transform.GetChild(waypointIndex);
}
```

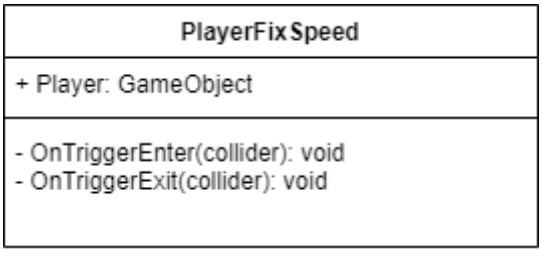
+ **GetNextWaypointIndex(int):** Donat un número, retorna el fill següent del pare MovingPlatformHorizontal. En cas de ser l'últim fill, es retorna el primer fill.

```
public int GetNextWaypointIndex(int currentWaypointIndex)
{
    int nextWaypointIndex = currentWaypointIndex + 1;

    if(nextWaypointIndex == transform.childCount)
    {
        nextWaypointIndex = 0;
    }

    return nextWaypointIndex;
}
```

PlayerFixSpeed:

Aquesta classe s'encarrega de regular la velocitat del jugador un cop està a sobre de la plataforma en moviment. La manera en la que es fa es molt senzilla, simplement movem el jugador dintre del objecte quan entra (fent que sigui el fill de la plataforma) i al sortir el trèiem del conjunt.

Model de la classe PlayerFixSpeed:



Atributs de la classe PlayerFixSpeed:

+ **Player:** GameObject que fa referència al jugador.

En la Figura 85 podem veure les propietats públiques assignades al component de la classe PlayerFixSpeed.

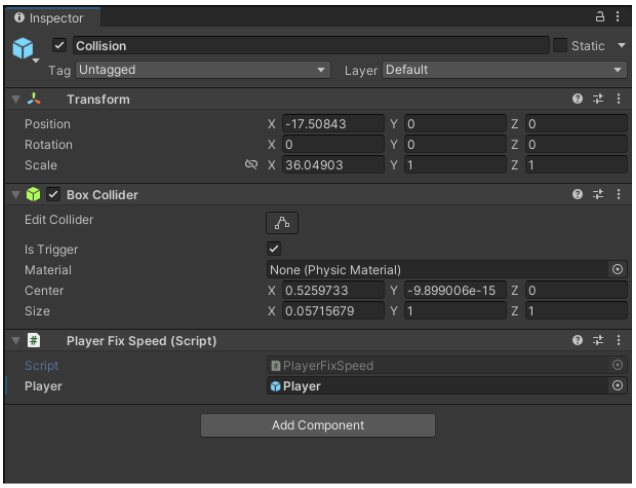


Figura 85: Propietats de la classe PlayerFixSpeed.

Mètodes de la classe PlayerFixSpeed:

- **OnTriggerEnter(collider):** Funció que s'activa quan el collider d'un altre objecte entra en contacte amb el collider d'aquests objectes. Quan el jugador entra dintre de la zona del collider, mou el jugador dintre de la plataforma en moviment per així mantenir la velocitat en relació a la plataforma.

```
private void OnTriggerEnter(Collider other)
{
    Player.transform.SetParent(transform);
}
```

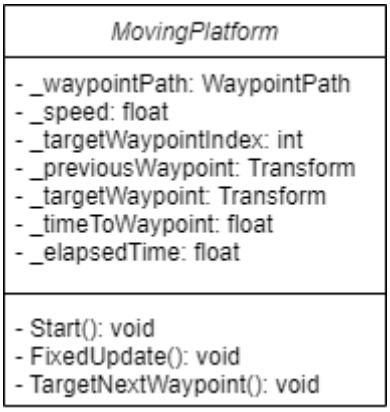
- **OnTriggerExit(collider):** Funció que s'activa quan el collider d'un altre objecte surt del collider d'aquests objectes. Quan el jugador surt del collider de la plataforma en moviment, aquests deixa de ser pare del jugador, fent que la velocitat torni a la normalitat.

```
private void OnTriggerExit(Collider other)
{
    Player.transform.SetParent(null);
}
```

MovingPlatform:

Per acabar, aquesta classe consisteix en gestionar el moviment de la plataforma. Segons la quantitat de punts que s'hagin establert per la zona i el temps donat, aquesta classe mourà l'objecte plataforma de manera cíclica entre els punts establerts anteriorment.

Model de la classe MovingPlatform:



Atributs de la classe MovingPlatform:

- **_waypointPath:** Script que està associat al grup de punts col·locats.
- **_speed:** Velocitat que volem que es mogui la plataforma.
- **_targetWaypointIndex:** Posició del punt dintre del pare.
- **_previousWaypoint:** Posició del punt anterior.
- **_targetWaypoint:** Punt a on s'ha de moure la plataforma.
- **_timeToWaypoint:** Temps que tarda en arribar a un punt.
- **_elapsedTime:** Temps transcorregut.

En la Figura 86 podem veure les propietats públiques assignades al component de la classe MovingPlatform.

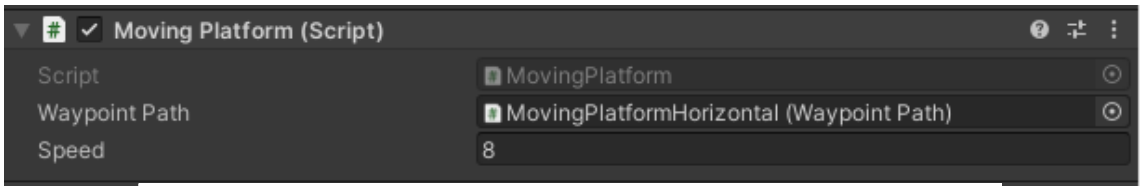


Figura 86: Propietats de la classe MovingPlatform.

Mètodes de la classe MovingPlatform:

- **Start():** Funció que s'executa al començament de l'escena. En aquests cas executa la funció TargetNextWaypoint() per agafar el primer punt.

```
    void Start()
    {
        TargetNextWaypoint();
    }
```

- **FixedUpdate():** Semblant a la funció Update amb la diferència que ofereix un moviment constat i fixe. S'encarrega de realitzar el moviment de la plataforma de manera constant i a una velocitat fixe gràcies a la funció Lerp de Unity. Un cop arriba al punt establert, crida la funció TargetNextWaypoint() per agafar el següent punt i seguir desplaçant-se.

```
    void FixedUpdate()
    {
        _elapsedTime += Time.deltaTime;

        float elapsedPercentage = _elapsedTime / _timeToWaypoint;
        transform.position = Vector3.Lerp(_previousWaypoint.position,
_targetWaypoint.position, elapsedPercentage);

        if(elapsedPercentage >= 1)
        {
            TargetNextWaypoint();
        }
    }
```

- **TargetNextWaypoint():** Funció que agafa el següent punt i calcula el temps que trigarà tenint en compte la distancia i velocitat.

```
    private void TargetNextWaypoint()
    {
        _previousWaypoint = _waypointPath.GetWaypoint(_targetWaypointIndex);
        _targetWaypointIndex =
_waypointPath.GetNextWaypointIndex(_targetWaypointIndex);
        _targetWaypoint = _waypointPath.GetWaypoint(_targetWaypointIndex);

        _elapsedTime = 0;

        float distanceToWaypoint =
Vector3.Distance(_previousWaypoint.position, _targetWaypoint.position);
        _timeToWaypoint = distanceToWaypoint / _speed;
    }
```

6.4 Implementació de la càmera

Per la implementació de la càmera en primera persona, hem decidit separar la càmera del jugador. La raó és per que Unity pot donar alguns errors si afegim la càmera com a component del jugador. Per sort, ho hem pogut solucionar-ho simplement movent la càmera fora d'objecte jugador i, dintre d'aquest, crear dos objectes que permetin agafar tant la orientació del jugador com el punt de vista (veure Figura 87).

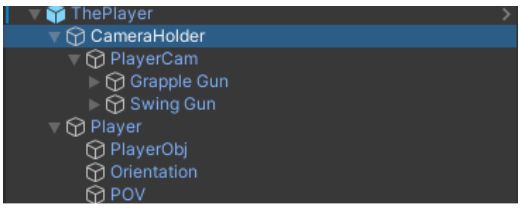
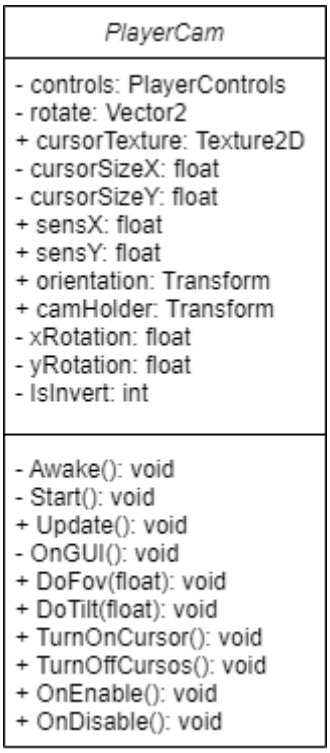


Figura 87: Distribució de la càmera per garantir un funcionament fluit.

6.4.1 PlayerCam

La classe és l'encarregada de moure la càmera en la direcció que el jugador proporcioni amb el ratolí o amb el comandament de consola.

Model de la classe PlayerCam:



Atributs de la classe PlayerCam:

- **controls:** Atribut que té per referencia els controls del jugador en el comandament de consola.

- **rotate:** Atribut on hi guardarem les rotacions del comandament de consola (es a dir, en quina direcció i distància s'ha mogut el botó assignat).

+ **cursorTexture:** Textura on hi guardem la imatge que utilitzarem com a cursor que es mostrarà en mig de la càmera. Importat per a que el jugador no es maregi quan mogui la càmera.

- **cursorSizeX:** Valor de la coordenada X de la imatge que utilitzarem com a cursor.

- **cursorSizeY:** Valor de la coordenada Y de la imatge que utilitzarem com a cursor.

+ **sensX:** Valor que utilitzarem per definir la velocitat a la que es mourà la càmera en el eix de les X.

+ **sensY:** Valor que utilitzarem per definir la velocitat a la que es mourà la càmera en el eix de les y.

+ **orientation:** Coordenades que hi guardarem la orientació del jugador.

+ **camHolder:** Coordenades on es guarda la posició on haurà de estar la càmera.

- **xRotation:** Atribut on es guarda el valor X de la rotació de la càmera.

- **yRotation:** Atribut on es guarda el valor Y de la rotació de la càmera.

- **IsInvert:** Booleà que indica si la coordenada Y de la càmera es vol invertir o no.

Mètodes de la classe PlayerCam:

- **Awake():** Funció utilitzada per inicialitzar valors abans de que l'escena comenci. En el nostre cas, omplim l'atribut control i el preparem per que llegeixi els valors de rotate per així que el jugador pugui moure la càmera amb el comandament de consola.

```
    private void Awake()
    {
        controls = new PlayerControls();
        controls.Gameplay.Rotation.performed += ctx => rotate =
ctx.ReadValue<Vector2>();
        controls.Gameplay.Rotation.canceled += ctx => rotate = Vector2.zero;
    }
```

- **Start():** Funció que s'inicia al començament de l'escena. Establim la mida del cursor, bloquegem i ocultem el ratolí per a que sempre es trobi enmig de la càmera i finalment omplim les variables sensX, sensY i IsInvert amb els valors guardats a PlayerPrefs (ja que aquests atributs es poden modificar en el menú d'opcions del menú principal o menú pausa).

```
    private void Start()
    {
        cursorSizeY = cursorSizeX = 20;
        Cursor.lockState = CursorLockMode.Locked;
        Cursor.visible = false;
        sensX = sensY = PlayerPrefs.GetFloat("masterSen");
        IsInvert = PlayerPrefs.GetInt("masterInvertY");

    }
```

+ **Update():** Funció que s'executa a cada frame. Agafem atributs del PlayerPrefs (per si el jugador ha modificat els valors en algun moment del nivell) i els utilitzem per moure la càmera cada cop que el jugador utilitzi el ratolí o el comandament de consola tenint en compte la sensibilitat establerta anteriorment i si el jugador ha volgut invertir la coordenada Y o no. A més també evita que, al prémer el botó pausa en el comandament, el ratolí aparegui enmig de l'escena.

```
public void Update()
{
    sensX = sensY = PlayerPrefs.GetFloat("masterSen");
    IsInvert = PlayerPrefs.GetInt("masterInvertY");
    if (PauseMenu.GameIsPaused)
    {
        if (!PauseMenu.UsingControl)
        {

            Cursor.lockState = CursorLockMode.None;
            Cursor.visible = true;
```

```
            }
        }
        else
        {
            Cursor.lockState = CursorLockMode.Locked;
            Cursor.visible = false;
            float mouseX = (Input.GetAxisRaw("Mouse X") + rotate.x) * sensX ;
            float mouseY = (Input.GetAxisRaw("Mouse Y")  + rotate.y) * sensY;


                yRotation += mouseX;

                if(IsInvert == 1)
                {
                    xRotation += mouseY;


                }
                else
                {
                    xRotation -= mouseY;

                }


                xRotation = Mathf.Clamp(xRotation, -90f, 90f);
                camHolder.rotation = Quaternion.Euler(xRotation, yRotation, 0);
                orientation.rotation = Quaternion.Euler(0, yRotation, 0);


        }



    }
```

- **OnGUI():** Funció que utilitzem per dibuixar la imatge establerta enteriorment.

```
    private void OnGUI()
    {
        GUI.DrawTexture(new Rect(Screen.width / 2, Screen.height / 2,
cursorSizeX, cursorSizeY), cursorTexture);
    }
```

+ **DoFov(float):** Funció que estableix el camp de visió segons el float donat.

```
    public void DoFov(float endValue)
    {
        GetComponent<Camera>().DOFieldOfView(endValue, 0.25f);
    }
```

+ **DoTilt(float):** Funció que gira la càmera lleugerament segons el valor establert en el float donat.

```
    public void DoTilt(float zTilt)
    {
        transform.DOLocalRotate(new Vector3(0, 0, zTilt), 0.25f);
    }
```

+ **TurnOnCursor():** Mostra la imatge del cursor establert anteriorment.

```
public void TurnOnCursor()
{
    cursorSizeX = 20;
    cursorSizeY = 20;

}
```

+ **TurnOffCursor():** Amaga el cursor modificant el valor de mida a 0.

```
public void TurnOffCursor()
{
    cursorSizeX = 0;
    cursorSizeY = 0;

}
```

+ **OnEnable():** Funció creada pel comandament de consola. Permet activar els controls de moviments del comandament.

```
private void OnEnable()
{
    controls.Gameplay.Enable();
}
```

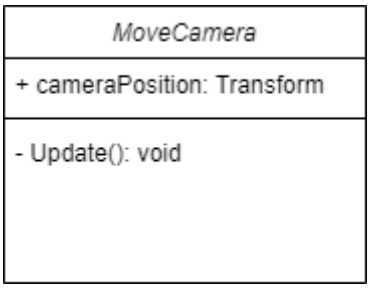
+ **OnDisable():** Funció creada pel comandament de consola. Permet desactivar els controls de moviments del comandament.

```
private void OnDisable()
{
    controls.Gameplay.Disable();
}
```

6.4.2 MoveCamera

Classe que hem creat per a que la càmera es mogui amb el jugador.

Model de la classe MoveCamera:



Atributs de la classe MoveCamera:

+ **cameraPosition:** Coordenades on guardem la posició de la càmera.

Mètodes de la classe MoveCamera:

- **Update():** Funció que s'activa a cada frame. Mou l'objecte a la posició de la cameraPosition.

```
private void Update()
{
    transform.position = cameraPosition.position;
}
```

6.5 Implementació del jugador

Aquest apartat correspon a una de les implementacions més importants de tot el videojoc per garantir una experiència positiva en el jugador. L'objecte del personatge principal és l'encarregat de registrar els inputs del jugador i reflectir-los en accions del joc, com les mecàniques de parkour o el tornar aparèixer quan es falla un nivell. En aquest apartat no només desglossarem tot el moviment que el jugador pot realitzar, si no també explicarem totes les mecàniques que el joc ofereix per poder superar el videojoc amb comoditat.

6.5.1 Player

L'objecte jugador és l'encarregat de només guardar tots els objectes necessaris pel funcionament de les mecàniques i moviments, si no també és l'encarregat de portar l'objecte càmera pel moviment en primera persona vist en l'apartat anterior (veure apartat 6.4). Concretament, l'objecte Player està format pels següents objectes fill:

- **CameraHolder:** Conté el script MoveCamera.
    - **PlayerCam:** Conté la càmera per on el jugador pot veure l'escena, un Audio Listener que permet escoltar el so del videojoc, l'element Post-process Volume que és la que s'encarrega de generar l'efecte de Neó en les textures dels objectes, un Canvas Renderer i finalment el script de PlayerCam.
        - **Grapple Gun:** Conté un Line Renderer per dibuixar la corda de la mecànica d'enganxar-se i el script que s'encarrega de mostrar-lo.
            - **GunTip:** Objecte per on surt la corda de la mecànica d'enganxar-se.
        - **Swing Gun:** Conté un Line Renderer per dibuixar la corda de la mecànica de gronxar-se i el script que s'encarrega de mostrar-lo.
            - **GunTip:** Objecte per on surt la corda de la mecànica de gronxar-se.

- **Player:** Conté el Rigidbody del personatge jugable i tots els scripts necessaris pel funcionament del moviment i les mecàniques.
    - **PlayerObj:** Conté un Mesh Renderer on es mostra la figura del personatge així com un Capsule Collider per interactuar amb els objectes dels escenaris.
    - **Orientation:** Objecte per saber l'orientació de l'objecte Player.
    - **POV:** Objecte que determina a on es posicionarà la càmera.

En la Figura 88 podem veure l'estructura del prefab del personatge que controla el jugador amb els objectes anteriorment mencionats.

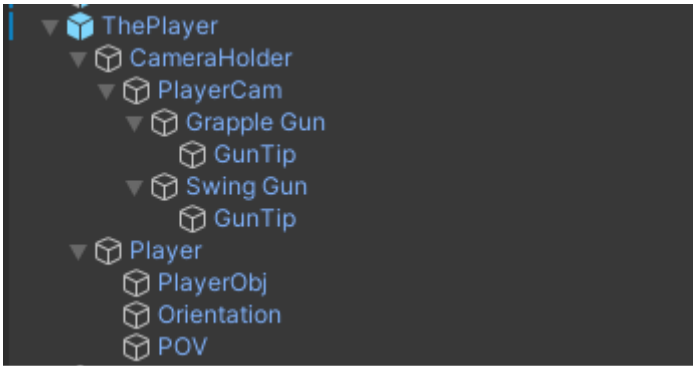


Figura 88: Prefab del personatge principal

Al ser un joc en primera persona, hem pensat abstenir-nos de crear un model pel jugador ja que ningú el veurà mai. Per aquesta raó ens ha semblat idoni només incorporar un collider de la mateixa mida que l'objecte del Player. Aquest collider s'encarregarà de detectar les col·lisions amb tot l'entorn, ja siguin parets, terra, rampes o plataformes en moviment.

6.5.2 Player Movement Advanced

La classe de Player Movement Advanced és l'encarregada de controlar el moviment, el control d'inputs i les mecàniques més bàsiques del jugador com saltar o ajupir-se, així com emmagatzemar variables que més tard ens seran útils per a la resta de mecàniques que estan distribuïdes en altres classes.

Per al moviment del jugador, utilitzem dos mètodes per poder moure al personatge: El primer utilitzem el sistema de Inputs que té Unity per gestionar els controls amb el teclat, mentre que el segon utilitzem una llibreria anomenada Input System pels controls amb el comandament de consola. En la Figura 89 podem observar la composició del nostre InputSystem anomenat PlayerControls que simplement guarda quins botons del comandament corresponen al les accions que pot realitzar el jugador. D'aquesta manera tenim un millor control dels dos tipus de controls de moviment per poder oferir la millor experiència possible.

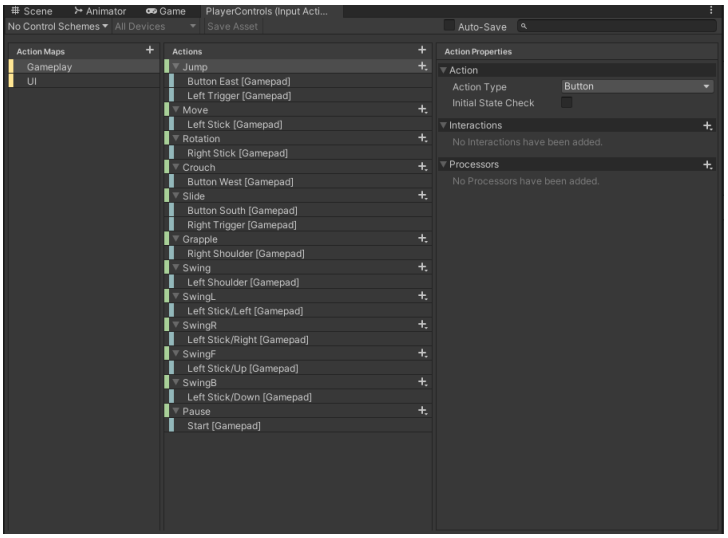
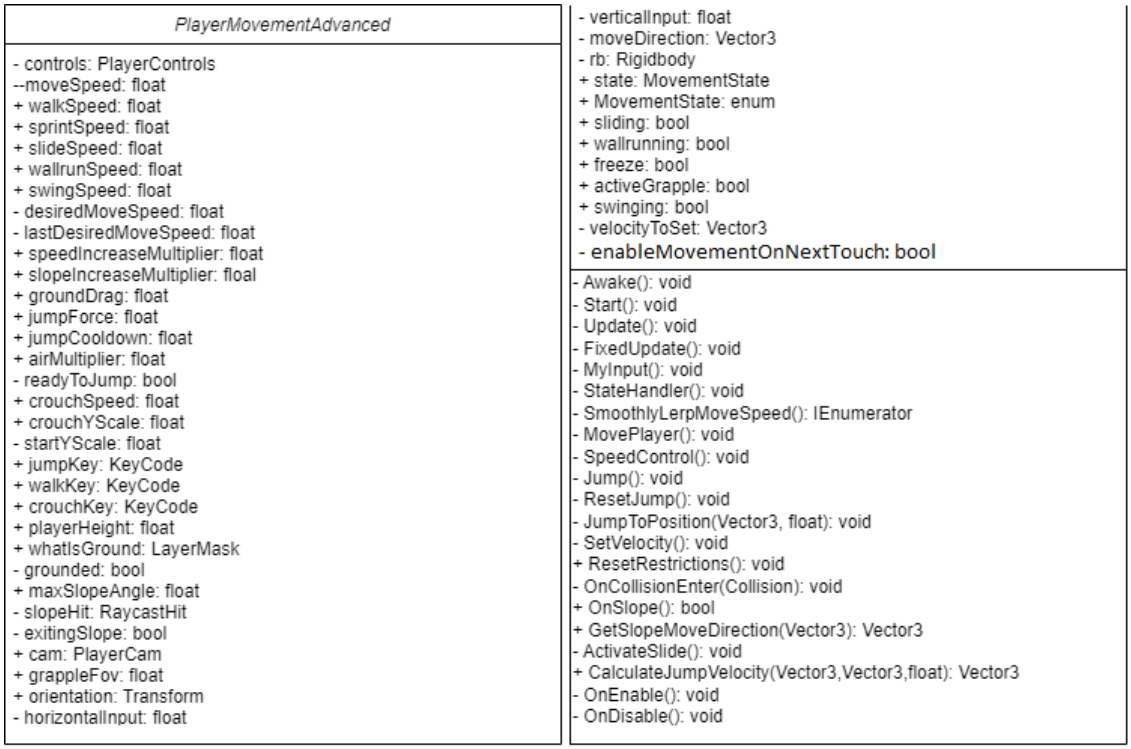


Figura 89: Contingut del element PlayerControls

Com es pot veure, també es genera una classe anomenada PlayerControls.cs, que és la que s'encarrega del funcionament d'aquesta llibreria. Al ser un script generat automàticament, creiem innecessari parlar-ne en detall en aquest apartat.

Model de la classe PlayerMovementAdvanced:



Atributs de la classe PlayerMovementAdvanced:

- **Controls:** Variable on hi assignem el PlayerControl que s'utilitza per gestionar els controls dels jugador amb el comandament de consola.

- **moveSpeed:** Valor que agafem per definir la velocitat a la que es mourà el jugador pel nivell.

+ **WalkSpeed:** Valor que utilitzarem per establir la velocitat a la que es mourà el jugador quan vulgui caminar (mecànica que no utilitzarem en el prototip però pot ser útil de cares a futur)

+ **sprintSpeed:** Valor que agafem per definir la velocitat a la que es mourà el jugador quan vulgui córrer (semblant a la mecànica de caminar, en el prototip només la utilitzem per definir el moviment del jugador predeterminat).

+ **slideSpeed:** Velocitat a la que el jugador és mourà quan estigui lliscant per una rampa.

+ **wallrunSpeed:** Velocitat a la que el jugador es mourà al fer una carrera de paret.

+ **swingSpeed:** Velocitat a la que el jugador es mourà al gronxar-se.

- **desiredMoveSpeed:** Velocitat màxima a la que volem que el jugador arribi al moure's.

- **lastDesiredMoveSpeed:** Valor que guarda si hi ha hagut una altre velocitat màxima desitjada anteriorment.

+ **speedIncreaseMultiplier:** Valor on guardem el multiplicador de velocitat.

+ **slopeIncreaseMultiplier:** Valor on guardem el multiplicador d'increment de la rampa.

+ **groundDrag:** Valor on guardem la fricció del terra respecta al jugador.

+ **jumpForce:** Força amb la que el jugador podrà saltar.

+ **jumpCooldown:** Valor on es guarda un petit comptador que impedeix al jugador saltar.

+ **airMultiplier:** Valor on guardem el multiplicador de velocitat del jugador a l'aire.

- **readyToJump:** Booleà on guardem quan el jugador pot saltar.

+ **crouchSpeed:** Velocitat del jugador al ajupir-se

+ **crouchYScale:** Valor on guardem la mida que agafa el model del jugador un cop està ajupit.

- **startYScale:** Valor on guardem la mida del jugador abans d'estar ajupit.

+ **jumpKey:** KeyBind on hi assignem la tecla de saltar.

+ **walkKey:** KeyBind on hi assignem la tecla de caminar.

+ **crouchKey:** KeyBind on hi assignem la tecla d'ajupir-se.

+ **playerHeight:** Valor on guardem l'altura del jugador.

+ **whatIsGround:** En aquest atribut es on guardarem la capa que indica quin objecte és considerat terra.

- **grounded:** Booleà que indica si el jugador està a terra o no.

+ **maxSlopeAngle:** Valor on guardem el grau màxim de les rampes que el jugador pot creuar.

- **slopeHit:** RaycastHit que s'encarrega de detectar quan un jugador està pujant o baixant una rampa.

- **exitingSlope:** Booleà que detecta quan el jugador hagi sortit d'una rampa.

+ **cam:** Atribut on hi haurà assignat la càmera del jugador.

+ **grappleFov:** Camp de visió que agafa el jugador quan decideix utilitzar la mecànica d'enganxar-se.

+ **orientation:** Atribut on hi assignarem cap a on està orientat el jugador.

- **horizontalInput:** Atribut float on hi guardarem els Inputs dels moviments horitzontals del teclat.

- **verticalInput:** Atribut float on hi guardarem els Inputs dels moviments verticals del teclat.

- **moveDirection:** Un Vector on emmagatzemarem la direcció a la que el jugador es mou.

- **rb:** Atribut on hi assignarem el rigidbody del jugador.

+ **State:** Aquest atribut serveix per crear una màquina d'estats que servirà per saber que esta fent el jugador en tot el moment.

+ **MovementState:** Atributs on hi guardem tipus de estats que el jugador pot adoptar, es divideixen en: freeze, walking, grappling, swinging, sprinting, wallrunning, crouching, sliding i air.

+ **sliding:** Booleà que utilitzarem per indicar quan el jugador està lliscant.

+ **wallrunning:** Booleà que utilitzarem per indicar quan el jugador està corrent per la paret.

+ **freeze:** Booleà que utilitzarem per indicar quan el jugador està congelat (és a dir, ni es mou, ni pot realitzar cap altra acció).

+ **activeGrapple:** Booleà que utilitzarem per indicar quan el jugador està enganxant-se.

+ **swinging:** Booleà que utilitzarem per indicar quan el jugador s'està gronxant.

- **velocityToSet:** Vector que indica quina velocitat agafa el jugador al gronxar-se.

- **EnableMovementOnNextTouch:** Booleà que indica quan el jugador podrà moure's després de gronxar-se.

En la Figura 90 podem veure les propietats públiques assignades al component de la classe PlayerMovementAdvanced.

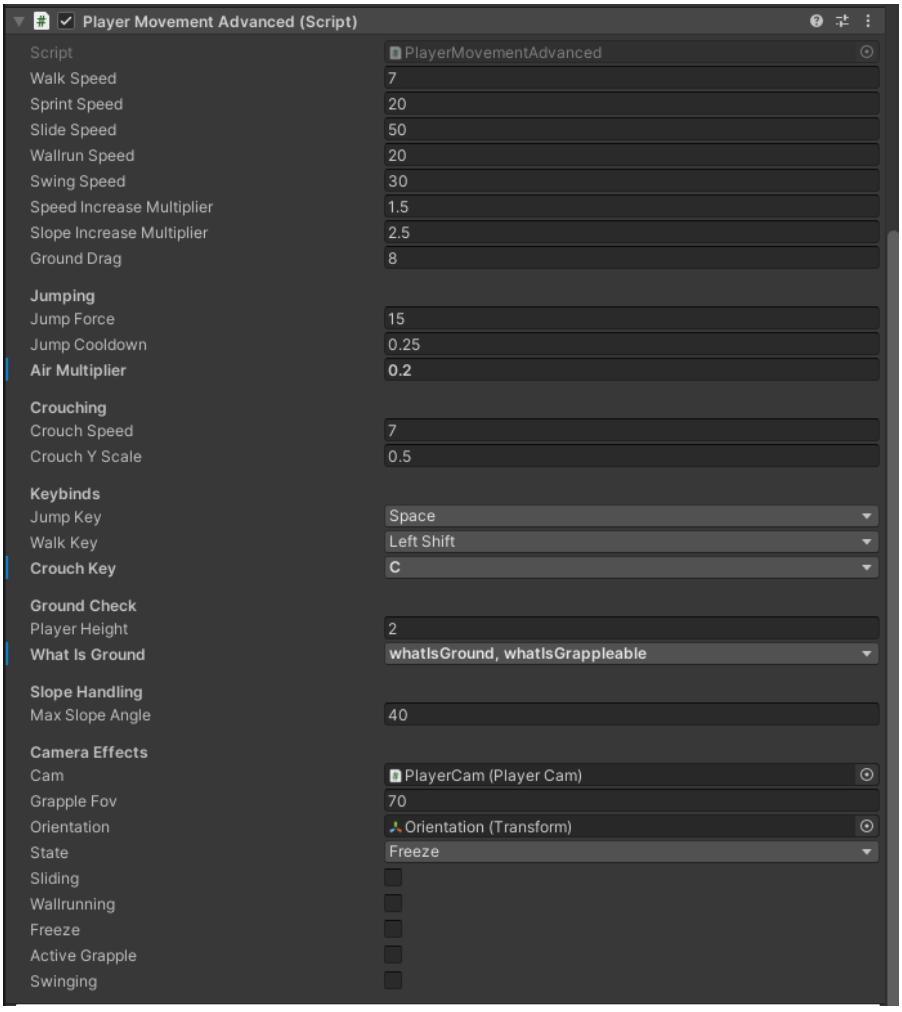


Figura 90: Propietats de la classe PlayerMovementAdvanced.

Mètodes de la classe PlayerMovementAdvanced

- **Awake():** Funció que s'executa un cop al iniciar l'escena. Simplement omple la variable controls.

```
    private void Awake()
    {
        controls = new PlayerControls();
    }
```

- **Start ():** Funció que s'executa al començament de l'escena. Omplim els atributs rb i startYscale així com establir el readyToJump com a cert.

```
    private void Start()
    {
        rb = GetComponent<Rigidbody>();
        rb.freezeRotation = true;

        readyToJump = true;

        startYScale = transform.localScale.y;
    }
```

- **Update():** Funció que s'executa cada frame de l'escena. Comprova que el jugador es trobi en el terra, acte seguit crida les funcions de MyInput(), SpeedControl() i StateHandler(). Finalment s'aplica la fricció del terra en cas de que el jugador no estigui en l'aire.

```
    private void Update()
    {
        // ground check
        grounded = Physics.Raycast(transform.position, Vector3.down,
playerHeight * 0.5f + 0.2f, whatIsGround);

        MyInput();
        SpeedControl();
        StateHandler();

        // handle drag
        if (grounded && !activeGrapple)
            rb.drag = groundDrag;
        else
            rb.drag = 0;
    }
```

- **FixedUpdate():** Semblant a la funció Update amb la diferència que ofereix un moviment constat i fixe. Simplement crida la funció MovePlayer() que és l'encarregada del moviment del jugador.

```
    private void FixedUpdate()
    {
        MovePlayer();
    }
```

- **MyInput():** Funció encarregada de gestionar el Inputs tant per teclat com per comandament de consola.

```
    private void MyInput()
    {
        horizontalInput = Input.GetAxisRaw("Horizontal");
        verticalInput = Input.GetAxisRaw("Vertical");

        // when to jump
        if ((Input.GetKey(jumpKey) || controls.Gameplay.Jump.triggered) &&
readyToJump && grounded)
        {
            readyToJump = false;

            Jump();

            Invoke(nameof(ResetJump), jumpCooldown);
        }

        // start crouch
        if (Input.GetKeyDown(crouchKey) ||
controls.Gameplay.Crouch.WasPressedThisFrame())
        {
            transform.localScale = new Vector3(transform.localScale.x,
transform.localScale.y/2, transform.localScale.z);
            rb.AddForce(Vector3.down * 5f, ForceMode.Impulse);
        }

        // stop crouch
        if (Input.GetKeyUp(crouchKey) ||
controls.Gameplay.Crouch.WasReleasedThisFrame())
        {
            transform.localScale = new Vector3(transform.localScale.x,
transform.localScale.y*2, transform.localScale.z);
        }
    }
```

- **StateHandler():** Funció encarregada de modificar la velocitat del jugador segons a quin estat es trobi. Per exemple, si està corrent per la paret, la funció modificarà la velocitat del jugador per la velocitat establerta en la variable wallrunSpeed.

```
private void StateHandler()
    {
        //Mode - freeze
        if (freeze)
        {
            state = MovementState.freeze;
            desiredMoveSpeed = 0;
            rb.velocity = Vector3.zero;
        }

        // Mode - Wallrunning
        else if (wallrunning)
        {
            state = MovementState.wallrunning;
            desiredMoveSpeed = wallrunSpeed;
        }


        // Mode - Sliding
        if (sliding)
        {
            state = MovementState.sliding;

            if (OnSlope() && rb.velocity.y < 0.1f)
                desiredMoveSpeed = slideSpeed;

            else
                desiredMoveSpeed = sprintSpeed;
        }
        // Mode - Grappling
        else if(activeGrapple)
        {
            GetComponent<Grappling>().grappleKey = KeyCode.None;
            GetComponent<Sliding>().slideKey = KeyCode.None;
            controls.Gameplay.Grapple.Disable();
            controls.Gameplay.Slide.Disable();
            Invoke(nameof(ActivateSlide), 0.25f);
            state = MovementState.grappling;
            moveSpeed = sprintSpeed;
        }
        // Mode - Swinging
        else if(swinging)
        {
            state = MovementState.swinging;
            //moveSpeed = swingSpeed;
            desiredMoveSpeed = swingSpeed;
        }
        // Mode - Crouching
        else if (Input.GetKey(crouchKey) ||
controls.Gameplay.Crouch.IsInProgress())
        {
            state = MovementState.crouching;
            desiredMoveSpeed = crouchSpeed;
        }

        // Mode - Sprinting
        else if (grounded && Input.GetKey(walkKey))
        {
            state = MovementState.walking;
            desiredMoveSpeed = walkSpeed;
        }
```

```
        // Mode - Walking
        else if (grounded)
        {
            state = MovementState.sprinting;
            desiredMoveSpeed = sprintSpeed;
        }

        // Mode - Air
        else
        {
            state = MovementState.air;
        }
        moveSpeed = desiredMoveSpeed;
        lastDesiredMoveSpeed = desiredMoveSpeed;
    }
```

- **SmoothlyLerpMoveSpeed():** Funció que s'encarrega d'anar augmentat la velocitat que agafa el jugador poc a poc mentre s'està movent fins a arribar a la desitjada.

```
    private IEnumerator SmoothlyLerpMoveSpeed()
    {
        // smoothly lerp movementSpeed to desired value
        float time = 0;
        float difference = Mathf.Abs(desiredMoveSpeed - moveSpeed);
        float startValue = moveSpeed;

        while (time < difference)
        {
            moveSpeed = Mathf.Lerp(startValue, desiredMoveSpeed, time /
difference);

            if (OnSlope())
            {
                float slopeAngle = Vector3.Angle(Vector3.up, slopeHit.normal);
                float slopeAngleIncrease = 1 + (slopeAngle / 90f);

                time += Time.deltaTime * speedIncreaseMultiplier *
slopeIncreaseMultiplier * slopeAngleIncrease;
            }
            else
                time += Time.deltaTime * speedIncreaseMultiplier;

            yield return null;
        }

        moveSpeed = desiredMoveSpeed;
    }
```

- **MovePlayer():** Funció encarregada de moure al jugador a la velocitat establera i en la direcció definida pels controls de moviment.

```
private void MovePlayer()
    {
        if (activeGrapple) return;
        if (swinging) return;

        if (swinging) return;
        // calculate movement direction
        moveDirection = orientation.forward * verticalInput +
orientation.right * horizontalInput;

        // on slope
        if (OnSlope() && !exitingSlope)
```

```
        {
            rb.AddForce(GetSlopeMoveDirection(moveDirection) * moveSpeed *
20f, ForceMode.Force);

            if (rb.velocity.y > 0)
                rb.AddForce(Vector3.down * 80f, ForceMode.Force);
        }

        // on ground
        else if (grounded)
            rb.AddForce(moveDirection.normalized * moveSpeed * 10f,
ForceMode.Force);

        // in air
        else if (!grounded)
            rb.AddForce(moveDirection.normalized * moveSpeed * 10f *
airMultiplier, ForceMode.Force);

        // turn gravity off while on slope
        if (!wallrunning) rb.useGravity = !OnSlope();
    }
```

- **SpeedControl():** Funció que impedeix al jugador superar la velocitat establerta, ja sigui en una rampa, en l'aire o en el terra.

```
private void SpeedControl()
    {
        if (activeGrapple) return;

        // limiting speed on slope
        if (OnSlope() && !exitingSlope)
        {
            if (rb.velocity.magnitude > moveSpeed)
                rb.velocity = rb.velocity.normalized * moveSpeed;
        }

        // limiting speed on ground or in air
        else
        {
            Vector3 flatVel = new Vector3(rb.velocity.x, 0f, rb.velocity.z);

            // limit velocity if needed
            if (flatVel.magnitude > moveSpeed)
            {
                Vector3 limitedVel = flatVel.normalized * moveSpeed;
                rb.velocity = new Vector3(limitedVel.x, rb.velocity.y,
limitedVel.z);
            }
        }
    }
```

- **Jump():** Funció que executa el salt al jugador.

```
    private void Jump()
    {
        exitingSlope = true;

        // reset y velocity
        rb.velocity = new Vector3(rb.velocity.x, 0f, rb.velocity.z);

        rb.AddForce(transform.up * jumpForce, ForceMode.Impulse);
    }
```

- **ResetJump():** Funció que restableix la capacitat del jugador de saltar.

```
    private void ResetJump()
    {
        readyToJump = true;

        exitingSlope = false;
    }
```

- **JumpToPosition(Vector3, float):** Funció que permet al jugador saltar fins a una posició donada (funció que s'utilitza en la mecànica d'enganxar-se)

```
    public void JumpToPosition(Vector3 targetPosition, float trajectoryHeight)
    {
        activeGrapple = true;

        velocityToSet = CalculateJumpVelocity(transform.position,
targetPosition, trajectoryHeight);
        Invoke(nameof(SetVelocity), 0.1f);

        Invoke(nameof(ResetRestrictions),3f);
        GetComponent<Sliding>().slideKey = KeyCode.LeftControl;
        controls.Gameplay.Slide.Enable();
    }
```

- **SetVelocity():** Funció que dóna al jugador la capacitat de moure's a la velocitat desitjada.

```
    private void SetVelocity()
    {

        enableMovementOnNextTouch = true;
        rb.velocity = velocityToSet;
        cam.DoFov(grappleFov);
    }
```

+ **ResetRestrictions():** Desactiva el grapple i torna el camp de visió a la normalitat.

```
    public void ResetRestrictions()
    {
        activeGrapple = false;
        cam.DoFov(60);
    }
```

- **OnCollisionEnter(Collision):** Funció que s'activa quan un objecte entra en contacte amb l'objecte associat a aquesta classe. Al tocar un altre objecte, el jugador deixa de tenir l'estat grapple i crida la funció ResetRestrictions().

```
    private void OnCollisionEnter(Collision collision)
    {
        if(enableMovementOnNextTouch)
        {
            enableMovementOnNextTouch = false;
            ResetRestrictions();
            GetComponent<Grappling>().StopGrapple();
        }
    }
```

+ **OnSlope():** Funció que detecta si el jugador està en una rampa o no.

```
    public bool OnSlope()
    {
        if (Physics.Raycast(transform.position, Vector3.down, out slopeHit,
playerHeight * 0.5f + 0.3f))
        {
            float angle = Vector3.Angle(Vector3.up, slopeHit.normal);
            return angle < maxSlopeAngle && angle != 0;
        }

        return false;
    }
```

+ **GetSlopeMoveDirection(Vector3):** Retorna la direcció a la que es mou el jugador en la rampa.

```
    public Vector3 GetSlopeMoveDirection(Vector3 direction)
    {
        return Vector3.ProjectOnPlane(direction, slopeHit.normal).normalized;
    }
```

- **ActivateSlide():** Funció que permet al jugador lliscar.

```
    private void ActivateSlide()
    {
        GetComponent<Sliding>().slideKey = KeyCode.LeftControl;
        controls.Gameplay.Slide.Enable();
    }
```

+ **CalculateJumpVelocity(Vector3, Vector3, float):** Funció que calcula l'impuls necessari per poder saltar a la posició final donada.

```
    public Vector3 CalculateJumpVelocity(Vector3 startPoint, Vector3 endPoint,
float trajectoryHeight)
    {
        float gravity = Physics.gravity.y;
        float displacementY = endPoint.y - startPoint.y;
        Vector3 displacementXZ = new Vector3(endPoint.x - startPoint.x, 0f,
endPoint.z - startPoint.z);

        Vector3 velocityY = Vector3.up * Mathf.Sqrt(-2 * gravity *
trajectoryHeight);
        Vector3 velocityXZ = displacementXZ / (Mathf.Sqrt(-1 *
trajectoryHeight / gravity) + Mathf.Sqrt(2 * (displacementY -
trajectoryHeight) / gravity));

        return velocityXZ + velocityY;
    }
```

- **OnEnable():** Funció creada pel comandament de consola. Permet activar els controls de moviments del comandament.

```
private void OnEnable()
{
    controls.Gameplay.Enable();
}
```

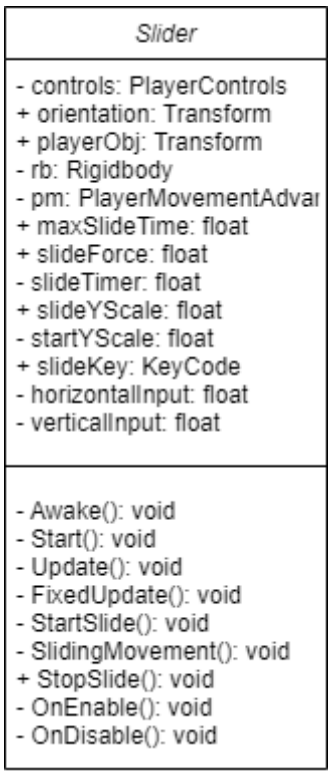
- **OnDisable():** Funció creada pel comandament de consola. Permet desactivar els controls de moviments del comandament.

```
private void OnDisable()
{
    controls.Gameplay.Disable();
}
```

6.5.3 Sliding

Aquesta classe és la que permet al jugador lliscar pel nivell, no només pel terra, si no per l'aire. A més, si ho realitza mentre està baixant una rampa, augmenta la velocitat per sobre de la màxima, fent que hi pugui saltar encara més lluny.

Model de la classe Sliding:



Atributs de la classe sliding:

- **controls:** Atribut que té per referencia els controls del jugador en el comandament de consola.

+ **Orienation:** Component que indica cap a on està orientat el jugador.

+ **playerObj:** Atribut on hi guardem la posició del jugador.

- **rb:** Atribut on hi assignarem el rigidbody del jugador.

- **pm:** Atribut on hi assignem el script PlayerMovementAdvanced que està vinculat al jugador.

+ **maxSlideTime:** Valor que guarda el temps màxim (en segons) que el jugador pot estar lliscant.

+ **slideForce:** Valor que indica la forca que proporcionarà al jugador per desplaçar-se al lliscar.

- **SlideTimer:** Float que comptarà el temps que el jugador està lliscant.

+ **slideYScale:** Mida que agafarà el jugador al lliscar.

- **StartYScale:** Mida inicial del jugador abans de lliscar.

+ **SlideKey:** KeyCode on hi guardarem la tecla de lliscar del teclat.

- **horizontalInput:** Atribut float on hi guardarem els Inputs dels moviments horitzontals dels teclats.

- **verticalInput:** Atribut float on hi guardarem els Inputs dels moviments horitzontals dels teclats.

En la Figura 91 podem veure les propietats públiques assignades al component de la classe Sliding.

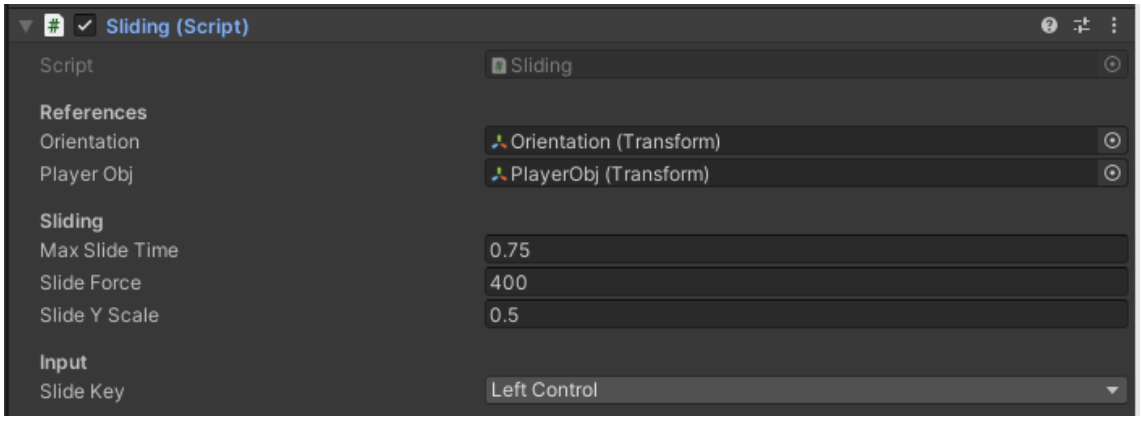


Figura 91: Propietats de la classe Sliding.

Mètodes de la classe sliding:

- **Awake():** Funció que s'executa un cop al iniciar l'escena. Simplement omple la variable controls.

```
    private void Awake()
    {
        controls = new PlayerControls();
    }
```

- **Start():** Funció que s'executa al primer frame de l'escena. Aquesta funció omple els atributs rb i pm amb els components del jugador corresponents, a més d'omplir l'atribut startYScale amb la mida del model del jugador.

```
    private void Start()
    {
        rb = GetComponent<Rigidbody>();
        pm = GetComponent<PlayerMovementAdvanced>();

        startYScale = playerObj.localScale.y;
    }
```

- **Update():** Funció que s'executa a cada frame de l'escena. Apart d'agafar els atributs dels inputs, s'encarrega de, en cas de prémer el botó de lliscar, cridarà la funció StartSlide(), un cop el deixi de pressionar cridarà a la funció stopSlide().

```
    private void Update()
    {
        horizontalInput = Input.GetAxisRaw("Horizontal");
        verticalInput = Input.GetAxisRaw("Vertical");

        if ((Input.GetKeyDown(slideKey) ||
controls.Gameplay.Slide.WasPressedThisFrame()) && (horizontalInput != 0 ||
verticalInput != 0))
        {
            StartSlide();
        }

        if ((Input.GetKeyUp(slideKey) ||
controls.Gameplay.Slide.WasReleasedThisFrame()) && pm.sliding)
        {
            StopSlide();
        }
    }
```

- **FixedUpdate():** Semblant a la funció Update amb la diferència que ofereix un moviment constat i fixe. Si l'estat del jugador està en Sliding, cridarà a la funció SlidingMovement().

```
    private void FixedUpdate()
    {
        if (pm.sliding)
            SlidingMovement();
    }
```

- **StartSlide():** Funció que inicialitza la mecànica de llsicar. Simplement canvia l'estat del jugador a sliding, modifica la mida del jugador i li afegeix una petita força al rigidbody del jugador, acte seguit inicialitza el comptador.

```
    private void StartSlide()
    {
          pm.sliding = true;

          playerObj.localScale = new Vector3(playerObj.localScale.x,
slideYScale, playerObj.localScale.z);
          rb.AddForce(Vector3.down * 5f, ForceMode.Impulse);

          slideTimer = maxSlideTime;

    }
```

- **SlidingMovement():** Funció que gestiona el moviment del jugador quan està lliscant. Si el jugador està lliscant en terra pla, lliscarà durant un temps definit per l'atribut maxSlideTime, si està lliscant cap avall en una rampa, lliscarà fins que el jugador deixi de prémer el botó o bé la rampa acabi.

```
    private void SlidingMovement()
    {
        Vector3 inputDirection = orientation.forward * verticalInput +
orientation.right * horizontalInput;

        // sliding normal
        if(!pm.OnSlope() || rb.velocity.y > -0.1f)
        {
            rb.AddForce(inputDirection.normalized * slideForce,
ForceMode.Force);

            slideTimer -= Time.deltaTime;
        }

        // sliding down a slope
        else
        {
            rb.AddForce(pm.GetSlopeMoveDirection(inputDirection) * slideForce,
ForceMode.Force);
        }

        if (slideTimer <= 0)
            StopSlide();
    }
```

+ **StopSlide():** Canvia l'estat del jugador a que ja no està lliscant i el torna a la mida original.

```
public void StopSlide()
    {
        pm.sliding = false;

        playerObj.localScale = new Vector3(playerObj.localScale.x,
startYScale, playerObj.localScale.z);
    }
```

- **OnEnable():** Funció creada pel comandament de consola. Permet activar els controls de moviments del comandament.

```
private void OnEnable()
{
    controls.Gameplay.Enable();
}
```

- **OnDisable():** Funció creada pel comandament de consola. Permet desactivar els controls de moviments del comandament.

```
private void OnDisable()
{
    controls.Gameplay.Disable();
}
```

6.5.4 Wall Running

La classe Wall Running s'encarrega de gestionar la mecànica de carrera de paret. Apart de permetre córrer per una paret, aquesta classe també dona l'opció al jugador de saltar a un altra paret mentre s'està corrent per així continuar sense tocar el terra. També ve inclòs una opció per baixar ràpidament utilitzant la tecla de ajupir-se mentre s'està corrent per la paret.

Model de la classe WallRunning:

Atributs de la classe WallRunning:

- **controls:** Atribut que té per referencia els controls del jugador en el comandament de consola.

+ **whatIsWall:** LayerMask que serveix per indicar quins objectes son considerats parets vàlides per fer la carrera de paret. Per fer-ho farem ús de les capes de Unity que ens permet assignar-les als objectes que vulguem.

+ **whatIsGround:** LayerMask que serveix per indicar quins objectes son considerats terra en el nostre escenari de Unity.

+ **wallRunForce:** Variable que serveix per definir la quantitat de força cap endavant que farà que el jugador es pugui moure amb velocitat mentre realitza la carrera de paret.

+ **wallJumpUpForce:** Variable on guardem el valor de la força que es realitzarà al jugador cap amunt per poder realitzar la carrera de paret durant un temps determinat.

+ **wallJumpSideForce:** Variable que consisteix en la força que es realitzarà al jugador en direcció a la paret per així mantenir-se a prop d'aquesta.

+ **wallClimbSpeed:** Valor que guarda la velocitat que tindrà el jugador per baixar al prémer la tecal d'ajupir-se mentre realitza una carrera de paret.

+ **maxWallRunTime:** Atribut on guardarem el temps màxim que el jugador podrà fer una carrera de paret.

- **wallRunTimer:** Comptador que defineix el temps que el jugador porta fent una carrera de paret.

+ **JumpKey:** KeyCode on guardem la tecla de saltar.

+ **downwardsRunKey:** KeyCode on guardarem la tecla que permet desplaçar-se cap avall mentre s'està fent una carrera de paret.

- **downwardsRunning:** Booleà que permet saber si el jugador està desplaçant-se cap avall mentre realitza una carrera de paret.

- **horizontalInput:** Atribut float on hi guardarem els Inputs dels moviments horitzontals dels teclats.

- **verticalInput:** Atribut float on hi guardarem els Inputs dels moviments verticals dels teclats.

+ **wallCheckDistance:** Float que utilitzarem per comprovar la distancia que hi ha del jugador a la paret que vol realitzar una carrera de paret.

+ **minJumpHeight:** Atribut on hi definim la altura del salt mínima per poder-hi fer la carrera de paret.

- **leftWallHit:** Atribut del tipus RaycastHit per detectar si el jugador ha entrat en contacte amb una paret que es troba a l'esquerra.

- **rightWallhit:** Atribut del tipus RaycastHit per detectar si el jugador ha entrat en contacte amb una paret que es troba a la dreta.

- **wallLeft:** Booleà que serveix per saber si la paret que el jugador està tocant es troba a l'esquerra.

- **wallRight:** Booleà que serveix per saber si la paret que el jugador està tocant es troba a la dreta.

- **exitingWall:** Atribut tipus booleà que ens permet identificar si el jugador està acabant de fer la carrera de paret.

+ **exitingWallTime:** Atribut float que ens indica el temps que ha de passar per a que el jugador pugui tornar a efectuar una carrera de paret.

- **exitWallTimer:** Float que utilitzarem per comptar el temps que ha passat des que el jugador ha deixat de realitzar la carrera de paret.

+ **useGravity:** Booleà que ens indicarà si durant la carrera de paret es vol utilitzar la gravetat de l'escena.

+ **gravityCounterForce:** Valor que utilitzarem per contrarestar la gravetat del escenari en cas de que s'utilitzi.

+ **Orientation:** Atribut on hi assignarem cap a on està orientat el jugador.

+ **Cam:** Atribut on hi haurà assignat la càmera del jugador.

- **pm:** Atribut on hi assignem el script PlayerMovementAdvanced que està vinculat al jugador.

- **rb:** Atribut on hi assignarem el rigidbody del jugador.

En la Figura 92 podem veure les propietats públiques assignades al component de la classe WallRunning.

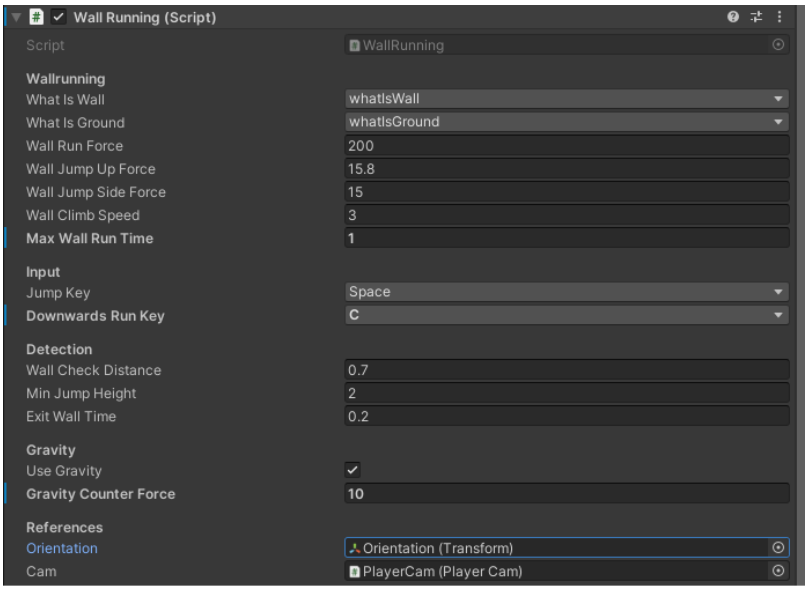


Figura 91: Propietats de la classe WallRunning.

Mètodes de la classe Wallrunning:

- **Awake():** Funció que s'executa un cop al iniciar l'escena. Simplement omple la variable controls.

```
private void Awake()
{
    controls = new PlayerControls();
}
```

- **Start():** Funció que s'executa al primer frame de l'escena. Aquesta funció omple els atributs rb i pm amb els components del jugador corresponents.

```
private void Start()
{
    rb = GetComponent<Rigidbody>();
    pm = GetComponent<PlayerMovementAdvanced>();
}
```

- **Update():** Funció que s'executa a cada frame de l'escena. Crida les funcions ChackForWall() i StateMachine().

```
private void Update()
{
    CheckForWall();
    StateMachine();
}
```

- **FixedUpdate():** Semblant a la funció Update amb la diferència que ofereix un moviment constat i fixe. Comprova si el jugador té l'atribut wallrunning en cert, de ser

```
    private void FixedUpdate()
    {
        if (pm.wallrunning)
            WallRunningMovement();
    }
```

- **CheckForWall():** Funció que s'encarrega de veure a quina direcció està la paret a la que el jugador vol realitzar la carrera de paret mitjançant Raycast.

```
    private void CheckForWall()
    {
        wallRight = Physics.Raycast(transform.position, orientation.right, out
rightWallhit, wallCheckDistance, whatIsWall);
        wallLeft = Physics.Raycast(transform.position, -orientation.right, out
leftWallhit, wallCheckDistance, whatIsWall);
    }
```

- **AboveGround():** Funció que retorna si el jugador està a una distancia del terra lo suficientment alta com per realitzar la carrera de paret.

```
    private bool AboveGround()
    {
        return !Physics.Raycast(transform.position, Vector3.down,
minJumpHeight, whatIsGround);
    }
```

- **StateMachine():** Funció que s'encarrega de gestionar els diferents estats pels que hi passa el jugador a l'hora de realitzar la carrera de paret. Primer agafa els inputs. Acte seguit, segons a quina situació es trobi el jugador (fent la carrera de paret, sortint de la carrera de paret o no està fent res) es cridaran diferents funcions. Per exemple, si el jugador va a realitzar la carrera de paret, cridarà la funció StartWallRun() i la funció WallJump() en cas de que el jugador decideixi saltar. En canvi, si el jugador està sortit de realitzar la carrera de paret, cridarà la funció StopWallRun().

```
    private void StateMachine()
    {
        // Getting Inputs
        horizontalInput = Input.GetAxisRaw("Horizontal");
        verticalInput = Input.GetAxisRaw("Vertical");

        downwardsRunning = Input.GetKey(downwardsRunKey);


        //State 1 - Wallrunning
        if((wallLeft || wallRight) && verticalInput > 0 && AboveGround() &&
!exitingWall)
        {
            if (!pm.wallrunning)
                StartWallRun();

            // wallrun timer
            if (wallRunTimer > 0)
                wallRunTimer -= Time.deltaTime;

            if(wallRunTimer <= 0 && pm.wallrunning)
            {
                exitingWall = true;
```

```
                exitWallTimer = exitWallTime;
            }

            // wall jump
            if (Input.GetKeyDown(JumpKey) ||
controls.Gameplay.Jump.WasPressedThisFrame()) WallJump();
        }

        // State 2 - Exiting

        else if (exitingWall)
        {
            if (pm.wallrunning)
                StopWallRun();
            if (exitWallTimer > 0)
                exitWallTimer -= Time.deltaTime;
            if (exitWallTimer <= 0)
                exitingWall = false;
        }

        // State 3 - None
        else
        {
            if (pm.wallrunning)
                StopWallRun();
        }
    }
```

- **StartWallRun():** Funció que canvia l'estat del jugador a wallrunning, inicialitza el comptador wallRunTimer, elimina la velocitat vertical que hi tenia el jugador i aplica els efectes de càmera per donar una millor sensació de moviment.

```
    private void StartWallRun()
    {
        pm.wallrunning = true;

        wallRunTimer = maxWallRunTime;

        rb.velocity = new Vector3(rb.velocity.x, 0f, rb.velocity.z);


        //Apply camera effects
        cam.DoFov(70f);
        if (wallLeft) cam.DoTilt(-5f);
        if (wallRight) cam.DoTilt(5F);
    }
```

- **WallRunningMovement():** Funció encarregada del moviment i velocitat del jugador mentre està realitzant la carrera de paret. Si la gravetat està activada, aquesta funció també s'encarrega de fer que el jugador vagi baixant poc a poc a mesura que hi va passant el temps per afegir realisme al moviment.

```
    private void WallRunningMovement()
    {
        rb.useGravity = useGravity;


        Vector3 wallNormal = wallRight ? rightWallhit.normal :
leftWallhit.normal;

        Vector3 wallForward = Vector3.Cross(wallNormal, transform.up);
```

```
        if ((orientation.forward - wallForward).magnitude >
(orientation.forward - -wallForward).magnitude)
            wallForward = -wallForward;

        // forward force
        rb.AddForce(wallForward * wallRunForce, ForceMode.Force);

        if (downwardsRunning || controls.Gameplay.Slide.triggered)
        {
            rb.velocity = new Vector3(rb.velocity.x, -wallClimbSpeed,
rb.velocity.z);
        }

        // push to wall force
        if (!(wallLeft && horizontalInput > 0) && !(wallRight &&
horizontalInput < 0))
            rb.AddForce(-wallNormal * 100, ForceMode.Force);

        // weaken gravity
        if (useGravity)
            rb.AddForce(transform.up * gravityCounterForce, ForceMode.Force);
    }
```

- **StopWallRun():** Funció que s'encarrega de treure l'estat Wallrunning del jugador i treu els efectes de càmera posats anteriorment a la funció StartWallrun().

```
    private void StopWallRun()
    {
        pm.wallrunning = false;

        //Reset camera effects
        cam.DoFov(60f);
        cam.DoTilt(0f);
    }
```

- **WallJump():** Funció que permet al jugador realitzar un salt mentre està fent una carrera de paret.

```
    private void WallJump()
    {
        // enter exiting wall state
        exitingWall = true;
        exitWallTimer = exitWallTime;

        Vector3 wallNormal = wallRight ? rightWallhit.normal :
leftWallhit.normal;

        Vector3 forceToApply = transform.up * wallJumpUpForce + wallNormal *
wallJumpSideForce;

        // reset y velocity and add force
        rb.velocity = new Vector3(rb.velocity.x, 0f, rb.velocity.z);
        rb.AddForce(forceToApply, ForceMode.Impulse);
    }
```

- **OnEnable():** Funció creada pel comandament de consola. Permet activar els controls de moviments del comandament.

```
    private void OnEnable()
    {
        controls.Gameplay.Enable();
    }
```

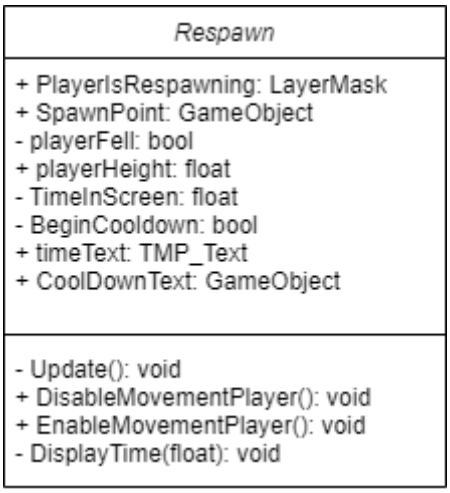
- **OnDisable():** Funció creada pel comandament de consola. Permet desactivar els controls de moviments del comandament.

```
private void OnDisable()
{
    controls.Gameplay.Disable();
}
```

6.5.5 Respawn

La classe Respawn permet al jugador reaparèixer un cop es caigui d'una estructura o no aconsegueixi realitzar un salt a temps. Per realitzar-ho simplement detectem que el jugador entri en contacte amb una superfície que tingui assignada una capa concreta i, si colisiona amb ella, s'activa les funcions d'aquesta classe.

Model de la classe Respawn:



Atributs de la classe Respawn:

+ **PlayerIsRespawning:** LayerMask que utilitzem per identificar quins objectes haurà de tocar el jugador per que hi pugui reapareixer al punt d'aparició de l'escenari.

+ **SpawnPoint:** Objecte del qual utilitzarem la seva posició per determinar a on apareixerà el jugador.

- **playerFell:** Booleà per determinar si el jugador ha caigut d'una plataforma.

+ **playerHeight:** Float on guardem la mida del jugador.

- **TimeInScreen:** Float on hi guardem el temps d'espera per a que el jugador pugui tornar a jugar.

- **BeginCooldown:** Booleà que ens permet saber quan comença el comptador per a que el jugador es pugui tornar a moure.

+ **timeText:** Text que agafarem del Canvas CoolDownRespawn (explicat en més detall a l'Apartat 6.2.5) per mostrar el comptador per pantalla.

+ **CoolDownText:** Objecte on hi està inclòs el Canvas CoolDownRespawn.

En la Figura 93 podem veure les propietats públiques assignades al component de la classe Respawn.

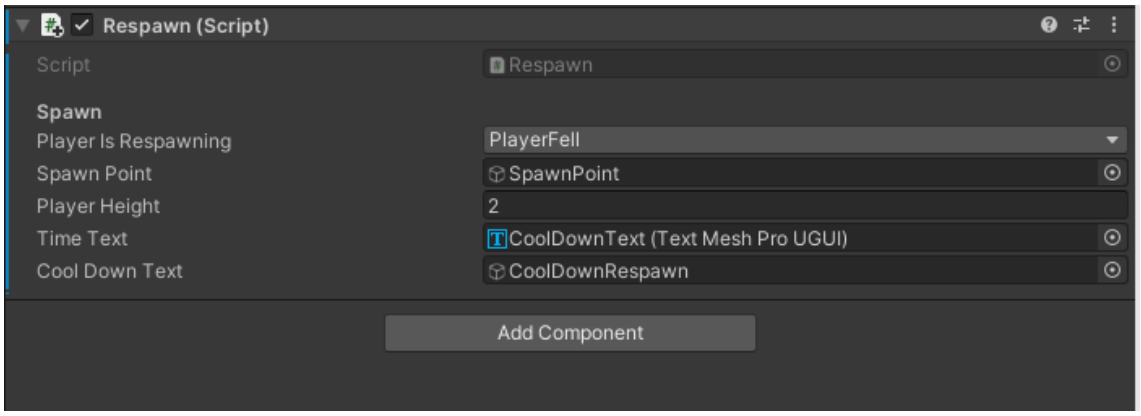


Figura 93: Propietats de la classe Respawn.

Mètodes de la classe Respawn:

- **Update():** Funció que s'executa a cada frame de l'escena. Primer detecta si el jugador toca una superfície amb la capa definida en PlayerIsRespawning. En cas de ser cert, mou el jugador a la posició de l'objecte SpawnPoint, crida la funció DisableMovementPlayer() i inicia la compta enrere. Un cop el comptador arriba a 0, es crida la funció EnableMovementPlayer().

```
void Update()
    {
        playerFell = Physics.Raycast(transform.position, Vector3.down,
playerHeight * 0.5f + 0.2f, PlayerIsRespawning);
        if (playerFell)
        {
            TimeInScreen = 3.0f;
            transform.position = SpawnPoint.transform.position;
            DisableMovementPlayer();
            CoolDownText.SetActive(true);
            BeginCooldown = true;

        }
        if(BeginCooldown)
        {
            DisplayTime(TimeInScreen);
            Physics.gravity = new Vector3(0,0,0);
            if(TimeInScreen > 0)
            {
                TimeInScreen -= Time.deltaTime;
```

```
            }
            else
            {
                Physics.gravity = new Vector3(0,-9.81f, 0);
                CoolDownText.SetActive(false);
                EnableMovementPlayer();
                BeginCooldown = false;
            }
        }


    }
```

+ **DisableMovementPlayer():** Aquesta funció canvia el estat del jugador a Freeze, impedint al jugador moure's.

```
    public void DisableMovementPlayer()
    {

        GetComponent<PlayerMovementAdvanced>().freeze = true;

    }
```

+ **EnableMovementPlayer():** Fucnió que treu al jugador del estat Freeze, permetent el moviment un altre cop.

```
    public void EnableMovementPlayer()
    {
        GetComponent<PlayerMovementAdvanced>().freeze = false;

    }
```

- **DisplayTime(TimeInScreen):** Funció que mostra el comptador donat per pantalla.

```
    private void DisplayTime(float TimeInScreen)
    {
        TimeInScreen += 1;
        int SecondsCooldown = (int)TimeInScreen;

        timeText.text = SecondsCooldown.ToString();
    }
```

6.5.6 Grappling

Classe que gestiona la mecànica d'enganxar-se. Donat un objecte, hi apareix una petita esfera que el jugador podrà utilitzar per apuntar a la zona que es vulgui enganxar. Un cop el jugador ho té clar, pot prémer la tecla adequada per impulsar-se cap al objectiu i arribar sense cap problema al destí marcat.

Model de la classe Grappling:

Atributs de la classe Grappling:

- **controls:** Atribut que té per referencia els controls del jugador en el comandament de consola.

+ **RopeSound:** Pista de so que s'utilitza per afegir un efecte al utilitzar la tecla d'enganxar-se.

- **pm:** Atribut on hi assignem el script PlayerMovementAdvanced que està vinculat al jugador.

+ **cam:** Atribut on hi haurà assignat la càmera del jugador.

+ **gunTip:** Objecte que utilitzarem per decidir la posició per la qual surt la corda al enganxar-se.

+ **whatIsGrappleable:** LayerMask que utilitzarem per definir quins objectes poden ser utilitzats pel jugador alhora d'enganxar-se.

+ **lr:** LinerRenderer que utilitzarem per dibuixar la corda de la mecànica d'enganxar-se.

+ **maxGrappleDistance:** Float on hi guardem la distància màxima que el jugador podrà utilitzar la mecànica d'enganxar-se.

+ **grappleDelayTime:** Valor que utilitzarem per definir el temps que el jugador haurà d'esperar per poder tornar a utilitzar la mecànica d'enganxar-se.

+ **overshootYAxis:** Variable que utilitzarem per calcular el punt més alt en l'arc que realitza el jugador al fer la mecànica d'enganxar-se.

+ **grapplePoint:** Vector on hi guardarem el punt exacte on el jugador ha volgut enganxar-se.

+ **grapplingCd:** Float que utilitzarem per establir el temps que ha de passar el jugador sense poder tornar a enganxar-se.

+ **grapplingCdTimer:** Float que ens servirà per saber quan podrà el jugador tornar a utilitzar la habilitat d'enganxar-se.

+ **grappleKey:** KeyCode on li assignarem la tecla del teclat per que el jugador pugui enganxar-se.

+ **predictionHit:** RayCastHit que ajudarà a saber a quin punt el jugador està mirant per enganxar-se.

+ **predictionSphereCastRadius:** Radi de l'esfera que utilitzarem per a que el jugador pugui apuntar amb facilitat a l'hora d'escollir cap a on enganxar-se.

+ **predictionPoint:** Objecte que mourem depenent de a on el jugador estigui apuntant, ajudant-lo a saber cap a on anirà un cop decideixi enganxar-se.

+ **grappling:** Booleà que ens indica si el jugador està executat la mecànica o no.

En la Figura 94 podem veure les propietats públiques assignades al component de la classe Grappling.

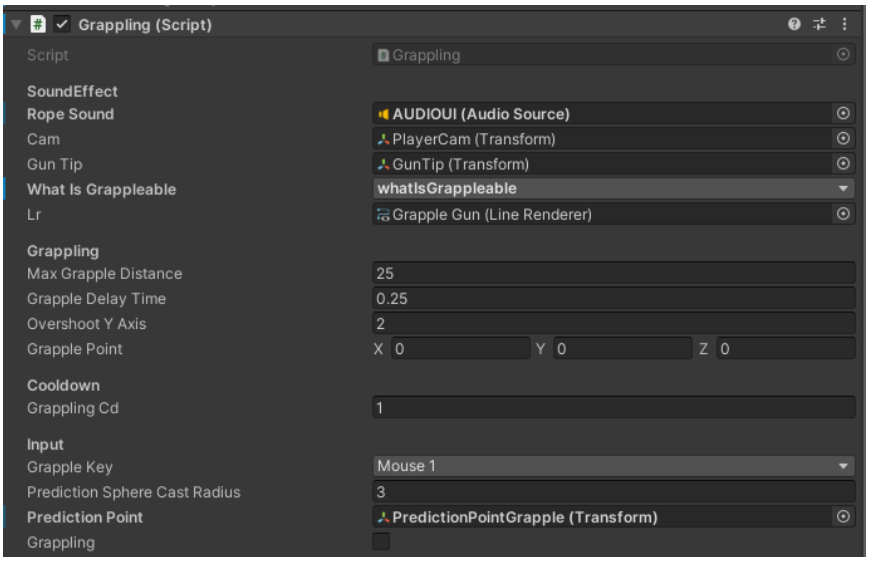


Figura 94: Propietats de la classe Grappling.

Mètodes de la classe Grappling:

- **Awake():** Funció que s'executa un cop al iniciar l'escena. Simplement omple la variable controls.

```
    private void Awake()
    {
        controls = new PlayerControls();
    }
```

- **Start():** Funció que s'executa al començament de l'escena. Simplement omple l'atribut pm amb el script PlayerMovementAdvanced vinculat al jugador.

```
    private void Start()
    {
        pm = GetComponent<PlayerMovementAdvanced>();
    }
```

- **Update():** Funció que s'executa a cada frame de l'escena. Aquesta funció crida constantment a CheckForGrapplePoints(), també s'encarrega de cridar el StartGrapple() un cop es premi un dels botons per executar la mecànica d'enganxar-se.

```
    private void Update()
    {
        if (Input.GetKeyDown(grappleKey) || controls.Gameplay.Grapple.WasPressedThisFrame()) {
StartGrapple(); }
        CheckForGrapplePoints();
        if (grapplingCdTimer > 0)
            grapplingCdTimer -= Time.fixedDeltaTime;
    }
```

- **StartGrapple():** Funció que inicia el moviment d'enganxar-se (només si el comptador grapplingCdTimer s'ha esgotat). Manté l'estat del jugador en Freeze i llença la corda cap al objecte apuntat (reproduint l'efecte de so). Si el jugador aconsegueix donar a un objecte vàlid, aquesta funció crida a l'altre anomenada ExecuteGrapple(), en cas contrari, crida a la funció StopGrapple(). En aquesta funció també eliminem momentàniament l'opció del jugador de poder lliscar o tornar-se a enganxar ja que pot crear conflictes en el codi.

```
private void StartGrapple()
{
    if (grapplingCdTimer > 0) return;


    GetComponent<Swing>().StopSwing();
    if(GetComponent<PlayerMovementAdvanced>().sliding)
    {
        GetComponent<Sliding>().StopSlide();
    }
    grappling = true;
    pm.ResetRestrictions();
    pm.freeze = true;

    grappleKey = KeyCode.None;
    controls.Gameplay.Grapple.Disable();
    GetComponent<Sliding>().slideKey = KeyCode.None;
    controls.Gameplay.Slide.Disable();
    RopeSound.Play();
    if (predictionHit.point != Vector3.zero)
    {
        grapplePoint = predictionHit.point;

        Invoke(nameof(ExecuteGrapple), grappleDelayTime);
    }
    else
    {
        grapplePoint = cam.position + cam.forward * maxGrappleDistance;

        Invoke(nameof(StopGrapple), grappleDelayTime);
    }
    lr.enabled = true;

}
```

- **ExecuteGrapple():** Funció que gestiona tots els càlculs per impulsar al jugador a la posició desitjada mitjançant la Funció JumpToPosition(grapplePoint, HighestPointOnArc) de la classe PlayerMovementAdvanced.

```
    private void ExecuteGrapple()
    {

        grappleKey = KeyCode.Mouse1;
        controls.Gameplay.Grapple.Enable();
        GetComponent<Sliding>().slideKey = KeyCode.LeftControl;
        controls.Gameplay.Slide.Enable();
        pm.freeze = false;

        Vector3 lowestPoint = new Vector3(transform.position.x, transform.position.y - 1f,
transform.position.z);

        float grapplePointRelativeYPos = grapplePoint.y - lowestPoint.y;
        float highestPointOnArc = grapplePointRelativeYPos + overshootYAxis;

        if (grapplePointRelativeYPos < 0) highestPointOnArc = overshootYAxis;

        pm.JumpToPosition(grapplePoint, highestPointOnArc);

        Invoke(nameof(StopGrapple), 1.0f);
    }
```

+ **StopGrapple():** Funció que finalitza la mecànica d'enganxar-se. Torna a donar al jugador l'opció de lliscar, canvia el seu estat i restaura el comptador al valor inicial entre altres coses.

```
    public void StopGrapple()
    {
        grappleKey = KeyCode.Mouse1;
        GetComponent<Sliding>().slideKey = KeyCode.LeftControl;
        controls.Gameplay.Grapple.Enable();
        controls.Gameplay.Slide.Enable();

        pm.freeze = false;

        grappling = false;

        grapplingCdTimer = grapplingCd;

        lr.enabled = false;

    }
```

- **CheckForGrapplePoint():** Funció que s'encarrega de comprovar els punts en el quals el jugador podria utilitzar la mecànica d'enganxar-se.

```
private void CheckForGrapplePoints()
    {
        RaycastHit sphereCastHit;
        Physics.SphereCast(cam.position, predictionSphereCastRadius, cam.forward, out
sphereCastHit, maxGrappleDistance, whatIsGrappleable);
        RaycastHit raycastHit;
        Physics.Raycast(cam.position, cam.forward, out raycastHit, maxGrappleDistance,
whatIsGrappleable);
        Vector3 realHitPoint;

        //Option 1 - Direct hit
        if (raycastHit.point != Vector3.zero)
            realHitPoint = raycastHit.point;

        //Option 2 - Indirect (predicted) Hit
        else if (sphereCastHit.point != Vector3.zero)
```

```
            realHitPoint = sphereCastHit.point;

        //Option 3 - Miss
        else
            realHitPoint = Vector3.zero;

        // realHitPoint found
        if (realHitPoint != Vector3.zero)
        {
            predictionPoint.gameObject.SetActive(true);
            predictionPoint.position = realHitPoint;
        }
        // realHitPoint not found
        else
        {
            predictionPoint.gameObject.SetActive(false);

        }

        predictionHit = raycastHit.point == Vector3.zero ? sphereCastHit : raycastHit;
    }
```

- **OnEnable():** Funció creada pel comandament de consola. Permet activar els controls de moviments del comandament.

```
    private void OnEnable()
    {
        controls.Gameplay.Enable();
    }
```

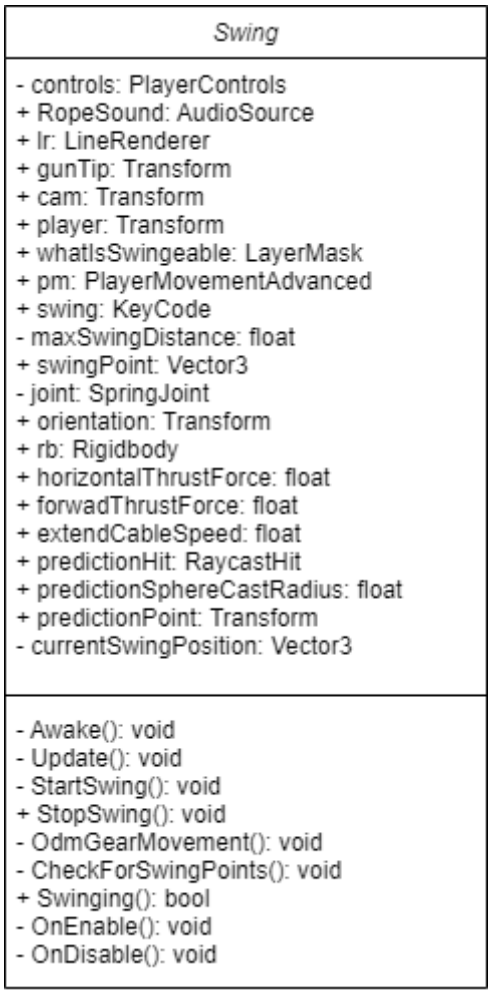
- **OnDisable():** Funció creada pel comandament de consola. Permet desactivar els controls de moviments del comandament.

```
    private void OnDisable()
    {
        controls.Gameplay.Disable();
    }
```

6.5.7 Swing

Classe que gestiona la mecànica de gronxar-se. Donat un objecte, hi apareix una petita esfera que el jugador podrà utilitzar per apuntar a la zona que vulgui. Mentre el jugador mantingui premuda la tecla adequada, podrà gronxar-se el temps que vulgui. També s'hi ha afegit un sistema de desplaçament millorat mentre el jugador està penjant per així facilitar la maniobrabilitat en aquestes circumstàncies.

Model de la classe Swing:



Atributs de la classe Swing:

- **controls:** Atribut que té per referencia els controls del jugador en el comandament de consola.

+ **RopeSound:** Pista de so que s'utilitza per afegir un efecte al utilitzar la tecla de gronxar-se.

+ **lr:** LinerRenderer que utilitzarem per dibuixar la corda de la mecànica de gronxar-se.

+ **gunTip:** Objecte que utilitzarem per decidir la posició per la qual surt la corda al gronxar-se.

+ **cam:** Atribut on hi haurà assignat la posició de la càmera del jugador.

+ **Player:** valor on guardarem la posició en la que es troba el jugador.

+ **whatIsSwingeable:** LayerMask que utilitzarem per definir quins objectes poden ser utilitzats pel jugador alhora d'enganxar-se.

+ **pm:** Atribut on hi assignem el script PlayerMovementAdvanced que està vinculat al jugador.

+ **swing:** KeyCode on li assignarem la tecla del teclat per que el jugador pugui gronxar-se.

+ **maxSwingDistance:** Float on hi guardem la distància màxima que el jugador podrà utilitzar la mecànica de gronxar-se.

+ **swingPoint:** Vector on hi guardarem el punt exacte on el jugador ha volgut gronxar.

- **Joint:** Component Spring Joint que permet que dos Rigidbodies, el del jugador i l'objecte, es mantinguin units però amb la possibilitat de que la distancia entre ells variï.

+ **Orientation:** Component que indica cap a on està orientat el jugador.

+ **rb:** Atribut on hi assignarem el rigidbody del jugador.

+ **horizontralThrustForce:** Valor on hi guardarem la quantitat de força que hi voldrem aplicar al jugador quan es vulgui moure cap els costats mentre s'està gronxant.

+ **forwardThrustForce:** Valor on hi guardarem la quantitat de força que hi voldrem aplicar al jugador quan es vulgui moure cap endavant mentre s'està gronxant.

+ **extendCableSpeed:** Valor que servirà per definir a quina velocitat volem que la corda s'allargui quan el jugador premi la tecla corresponent.

+ **predictionHit:** RayCastHit que ens ajudarà a saber a quin punt el jugador està mirant per gronxar-se.

+ **predictionSphereCastRadius:** Radi de l'esfera que utilitzarem per a que el jugador pugui apuntar amb facilitat a l'hora d'escollir cap a on gronxar-se.

+ **predictionPoint:** Objecte que mourem depenent de a on el jugador estigui apuntant, ajudant-lo a saber cap a on anirà un cop decideixi gronxar.

- **currentSwingPosition:** Vector on hi guardarem la posició del ganxo que utilitzarem per gronxar-nos.

En la Figura 95 podem veure les propietats públiques assignades al component de la classe Swing.

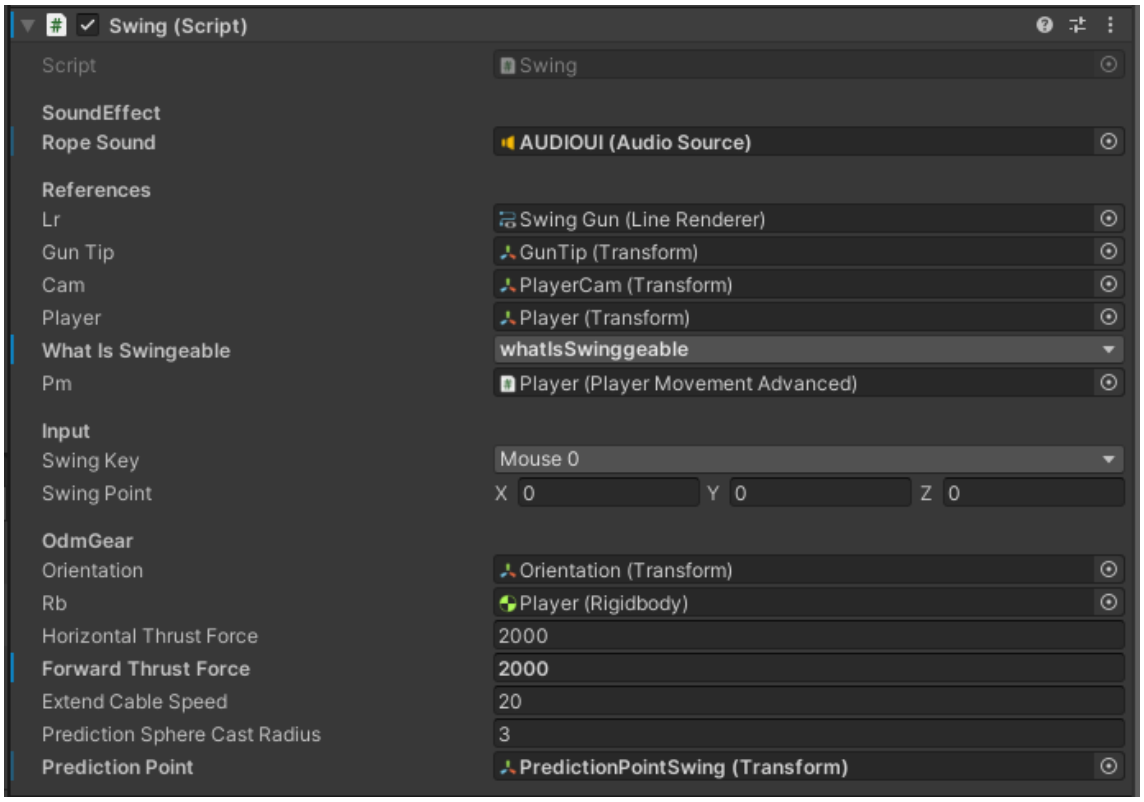


Figura 95: Propietats de la classe Swing.

Mètodes de la classe Swing:

- **Awake():** Funció que s'executa un cop al iniciar l'escena. Simplement omple la variable controls.

```
    private void Awake()
    {
        controls = new PlayerControls();
    }
```

- **Update():** Funció que s'executa a cada frame de l'escena. Consisteix en cridar constantment la funció CheckForSwingPoints() i, si es prem (i es manté) la tecla de gronxar-se, crida la funció StartSwing(). Un cop deixa de prémer la tecla, es crida la funció de StopSwing(). Aquesta funció també crida constantment la funció OdmGearMovement() sempre i quan el jugador estigui gronxant-se.

```
    void Update()
    {
        if (Input.GetKeyDown(swingKey) || controls.Gameplay.Swing.WasPressedThisFrame())
StartSwing();
        if (Input.GetKeyUp(swingKey) || controls.Gameplay.Swing.WasReleasedThisFrame())
StopSwing();

        CheckForSwingPoints();

        if (joint != null) OdmGearMovement();
    }
```

- **StartSwing():** Aquesta funció s'encarrega d'inicialitzar la mecànica de gronxar-se. A més de reproduir l'efecte de so definit anteriorment, dibuixar la corda i canviar l'estat del jugador a Swinging, si el jugador troba un objecte vàlid, s'afegeix un component al jugador anomenat SpringJoint per realitzar la unió entre aquest i l'objecte desitjat. Un cop creat el component es defineixen les seves variables a més de calcular la distancia que hi ha entre el jugador i el punt d'encolatge.

```
private void StartSwing()
{

    if (predictionHit.point == Vector3.zero) return;

    if(GetComponent<Grappling>()!=null)
        GetComponent<Grappling>().StopGrapple();
    pm.ResetRestrictions();

    pm.swinging = true;
    RopeSound.Play();
    swingPoint = predictionHit.point;
    joint = player.gameObject.AddComponent<SpringJoint>();
    joint.autoConfigureConnectedAnchor = false;
    joint.connectedAnchor = swingPoint;

    float distanceFromPoint = Vector3.Distance(player.position, swingPoint);

    joint.maxDistance = distanceFromPoint * 0.8f;
    joint.minDistance = distanceFromPoint * 0.25f;

    joint.spring = 4.5f;
    joint.damper = 7f;
    joint.massScale = 4.5f;

    currentSwingPosition = gunTip.position;
    lr.enabled = true;

}
```

+ **StopSwing():** Funció que destrueix el component SpringJoint i elimina l'estat swinging del jugador.

```
public void StopSwing()
{
    pm.swinging = false;

    Destroy(joint);
    lr.enabled = false;
}
```

- **OdmGearMovement():** Funció que s'encarrega del moviment del jugador mentre s'està gronxant, fent que sigui més fluït i còmode per l'usuari.

```
    private void OdmGearMovement()
    {
        if (Input.GetKey(KeyCode.D) || controls.Gameplay.SwingR.IsPressed())
rb.AddForce(orientation.right * horizontalThrustForce * Time.deltaTime);

        if (Input.GetKey(KeyCode.A) || controls.Gameplay.SwingL.IsPressed()) rb.AddForce(-
orientation.right * horizontalThrustForce * Time.deltaTime);

        if (Input.GetKey(KeyCode.W) || controls.Gameplay.SwingF.IsPressed())
rb.AddForce(orientation.forward * forwardThrustForce * Time.deltaTime);

        if(Input.GetKey(KeyCode.Space) || controls.Gameplay.Jump.IsPressed())
        {
            Vector3 directionToPoint = swingPoint - transform.position;
            rb.AddForce(directionToPoint.normalized * forwardThrustForce * Time.deltaTime);

            float distanceFromPoint = Vector3.Distance(transform.position, swingPoint);

            joint.maxDistance = distanceFromPoint * 0.8f;
            joint.minDistance = distanceFromPoint * 0.25f;

        }

        if(Input.GetKey(KeyCode.S) || controls.Gameplay.SwingB.IsPressed())
        {
            float extendedDistanceFromPoint = Vector3.Distance(transform.position, swingPoint)
+ extendCableSpeed;

            joint.maxDistance = extendedDistanceFromPoint * 0.8f;
            joint.minDistance = extendedDistanceFromPoint * 0.25f;
        }
    }
```

- **CheckForSwingPoints():** Funció que s'encarrega de comprovar els punts que el jugador es pot gronxar en l'escenari.

```
private void CheckForSwingPoints()
    {
        if (joint != null) return;

        RaycastHit sphereCastHit;
        Physics.SphereCast(cam.position, predictionSphereCastRadius, cam.forward, out
sphereCastHit, maxSwingDistance, whatIsSwingeable);

        RaycastHit raycastHit;
        Physics.Raycast(cam.position, cam.forward, out raycastHit, maxSwingDistance,
whatIsSwingeable);

        Vector3 realHitPoint;

        if (raycastHit.point != Vector3.zero)
            realHitPoint = raycastHit.point;

        else if (sphereCastHit.point != Vector3.zero)
            realHitPoint = sphereCastHit.point;
```

```
        else if (sphereCastHit.point != Vector3.zero)
            realHitPoint = sphereCastHit.point;

        else
            realHitPoint = Vector3.zero;

        if(realHitPoint!=Vector3.zero)
        {
            predictionPoint.gameObject.SetActive(true);
            predictionPoint.position = realHitPoint;
        }
        else
        {
            predictionPoint.gameObject.SetActive(false);
        }

        predictionHit = raycastHit.point == Vector3.zero ? sphereCastHit : raycastHit;
    }
```

+ **Swinging():** Funció que retorna si el jugador té l'estat swinging o no.

```
    public bool swinging()
    {
        return pm.swinging;
    }
```

+ **OnEnable():** Funció creada pel comandament de consola. Permet activar els controls de moviments del comandament.

```
    private void OnEnable()
    {
        controls.Gameplay.Enable();
    }
```

+ **OnDisable():** Funció creada pel comandament de consola. Permet desactivar els controls de moviments del comandament.

```
    private void OnDisable()
    {
        controls.Gameplay.Disable();
    }
```

6.5.8 Grappling/Swinging Rope

Finalment, les classes Grappling Rope i Swinging Rope són les encarregades d'aplicar els efectes de les cordes cada cop que el jugador vulgui executar les mecàniques de gronxar-se o enganxar-se.

Per fer-ho, primer vam afegir un component LineRenderer als objectes Grapple Gun i Swing Gun. Aquest component ens serveix per dibuixar una línia que utilitzarem a la classe per simular un efecte similar a una corda. En la Figura 96 podem observar els detall d'aquest component en els dos objectes.

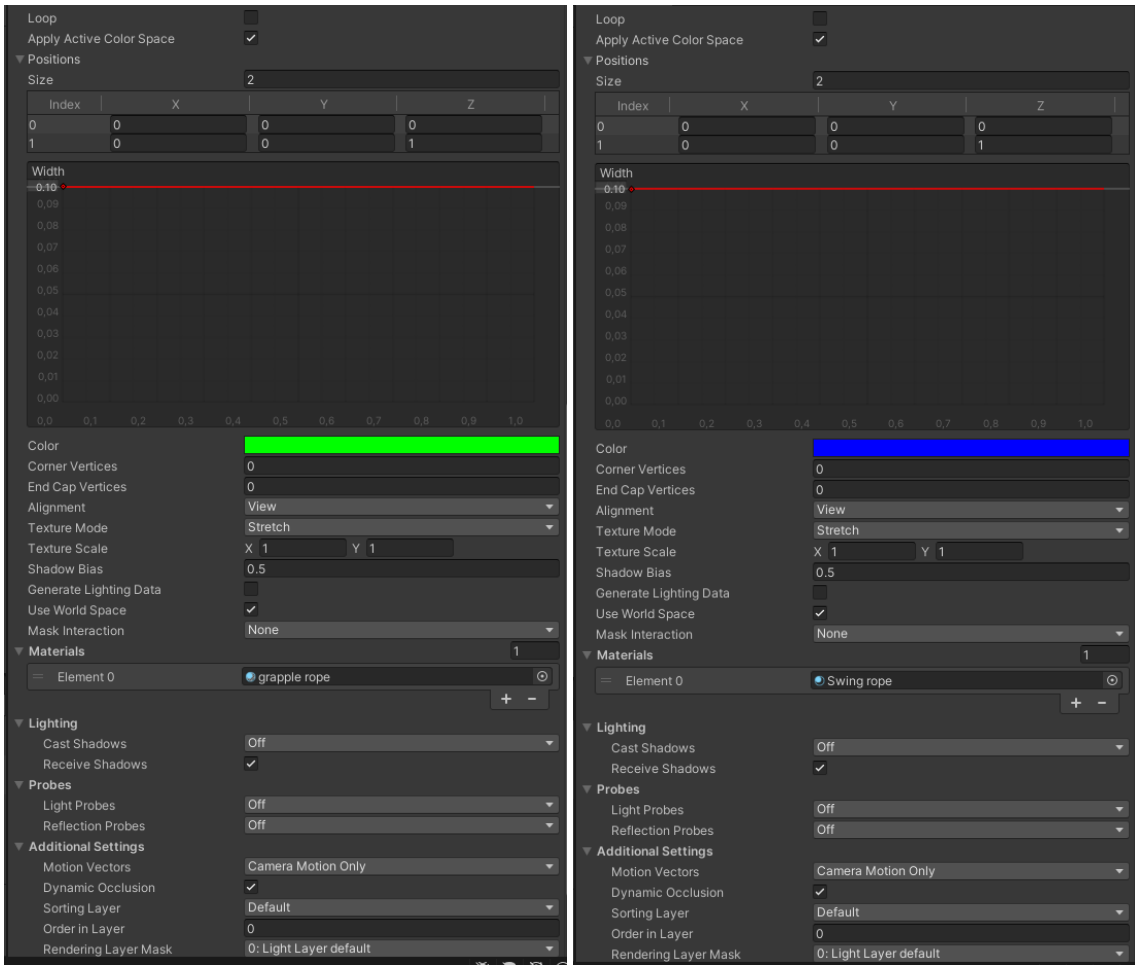
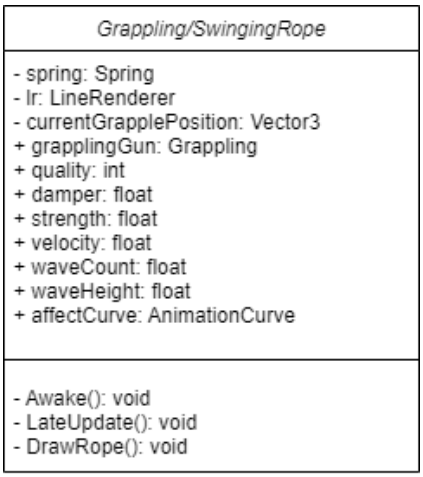


Figura 96: Propietats del component Line Renderer tant a Grapple Gun com a Swing Gun.

En el projecte, existeixen dues classes per simular l'efecte de la corda: SwingingRope i GrappleRope. La raó d'això és que, durant la fase d'implementació, si utilitzàvem una classe per les dues cordes, donava bastants problemes (l'efecte de la corda no es veia bé, a vegades s'ajuntaven els dos efectes, etc.). La nostra solució va ser duplicar el mateix codi però amb noms diferents per a que el motor els diferenciés. Per aquesta raó, només explicarem una de les classes, ja que l'altre classe és una copia exacta d'aquesta.

Model de les classes GrappleRope/SwingingRope:



Atributs de les classes GrappleRope/SwingingRope:

- **Spring:** Variable que utilitzarem per donar l'efecte de ressort.

- **lr:** LinerRenderer que utilitzarem per dibuixar la corda.

- **currentGrapplePosition:** Vector on hi guardarem la posició actual per a on sortirà la corda tant de la mecànica d'enganxar-se com la de gronxar-se.

+ **GrapplingGun:** Atribut que s'associa al script Grappling o Swing per agafar la posició d'on sortirà la corda (és l'única variable diferent dels dos scripts).

+ **quality:** Atribut que defineix el valor de la qualitat de la corda.

+ **damper:** Atribut on hi guardarem la quantitat a la qual el ressort es reduït quan està actiu.

+ **strength:** Valor on definirem la força de la variable Spring.

+ **velocity:** Atribut que defineix la velocitat a la que s'activa el spring (com de ràpid es mourà l'animació).

+ **waveCount:** Nombre float on guardarem la quantitat d'ones que mostrarà l'efecte de corda.

+ **waveHeight:** ombre que determinarà la mida de les ones de la corda.

+ **affectCurve:** Animació que farà la corda un cop s'executi.

En la Figura 96 podem veure les propietats públiques assignades als components de les classes GrappleRope i SwingingRope.

Figura 96: Propietats de les classes SwingRope i GrapplingRope.

Mètodes de les classes GrappleRope/SwingingRope:

- **Awake():** Funció que s'executa un cop al iniciar l'escena. Agafa el component LineRenderer i s'omple la variable spring.

```
private void Awake()
{
    lr = GetComponent<LineRenderer>();
    spring = new Spring();
    spring.SetTarget(0);
}
```

- **LateUpdate():** Funció que s'executa després de que la resta de funcions Update() s'hagin ejecutat. Crida la funció DrawRope().

```
private void LateUpdate()
{
    DrawRope();
}
```

- **DrawRope():** Funció que s'encarrega de dibuixar la corda segons els atributs establerts a l'inici.

```
void DrawRope()
{
    if (!grapplingGun.swinging())
    {
        currentGrapplePosition = grapplingGun.gunTip.position;
        spring.Reset();
        if (lr.positionCount > 0)
```

```
lr.positionCount = 0;
            return;
        }

        if (lr.positionCount == 0)
        {
            spring.SetVelocity(velocity);
            lr.positionCount = quality + 1;
        }
        spring.SetDamper(damper);
        spring.SetStength(strength);
        spring.Update(Time.deltaTime);

        var grapplePoint = grapplingGun.swingPoint;
        var gunTipPosition = grapplingGun.gunTip.position;
        var up = Quaternion.LookRotation((grapplePoint -
gunTipPosition).normalized)*Vector3.up;


        currentGrapplePosition = Vector3.Lerp(currentGrapplePosition, grapplePoint,
Time.deltaTime * 12f);

        for(int i = 0; i < quality + 1; i++)
        {
            var delta = i / (float)quality;
            var offset = up * waveHeight * Mathf.Sin(delta * waveCount * Mathf.PI) *
spring.Value * affectCurve.Evaluate(delta);
            lr.SetPosition(i, Vector3.Lerp(gunTipPosition, currentGrapplePosition, delta) +
offset);
        }
    }
```

6.6 Proves

Durant el desenvolupament del joc, hem anat realitzant de forma contínua diverses proves per garantir el correcte funcionament de les mecàniques que hem implementat. A més, hem estat realitzant diverses proves de disseny del mapa per garantir la millor experiència del jugador.

Durant la programació del joc, amb l'objectiu de verificar la correcta detecció d'alguns objectes així com el correcte funcionament d'algunes mecàniques, hem utilitzat vàries tècniques. Encara que la més habitual ha sigut mitjançant el mòdul "Debug". Aquest mòdul ens proporciona un parell de funcions que ajuden bastant a l'hora de monitorar els valors de les variables i el seu correcte funcionament. També vam utilitzar algunes variables de caràcter públic per monitorar exactament els canvis pels quals hi passava durant els nivells.

Finalment, un altre recurs que va ser molt útil, sobretot a l'hora de provar algunes mecàniques, va ser la creació d'una zona de proves dintre d'una escena al Unity (veure Figura 97). D'aquesta manera vam poder estudiar i calcular tant les distàncies com els valors idonis per garantir una experiència satisfactòria pel jugador.

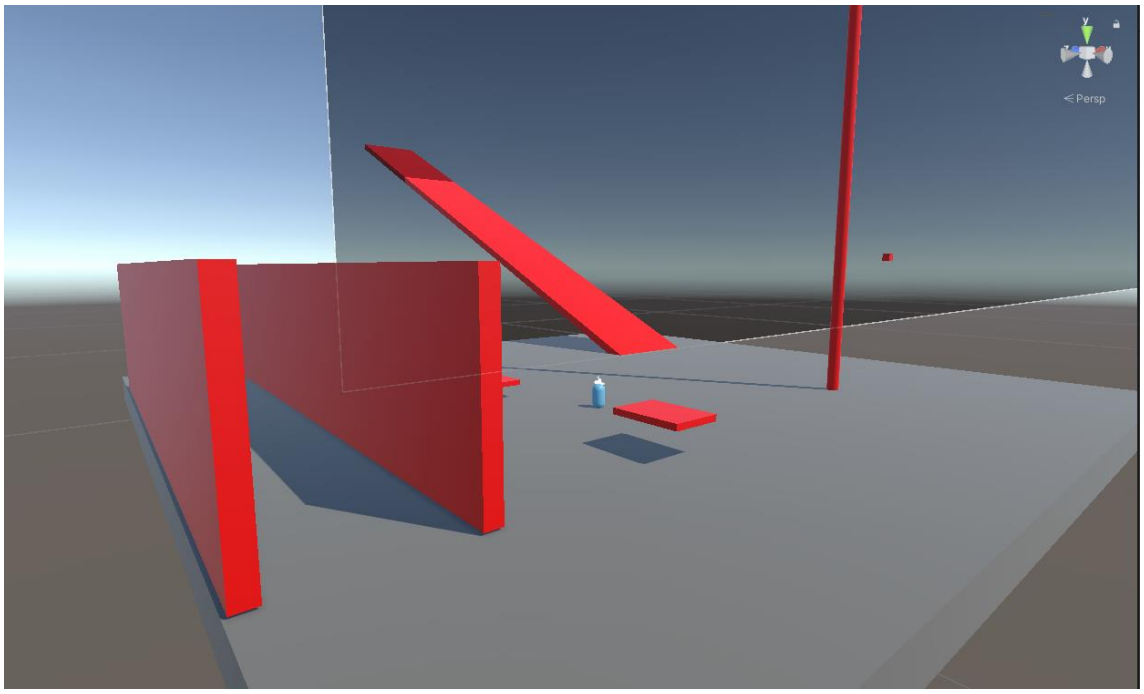


Figura 97: Nivell de proves que es va utilitzar per comprovar el funcionament de les mecàniques dissenyades pel videojoc.

7. Resultats

7.1 Legislació i normativa vigent

El joc desenvolupat no presenta cap problema en aspectes legislatius. En cap moment del videojoc guardem informació de caràcter personal del jugador, per tant la Llei Orgànica de Protecció de Dades (LOPD) no ens aplicaria em cap moment. Al no constituir cap activitat de caràcter econòmic, tampoc ens aplicaria la Llei de Serveis de la Societat de la Informació i Comerç Electrònic.

Pel que fa al tema del Copyright, a excepció d'alguns sons d'ús lliure, el joc ha estat desenvolupat per nosaltres en la seva totalitat, per tant no ens hauria de suposar cap problema a l'hora de comercialitzar-lo. Tot i així, si el producte fos un èxit i es superés cert llindar definit pels directors de Unity, ens veuríem amb l'obligació de comprar llicències de caràcter professional per evitar problemes legals a futur.

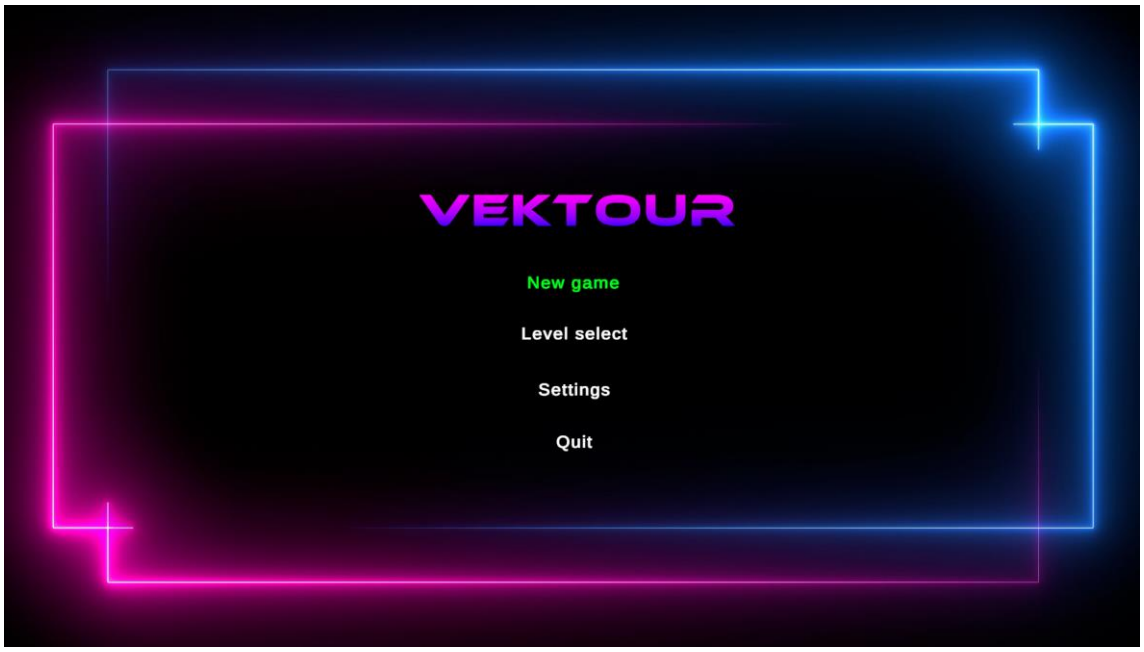
7.2 Pegi

El sistema PEGI (Pan European Game Information) és el mecanisme d'autoregulació dissenyat per la indústria amb l'objectiu de dotar els seus productes d'informació orientativa sobre l'edat adequada per consumir-los. D'aquesta manera, s'evita en gran mesura els error que hi poden haver sobre la interpretació d'allò que és apte per a cada consumidor, ja que això li permet comprendre els tipus de continguts que es pot trobar i realitzar una elecció lliure i informada.

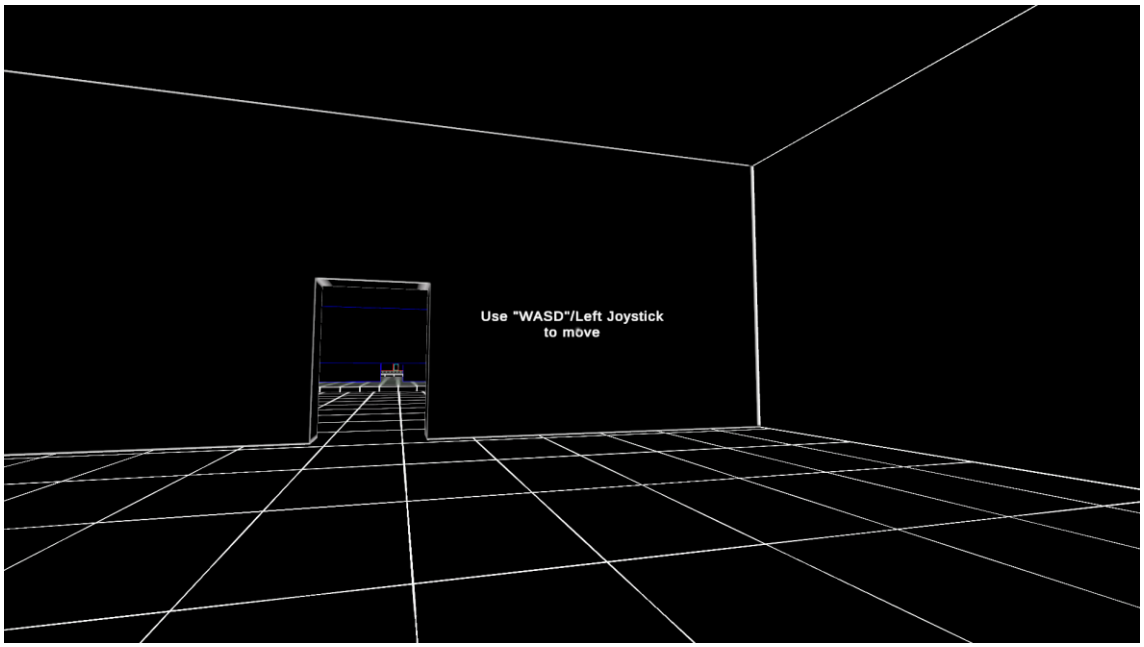
En cas del projecte desenvolupat, l'etiqueta d'edat haurà de ser la PEGI 16, sense cap altre etiqueta addicional. Això és deu al nivell de dificultat del joc i a la necessitat de certa coherència per entendre certs temes complexos com la identitat o el que significar estar viu.

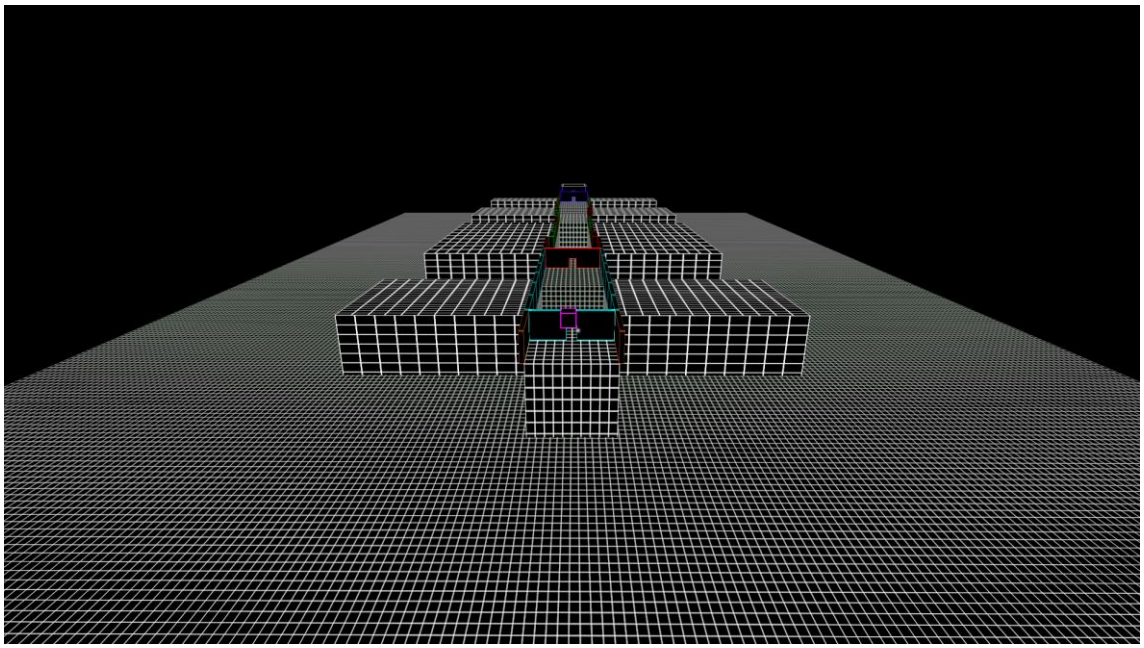
7.3 Resultat final

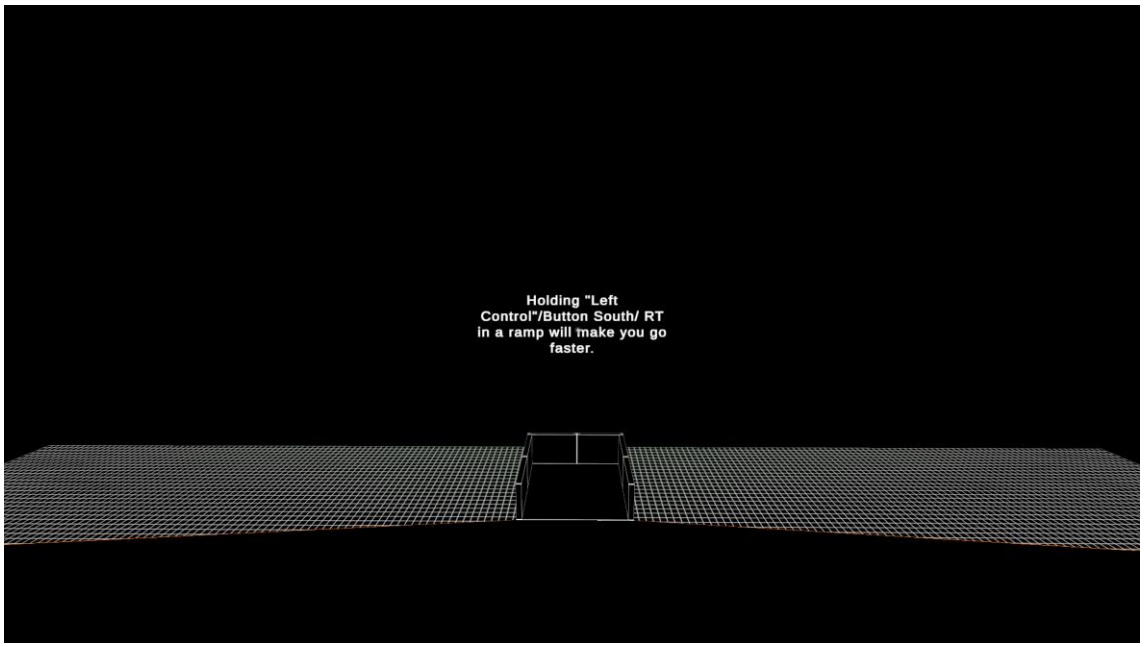
En aquest apartat, es presenten imatges que reflecteixen el resultat final del joc.


VEKTOUR
New game
Level select
Settings
Quit

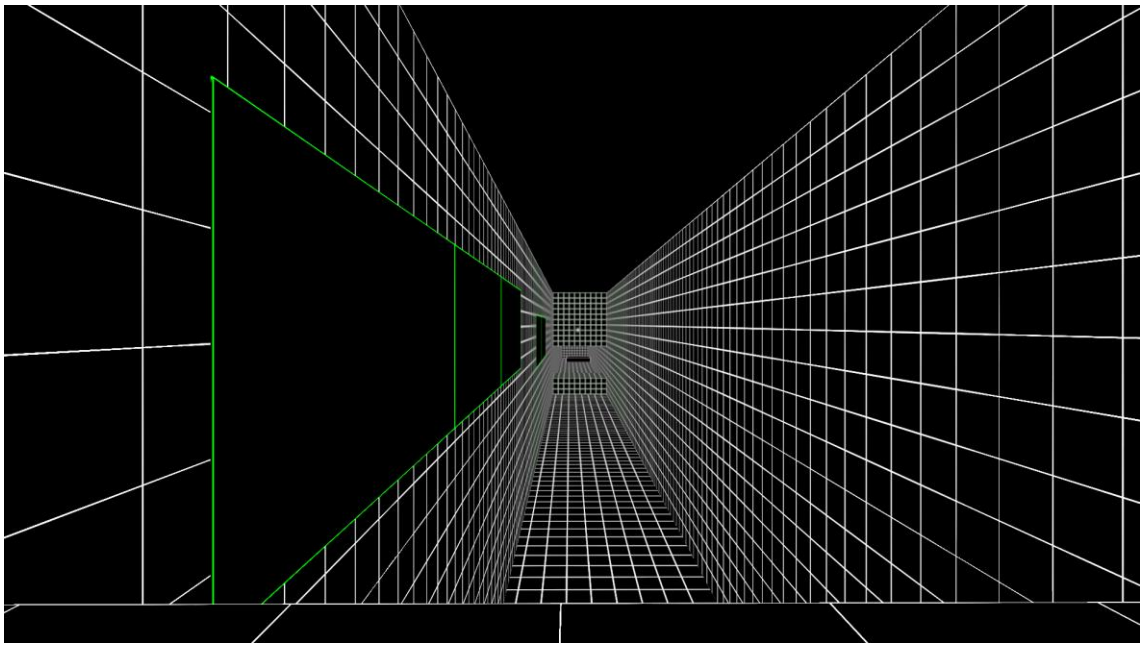

VEKTOUR
Select Level
Level 1
???
???
???
???
???
???
???
???
???
Level 10
Return

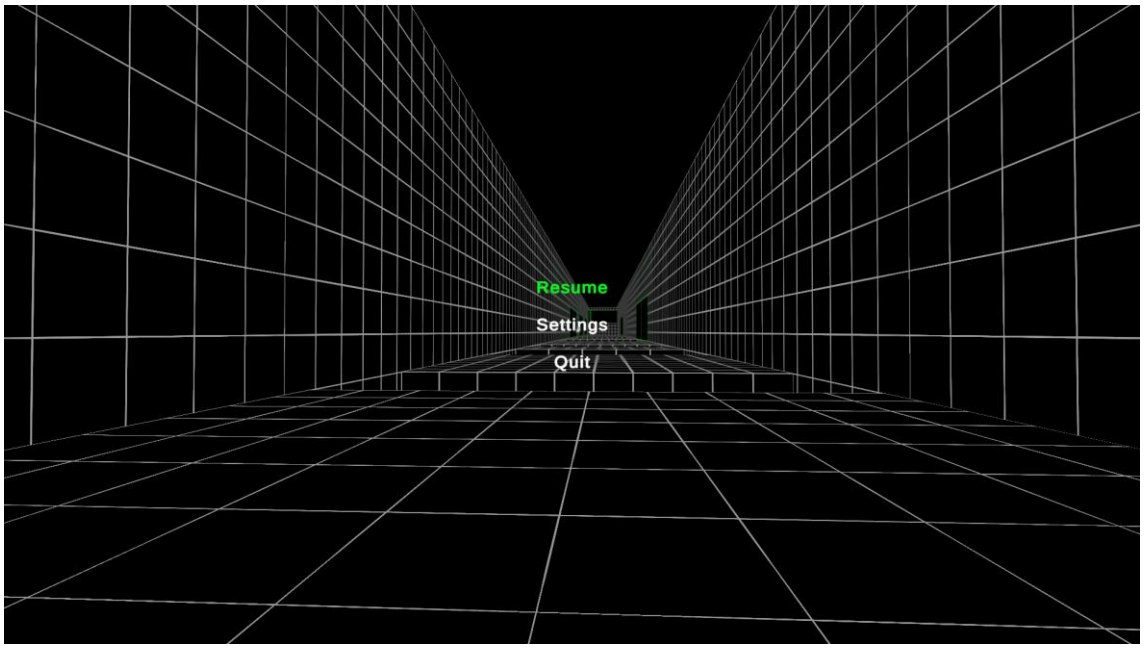

Use "WASD"/Left Joystick
to move

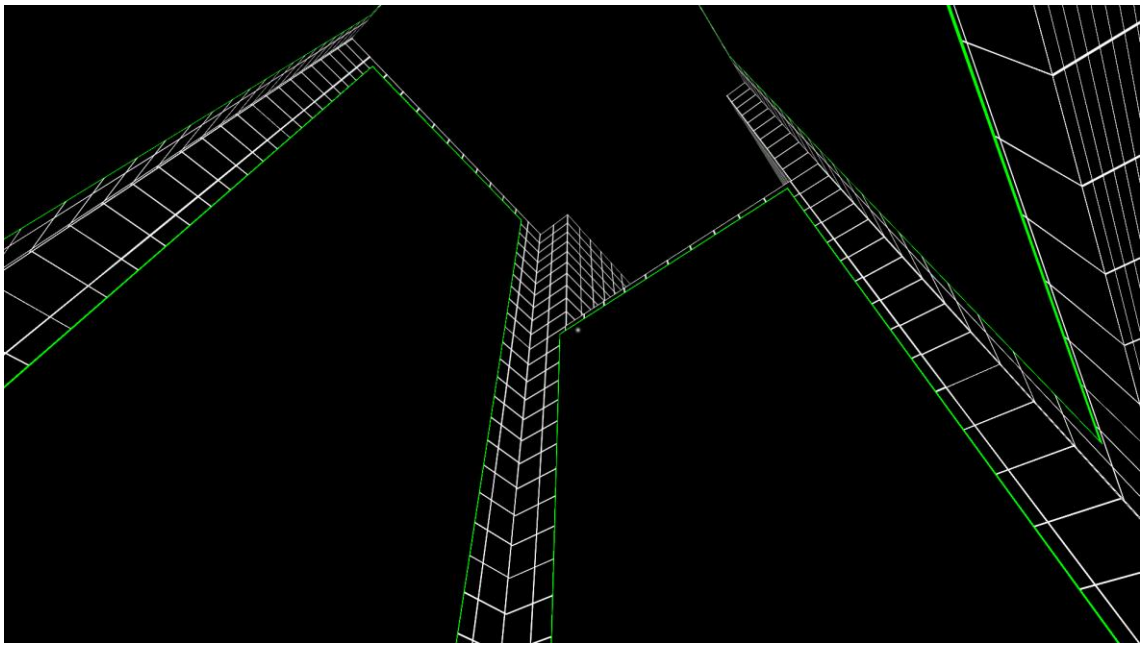


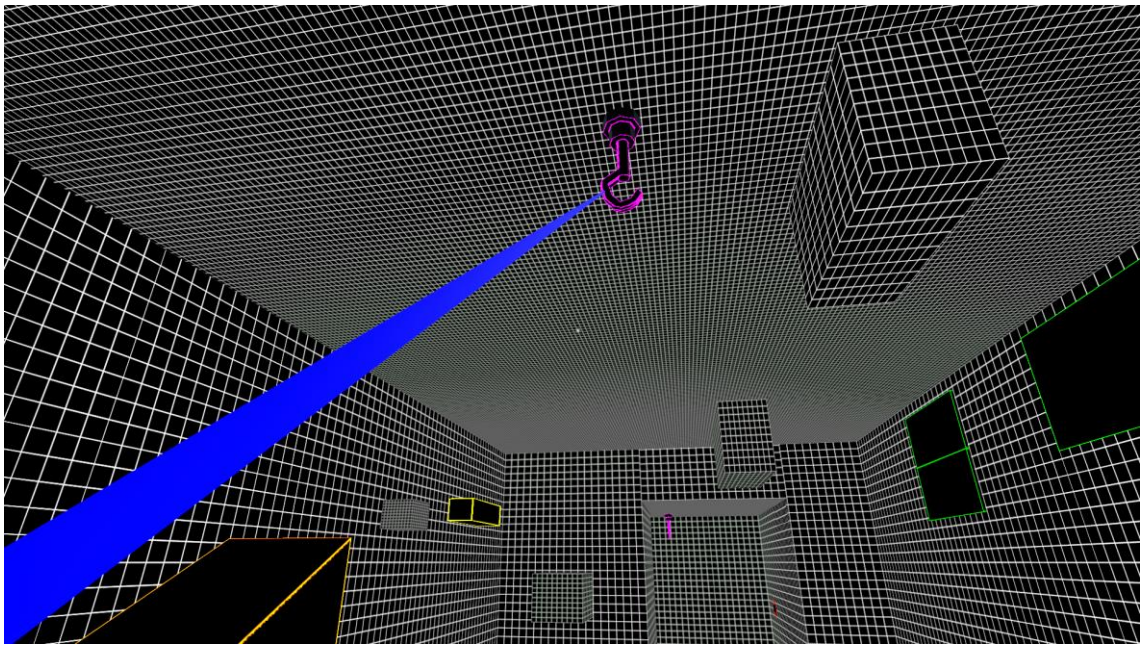

Holding "Left
Control"/Button South/ RT
in a ramp will make you go
faster.


Congratulations, you beat the level!
Would you like to advance or play the actual
level again?
Play it again!
Advance next level!

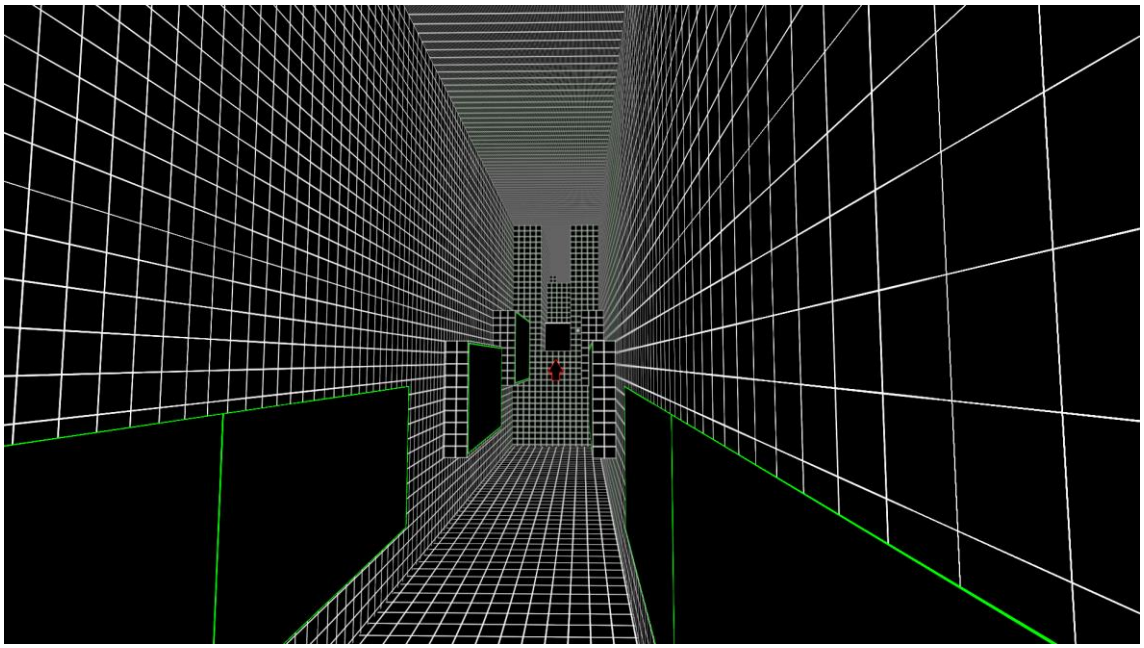



Resume
Settings
Quit

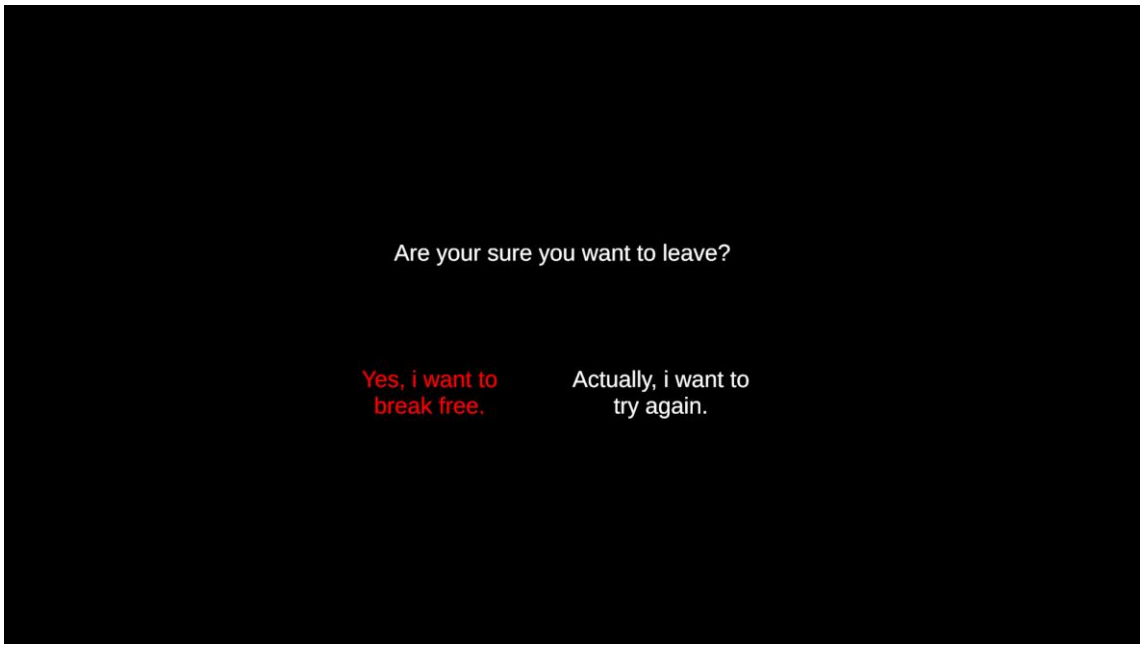





GET READY
1




Congratulations, you beat the level!
Seems like there's an exit. Do you want to be free or repeat the level?
Play it again!
Be free!


Are your sure you want to leave?
Yes, i want to break free.
Actually, i want to try again.

8. Conclusions

Per la meva part estic orgullós del resultat final del projecte. No només s'han assolit els objectius establerta al principi del projecte, si no que s'ha portat un ritme constant que ha sigut determinant per aconseguir el proposat. També cal destacar l'experiència d'aprenentatge que he anat adquirint al llarg del desenvolupament del joc. Aquesta oportunitat m'ha permès provar les meves capacitats davant d'un projecte relativament gran i adquirir molta habilitat amb les eines que m'ha ofert la carrera al llarg d'aquests anys.

Un dels aspectes més important a l'hora d'assolir aquest projecte ha estat la manera en que hem estructurat el projecte. Una bona gestió del temps i una organització de les tasques han estat vitals per assegurar un ritme eficient i constant. Ha estat molt gratificant veure com el projecte anava agafant forma a mesura que anàvem seguint l'estructura establerta.

Personalment, el que més m'ha agradat ha sigut la programació i implementació de les mecàniques el joc. Veure com cada línia de codi anava formant part de la totalitat del projecte ha estat una de les millors experiències d'aquest projecte. També m'ha servit per ser una mica més eficient amb el meu propi codi, ja que en l'àmbit dels videojocs, és molt important tenir un codi net per evitar que el joc es saturi, arruïnant l'experiència dels usuaris. Aquesta és una lliçó que tindré sempre en compte.

La realització d'aquest projecte també m'ha permès treballar en àrees que no havia explorant suficient durant el grau, un exemple a destacar seria el disseny de nivells i la implementació d'interfícies fàcils i còmodes. Aquesta experiència m'ha fet valorar molt més les persones que s'especialitzen en aquest àmbit, ja que requereix no només un punt de vista tecnològic, si no també una mica d'habilitats en les arts humanístiques.

L'experiència de treballar amb Unity ha estat excepcional. Durant aquest procés he notat un increment notable dels coneixement en aquest motor, ja que he hagut d'afrontar problemes i reptes únics que no hauria trobat de no ser per aquest projecte. Durant aquesta experiència he notat com adquiria noves habilitat a l'hora de manipular el motor per aconseguir els efectes desitjats pel meu projecte. També he notat una millora en la utilització de llibreries de Unity per facilitar certes situacions on el motor es veia limitat.

Per finalitzar, aquest projecte ha estat una gran lliçó sobre el desenvolupament de videojocs. L'estructuració del projecte, l'aprenentatge continu i la dedicació son factor que son molt important per aconseguir resultats positius. Gràcies a aquesta experiència m'ha permès evolucionar en gran mesura en l'àmbit de la creació dels videojocs i estic satisfet amb tot el progrés que he aconseguit durant aquest projecte i durant aquests anys en el grau.

9. Treball Futur

Després d'haver completat el prototipus, he estat pensant algunes idees de cara a futur per si volgués continuar el projecte. Encara que estigui satisfet amb el producte final, no significa que no hi hagi espai per la millora. Com a treball a futur, tinc unes quantes idees sobre la millora del videojoc:

- **Implementació de nivells:** En l'estat actual del joc només hi son jugables 2 nivells sense comptar el tutorial. Com a treball a futur, seria la implementació de la resta dels nivells que hi falten (del 2 al 9) i la implementació d'altres nivells alternatius que el jugador hauria de buscar amb més profunditat. Ja que en el final del joc es deixa veure que el jugador hi podria trobar altres alternatives per sortir del videojoc.

- **Implementació de més finals:** Com he explicat anteriorment, el final del joc dóna peu a que el jugador explori amb més detall els nivells que ha estat jugant per així trobar camins alternatius. Un dels plans a futur seria la creació d'un final alternatiu en cas de que el jugador aconsegueixi arribar per aquells camins alternatius.

- **Nivell cronometrats:** Al ser un joc on la velocitat i la fluïdesa són clau per a una bona experiència, penso que estaria bé la implementació d'una opció de repetir nivells amb un comptador de temps. D'aquesta manera la gent podria veure d'una manera més directa la millora en les seves habilitats i, fins i tot, oferir un sentiment de competitivitat als jugadors més determinats.

- **Creació de eina per fer nivells:** Apart dels nivells creats, penso que la incorporació d'eines per a que els usuaris poguessin crear els seus propis nivells oferiria molta més rejugabilitat, a més de oferir la oportunitat de crear una comunitat sana amb la qual la gent pot valorar i compartir les creacions.

- **Millores tècniques:** Per a l'optimització del joc, seria beneficiós escoltar els comentaris dels jugadors, ja que això ens permetria identificar els possibles errors molt més ràpid que de normal, a més de poder implementar millores que nosaltres no hi havíem pensat.

10. Bibliografia

1. Unity tecnologies(s. f.). Unity Manual https://docs.unity3d.com/
2. Unity Technologies. (s. f.). Welcome to Unity. Unity Blog. https://blog.unity.com/
3. Unity tecnologies(s. f.). Unity Forum. https://forum.unity.com
4. Unity tecnologies(s. f.). Unity Discussions. https://discussions.unity.com/
5. Unity tecnologies. Getting started with post-processing | Post Processing | 3.0.3. (s. f.). https://docs.unity3d.com/Packages/com.unity.postprocessing@3.0/manual/Quick-start.html
6. Unity tecnologies. Controls | Input System | 1.0.2. (s. f.). https://docs.unity3d.com/Packages/com.unity.inputsystem@1.0/manual/Controls.html
7. Brackeys. Canal de youtube “Brackeys”. https://www.youtube.com/brackeys
8. SpeedTutor. Canal de youtube “SpeedTutor”. https://www.youtube.com/SpeedTutor
9. Sasquatch B Studios. Canal de youtube “Sasquatch B Studios”. https://www.youtube.com/sasquatchbgames

11. Annexos

En l'entrega, hem adjuntat un arxiu .txt amb enllaços on es podrà trobar tant l'executable del joc com el projecte de Unity sencer. Apart també hem afegit un enllaç per descarregar un vídeo on es pot veure tots els nivells, incloent-hi el final.

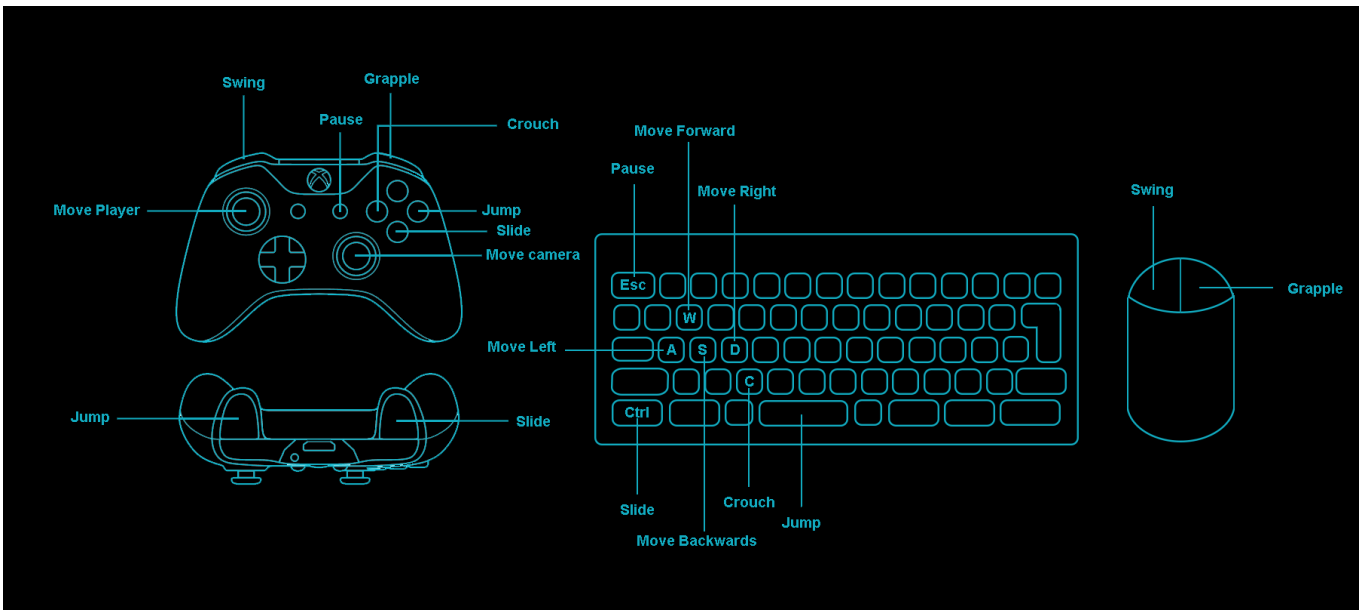
12. Manual d'usuari

Aquest apartat s'explicarà pas a pas com jugar-hi al projecte a més d'explicar en més detall els controls d'aquest.

Per començar es descomprimeix el zip del executable que hem proporcional en el link del text adjunt, un cop descomprimit obrim l'executable. La primera cosa que apareix és el menú principal, on hi ha vàries opcions, per començar a jugar s'ha de fer clic a "New game". Encara que en el menú d'opcions (a l'apartat de gameplay concretament) hi ha una opció per veure els controls, el primer nivell permet al jugador veure els controls d'una manera dinàmica i còmode.

- El personatge és mou utilitzant tant les tecles W,A,S i D com el Joystick dret del comandament de consola.
- Per moure la càmera simplement s'ha de moure el ratolí. En cas del comandament s'ha d'utilitzar el Joystick esquerra
- Per saltar s'ha de prémer la tecla espai en el teclat, o bé el botó dret del comandament (encara que també hi ha l'opció de saltar amb el botó LT).
- Per fer que el jugador llisqui, s'ha de prémer la tecla Lctrl del teclat o, en cas de comandament, el botó d'avall a la dreta o bé el botó RT.
- Amb la tecla C el jugador es pot ajupir. En el comandament també o pot fer prement el botó de l'esquerra.
- Per gronxar-se s'ha de fer clic amb el botó esquerra del ratolí o bé el LB del comandament. Mentre el jugador s'està gronxant, pots allargar la corda fent clic al botó d'ajupir-se o escurçar-la donant-li al botó de saltar.
- Per enganxar-se s'ha de fer clic amb el botó dret del ratolí o be prémer RB en el comandament de consola.
- Finalment, per pausar el joc, s'ha de fer clic al ESC del teclat o, en el seu defecte, el botó select del comandament.

A la següent pàgina podem observar una imatge amb tots els botons necessaris per jugar-hi.


Swing
Grapple
Pause
Crouch
Move Player
Jump
Slide
Move camera
Move Left
Jump
Slide
Move Forward
Pause
Move Right
Esc
W
A
S
D
C
Ctrl
Slide
Crouch
Jump
Move Backwards
Swing
Grapple