Skyline Chase

(Projecte de desenvolupament)
CFGM Administració de Sistemes Microinformàtics i Xarxes

Autors: Dídac Rubio i Rubèn Alcantarilla
Grup: SMX2A
Curs acadèmic: 2025-2026
Tutor: Ruben Arroyo
Fecha de entrega: 17/05/2026

Resum del projecte:

Temàtica del projecte

Skyline Chase és un joc de plataformes que hem creat amb Godot Engine. En el joc podràs controlar diferents personatges, però el principal és un cavaller, i hauràs d'anar superant nivells plens de salts i obstacles. La idea principal és moure't bé i calcular quan has de saltar per no caure.

Objectiu del projecte
Volíem fer un joc que fos divertit, però que també obligués el jugador a estar atent. Cada nivell és diferent perquè no sigui repetitiu. Per superar-los cal utilitzar habilitats com el doble salt o lliscar per la paret. També hi ha tres estrelles amagades a cada nivell, que no són obligatòries, però si les vols aconseguir totes has d'explorar més i jugar millor. El joc també compta amb un sistema de vides interactiu que limita els intents i afegeix més dificultat.

Metodologia seguida per aconseguir l'objectiu
Primer vam programar el moviment del personatge fins que ens va agradar com se sentia. Després vam anar creant els nivells i provant coses diferents, com plataformes que es mouen o que desapareixen. Hem hagut de provar el joc moltes vegades per veure què funcionava i què no, fent petits ajustos durant el procés.

Resum de les conclusions
Hem après que fer un videojoc no és només programar, sinó també provar molt i tenir paciència. A poc a poc el joc ha anat millorant i, al final, hem aconseguit un resultat del qual estem contents.

Paraules clau (entre 4 i 8):

- Skyline Chase
- Plataformes
- Personatges
- Parkour
- Enemics
- Nivells

Abstract:

Project topic

Skyline Chase is a platform game that we created with Godot Engine. In the game, you can control different characters, but the main one is a knight, and you will have to complete levels full of jumps and obstacles. The main idea is to move carefully and calculate when to jump so you don't fall.

Project objective

We wanted to make a fun game, but it also required players to stay focused. Each level is different, so it doesn't feel repetitive. To complete them, you must use skills such as double jump or sliding wall. There are also three hidden stars at all levels. They are not required to finish the level, but if you want to pick them all up, you have to explore more and play better. The game also includes an interactive life system that limits the number of attempts and adds additional difficulties.

Methodology used to achieve the goal

First, we programmed the character movement until we were happy about how it felt. We then started creating levels and testing different ideas, such as mobile platforms or missing platforms. We've had to try the game many times to see what worked and what didn't, making small changes along the way.

Summary of conclusions

We learned that making a video game is not just about programming, but also about trying a lot and being patient. Gradually, the game improved, and in the end we got a result of which we are proud.

Keywords (entre 4 i 8):

- Skyline Chase
- Platforms
- Characters
- Parkour
- Enemies
- Levels

Índex

Llista de figures

1. Introducció

Si fem un cop d'ull al panorama actual del món dels videojocs, ens adonem ràpidament que convivim en una dualitat curiosa: d'una banda, produccions amb pressupostos molt elevats que busquen el fotorrealisme, i de l'altra, idees molt senzilles que aconsegueixen captivar milions de persones. Tot i aquesta evolució constant, si hi ha un gènere que aguanta el pas dels anys sense perdre ni un pèl de frescor, és el de les plataformes. La premissa de saltar, calcular distàncies i esquivar obstacles és una cosa que gairebé qualsevol persona entén a la primera, sense necessitat de manuals complexos. És precisament aquesta **universalitat** el que ens va empènyer a nosaltres a crear **Skyline Chase**. Volíem agafar aquesta base clàssica que tots hem jugat de petits i traslladar-la a un entorn 3D que fos modern i divertit, però que sobretot no perdés l'essència dels videojocs classic de parkour: la precisió.

Skyline Chase no és només un projecte on pots controlar un cinc personatges en un món 3D; és el resultat de mesos de barallar-nos amb el motor **Godot Engine** per aconseguir que cada moviment se senti le mes natural posible. La missió principal de cada nivell és arribar a la bandera final, però a mesura que avançàvem en el disseny, ens vam adonar que el repte real no era "omplir" el mapa de coses, sinó aconseguir que el joc fos "just". En molts jocs 3D fets per equips petits, el jugador acaba frustrat perquè la càmera fa coses rares o perquè el personatge no respon exactament quan prems el botó. Per evitar això, ens hem obsessionat a polir el codi del control. Hem buscat que el jugador aprengui noves habilitats de manera orgànica; no et donem tot el ventall de moviments des del segon zero, sinó que et deixem descobrir el **doble salt** o el **wallslide** quan el nivell realment t'ho demana.

Aquest aprenentatge progressiu és la columna vertebral del joc. No volíem una experiència plana on des del primer minut ja ho haguessis vist tot. A **Skyline Chase**, la dificultat és una escala: primer aprens a moure't i a calcular salts bàsics en plataformes fixes, i quan creus que ja tens el control, introduïm plataformes mòbils, invisibles o parets per on has de lliscar. Aquesta corba d'aprenentatge ens ha obligat a ser molt crítics amb nosaltres mateixos; si un salt ens semblava massa difícil després de deu intents, volia dir que el codi o el disseny del nivell s'havia de retocar.

Un dels punts on hem invertit més hores de "brainstorming" i de picar codi ha estat la **rejugabilitat**. Teníem clar que no volíem fer un joc de "passar i oblidar" en cinc minuts. Aquí és on entra el sistema de les tres estrelles per nivell. No estan posades a l'atzar; cada estrella està situada en punts estratègics que t'obliguen a sortir del camí principal, a jugar-te una vida amb els enemics o a investigar racons del mapa que semblen estar buits. Per gestionar tot això, vam haver de fer el salt tècnic cap a [1]**SQLite**. Volíem que el joc tingués "memòria", que quan tanquessis la partida i tornessis l'endemà, aquelles estrelles que tant t'havia costat aconseguir seguissin allà marcades al menú. Veure les siluetes buides al selector de nivells és la nostra manera de picar el jugador perquè torni a intentar-ho fins a aconseguir el 100%.

D'altra banda, el sistema de vides va ser un dels temes que més vam discutir entre nosaltres. Inicialment pensàvem a fer vides infinites, però ens vam adonar que això matava la tensió. Si morir no té cap càstig, el jugador deixa de tenir por al buit o als enemics i acaba saltant "a la babalà". En posar el límit de **tres vides**, hem aconseguit que la gent jugui amb més calma, observant els patrons dels enemics i calculant millor cada moviment. És una decisió que fa que el joc se senti més "seriós" i gratificant quan finalment toques la bandera.

Figura 1: Foto cavaller i estrella

1.1 Context

El panorama de la creació de videojocs ha fet un gir de cent vuitanta graus en molt poc temps, gairebé sense que ens n'adonéssim. Fa només una dècada, plantejar-se un projecte en 3D era una odissea reservada gairebé exclusivament a grans estudis que disposaven de pressupostos intocables, equips de desenes de programadors i llicències de programari caríssimes que podien costar milers d'euros. Però avui dia la situació és radicalment diferent: el desenvolupament *indie* ha explotat i s'ha democratitzat gràcies a l'aparició de motors cada cop més accessibles com Unity, Unreal o el que hem triat nosaltres per a aquest projecte, **Godot Engine**. Aquestes eines ens donen una base tècnica brutal ja resolta —gestió de renderitzat, memòria i físiques bàsiques—, fet que ens ha permès com a equip centrar-nos en el que realment ens importava: la part creativa, el disseny de nivells i aconseguir que el personatge respongui amb una fidelitat absoluta a les ordres del jugador.

Actualment, el sector ofereix moltes facilitats per fer jocs de plataformes, però som conscients que aquesta facilitat amaga un parany. Godot s'ha fet un lloc en aquesta indústria no només per ser gratuït, sinó per ser programari lliure (*open source*) i extremadament lleuger. Això ha estat un avantatge clau per a un equip d'estudiants com el nostre, ja que ens ha permès treballar sense necessitat d'un ordinador de gamma alta i aprofitar un sistema de nodes que és tan intuïtiu que permet prototipar idees en qüestió de minuts. Tot i això, hem analitzat que tenir tantes eines a l'abast de tothom ha provocat un efecte secundari nociu: una allau de jocs "clons" que no tenen gens de personalitat. El mercat actual, especialment en el gènere de les plataformes, està saturat de projectes que fan servir els mateixos *assets* genèrics i les mateixes mecàniques copiades d'altres títols sense aportar absolutament res de nou a l'experiència de joc.

Analitzant el que hi ha ara mateix a les botigues d'aplicacions o a plataformes de referència per a creadors independents com **Itch.io**, hem detectat un problema que es repeteix constantment i que volíem evitar a tota costa: la manca de poliment. Sovint ens trobem amb jocs que o bé són tan simples que perden l'interès del jugador al cap de pocs minuts, o bé presenten unes físiques tan poc cuidades que el personatge no salta quan toca, rellisca sense sentit o es queda clavat en racons del mapa per culpa de col·lisions mal definides. Som del parer que en un plataformes 3D, la precisió en el control ho és tot; és el 90% de l'èxit. Si el jugador sent que ha mort per culpa d'un error del joc i no per la seva pròpia manca d'habilitat, la frustració apareix i l'experiència es trenca definitivament.

A més, moltes solucions actuals fallen en un aspecte psicològic fonamental: la **corba de dificultat**. Hem vist molts projectes que llancen el jugador a reptes impossibles d'entrada o, per contra, mantenen una dificultat plana que no suposa cap estímul. A Skyline Chase hem volgut trencar amb aquesta tendència. El nostre objectiu no ha estat simplement fer un personatge que camini; la nostra intenció des del primer dia ha estat entendre a fons la interacció entre el jugador i l'espai tridimensional. Volíem que el joc fos un "mestre silenciós", on cada obstacle superat ensenyés una cosa nova al jugador, preparant-lo per al següent repte sense necessitat de tutorials avorrits que tallin el ritme de la partida.

És precisament aquí on creiem que un projecte com el nostre encara té tot el sentit del món. En un moment on sembla que tot ja s'ha inventat, el repte real no és fer la cosa més complexa del món, sinó fer que el que facis se senti sòlid i coherent. Skyline Chase neix de la necessitat de recuperar el plaer per l'exploració i el repte personal. Ja sigui per trobar

aquella estrella amagada en un racó que semblava inaccessible o per superar el nivell sense rebre ni un sol cop, hem buscat incentius perquè la gent vulgui repetir les pantalles. No ens conformàvem amb una experiència de "passar i oblidar"; volíem un joc que es recordés per la qualitat del seu control i per una dificultat equilibrada que, tot i ser exigent, resultés sempre gratificant. Al final, aquest projecte és una aposta per la qualitat per sobre de la quantitat, intentant demostrar que amb eines potents i moltes hores d'ajustar petits detalls, encara es poden crear experiències de plataformes que se sentin fresques, professionals i, sobretot, ben executades.

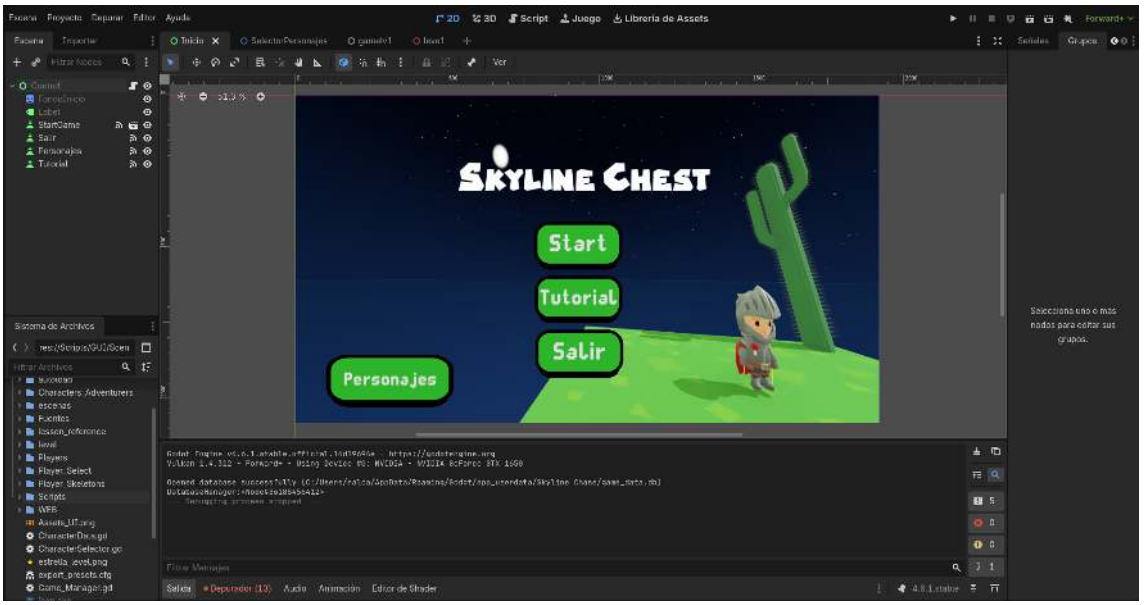

Figura 2: Foto de godot engine amb el nostre videojoc

1.2 Justificació

Hem decidit tirar endavant el projecte de **Skyline Chase** perquè, després d'analitzar detingudament el panorama dels jocs de plataformes actuals, ens vam adonar que la gran majoria pateixen un problema de base que arruïna l'experiència de joc. Molts títols contemporanis, especialment en el sector independent, cauen en un dels dos extrems: o són tan extremadament senzills que no suposen cap estímul intel·lectual ni d'habilitat, o bé presenten unes físiques tan poc cuidades que el personatge no respon amb la precisió que un gènere tan exigent demana. Com a estudiants d'informàtica, ens semblava un repte tècnic i creatiu apassionant intentar resoldre aquesta mancança des de la base. El nostre objectiu no era simplement "fer un joc més", sinó crear una experiència on el control fos realment fi i on el jugador hagués d'aprendre a dominar el moviment de veritat per poder sentir una satisfacció real en completar cada repte.

El valor d'aquest joc no rau només en el fet d'anar d'un punt A a un punt B del mapa; això seria quedar-se a la superfície del que realment hem volgut construir. El valor real es troba en tot l'ecosistema de mecàniques i regles que hem muntat al seu voltant per donar coherència al món de **Skyline Chase**. Per exemple, la decisió d'afegir el sistema de les tres estrelles col·leccionables i el límit estricte de vides no ha estat, ni de bon tros, un caprici estètic o una còpia de jocs clàssics. Ho hem fet amb la intenció clara de dotar de sentit l'exploració en un entorn 3D. Volíem que el jugador es veiés obligat a fixar-se en els petits detalls de l'escenari, a mesurar els riscos de cada salt i a sentir que, quan aconsegueix la bandera final, és exclusivament gràcies a la seva pròpia destresa i no a la sort. Aquesta profunditat aporta una rejugabilitat que molts projectes *amateur* ignoren, i estem convençuts que suposa una millora substancial respecte al que se sol veure en jocs fets per equips petits, on sovint es descuida la corba de dificultat en favor de la part purament visual.

Des d'una vessant més tècnica i acadèmica, la justificació d'aquest treball resideix en la quantitat ingent de coneixements que hem hagut d'adquirir i aplicar, sovint a base de "cops de cap" contra el monitor, analitzant documentació i repetint proves fins a l'esgotament. No es tracta només de fer que un model 3D es desplaci d'un costat a l'altre; hem hagut de picar codi complex en **GDScript**[4], configurar i polir les col·lisions en un espai tridimensional que és infinitament més "puntero" i traïdor que treballar en 2D— i optimitzar les escenes dins de **Godot** perquè tot funcioni amb una fluïdesa total, sense caigudes de fotogrames que puguin afectar la jugabilitat. Aquest aprenentatge ens ha servit per entendre, potser per primera vegada de manera tangible, com es gestiona un projecte de programari real de principi a fi. Hem après que cada petita decisió de disseny, per insignificant que sembli, pot ser la diferència entre un producte final que sigui una delícia de jugar o una font constant de frustracions per a l'usuari.

En definitiva, desenvolupar Skyline Chase ens ha permès tocar moltes tecles de manera simultània, posant a prova la nostra capacitat de resolució de problemes. Hem passat de la programació pura i dura de la lògica del personatge fins al disseny d'arquitectura de nivells, passant per la gestió de dades amb **SQLite** i l'optimització de recursos gràfics. Per a nosaltres, aquest projecte és la millor carta de presentació possible per demostrar que som capaços d'agafar una eina professional i transformar una idea abstracta en un producte sòlid i executable. No ens hem conformat amb fer "un joc que funcioni", sinó que hem buscat crear un sistema ben pensat des de la seva estructura interna, que ofereixi una experiència de qualitat i que sigui plenament coherent amb les expectatives del jugador

actual. Creiem fermament que la suma de tots aquests detalls —el control polit, el repte de les estrelles i la tensió de les vides— és el que realment justifica les hores invertides i el que dóna un valor diferencial al nostre treball dins del sector dels videojocs independents.1.3 Objectius

Amb aquest treball no ens volem quedar només en la superfície de fer un personatge que camini per una pantalla. La nostra meta real és veure si som capaços de muntar un sistema de joc que se senti professional, on cada moviment i cada mecanisme estigui ben lligat. Volem que **Skyline Chase** sigui un repte de veritat per a qui el jugui, i això ens obliga a ser molt precisos amb tot el que anem a programar.

1.3.1 Objectiu general

La idea principal és desenvolupar amb **Godot Engine** un videojoc de plataformes en 3D que sigui fluid i on el jugador hagi d'anar superant diferents nivells. No és només saltar, sinó dominar el personatge i saber gestionar les poques vides que tens per arribar a la bandera de meta.

1.3.2 Objectius específics

De cara a la part més pràctica, ens hem marcat uns punts concrets que hem de complir sí o sí:

- Primer de tot, **dominar a la perfecció el moviment del personatges**. Si el doble salt o el lliscament per les parets falla una mica o no funciona com una bona mecanica, el joc ja no serveix. Hem de programar-ho perquè la resposta al comandament sigui instantània i fliuda per al jugador.
- Dissenyar uns **mapes que vagin de menys a més (**dificultat ascendent**)**. Volem que la dificultat pugi a poc a poc, ficant plataformes que es mouen o trampes que et facin caure si no vas amb compte.
- Ficar **enemics als nivells**. Han de tenir les seves rutes de patrulla alatzar (utilitzarem un sistema de IA) per fer que el jugador s'hagi de parar a pensar per on passar sense que el matin.
- Fer que el **sistema de vides i estrelles** funcioni bé. Cada nivell tindrà tres estrelles repartides per tot el nivell per obligar la gent a buscar per tot el mapa i no anar nomes cap a la bandera.
- Muntar una **bona interfície de usuari GUI**. Necessitem un menú principal i una pantalla per triar nivell on es vegi quin progrés portes i quantes estrelles has recollit als diferents nivells, tambe tindrem per poder escollir al personatge.
- Assegurar-nos que el **3D no doni problemes de rendiment**. El joc ha d'anar suau, sense estrebades que facin que perdis una vida per culpa d'un error de frames.
- Finalment, **afegir sons i efectes** que donin vida al joc, perquè no volem que l'experiència sigui avorrida o que se senti buida quan saltes o agafes una estrella.
- Tindre **totes les animacions preparades.** Cada personatge te animacions fins els enemics tenen animacions de mort llavors hem de tenir prepardes totes les animacions per les diferents situacions.

Si al final del projecte hem aconseguit tot això, voldrà dir que hem estat capaços de gestionar un programari complex des de zero. Més que fer un joc, el que volem és

demostrar que podem resoldre tots els problemes tècnics que surten quan ajuntes codi, gràfics i jugabilitat en un mateix lloc desde zero i casi sense coneixements.

1.4 Estratègia i planificació del projecte

Abans de posar-nos a picar ni una sola línia de codi a Godot, ens vam haver de seure a pensar seriosament com ho faríem per no acabar perduts pel camí. Teníem clarissim que desenvolupar un joc en 3D sent només dos, i amb el temps tan limitat de què disposàvem, era un repte considerable que ens podia explotar a la cara si no anàvem amb peus de plom. Per això, la nostra estratègia no ha estat mai fer coses a cegues o per impuls, sinó seguir un ordre lògic que tingués sentit dins d'un entorn de producció real: primer, entendre a fons com funciona el motor per dins; després, aconseguir que els personatges es moguésin amb una fluïdesa total i, finalment, abocar-nos al disseny dels nivells. Sabíem que si la base del moviment o el comportament de les càmeres fallaven, tota la capa visual o el contingut dels nivells no serviria per a res. Per aquest motiu, les primeres setmanes les vam "cremar" literalment fent mil proves de físiques, ajustant valors de gravetat al mil·límetre i polint les col·lisions per evitar aquell efecte tan molest on el personatge es queda travat a les parets o vibra inexplicablement en tocar un obstacle.

Pel que fa a la gestió del dia a dia, ens hem organitzat dividint la feina per tipus o objectius setmanals concrets. En lloc d'atabalar-nos amb la idea gegantina de "fer el joc", que de vegades es veia inabastable, anàvem a pams. Ens fixàvem metes curtes però realistes; dèiem, per exemple: "aquesta setmana la prioritat absoluta és tenir el doble salt i el sistema de vides funcionant al 100%". Aquest mètode de treball per iteracions ens ha permès mantenir un ritme constant i, sobretot, ens ha ajudat a no perdre els nervis quan sortien aquells errors estranys de codi o bugs visuals que n'han sortit a patolls durant tot el procés. Per gestionar aquest caos de tasques, hem fet servir **Trello**, on anàvem movent les targetes des de la columna de "pendents" a la de "fet", assegurant-nos que cada funcionalitat passava per una fase de test abans de donar-la per tancada. A més, l'ús de **GitHub** ha estat la nostra salvació real i la columna vertebral de la col·laboració; ens ha permès treballar en paral·lel sense la por constant a sobreescriure la feina de l'altre i, el més important, ens ha donat la seguretat de poder fer un "rollback" i tornar enrere si una idea nova trencava accidentalment alguna cosa que ja funcionava bé.

D'altra banda, durant tot el procés hem intentat ser molt realistes amb els recursos i el temps de què disposàvem. Som conscients que en un projecte escolar o de recerca s'ha de posar el límit on toca. No podíem pretendre fer un món obert gegant que mai acabaríem, així que vam decidir centrar-nos a polir al màxim tres o quatre mecàniques que fossin realment divertides i satisfactòries: el doble salt, el lliscament per les parets i el sistema de recollida d'estrelles. El nostre pla de treball ha estat un organisme viu; si vèiem que una idea no acabava de funcionar en pantalla o que el personatge feia moviments estranys en xocar amb determinats objectes, paràvem màquines. No seguíem endavant fins que no trobàvem l'arrel de l'error o el resultat exacte que realment buscàvem. Aquesta filosofia d'assaig i error ens ha tret moltes hores de son, però ha estat l'única manera de garantir un mínim de qualitat.

Aquesta manera de treballar ens ha obligat a estar en una comunicació constant i totalment transparent per decidir què era prioritari en cada moment de crisi. Encara que a

vegades aquesta deliberació ens ha fet anar una mica més lents del que ens hagués agradat, estem convençuts que és precisament aquest rigor el que ha garantit que el resultat final sigui un bloc sòlid, coherent i ben estructurat. Volíem que aquesta memòria que teniu a les mans no fos només un recull tècnic de codi font, sinó que reflectís un procés de treball seriós, madur i conscient, on cada decisió, per petita que fos, s'ha pres amb l'objectiu de treure el màxim profit dels nostres coneixements de programació i de les potents possibilitats que ens ofereix un motor com Godot.

1.5 Justificació

La decisió de tirar endavant el projecte de **Skyline Chase** neix d'una inquietud personal profunda pel disseny de sistemes interactius i, especialment, per la falta de jocs de plataformes 3D gratuïts que realment cuidin la relació íntima entre el jugador i el joc. Després d'analitzar molts projectes en plataformes com Itch.io o en diversos repositoris de codi obert, ens vam adonar que la gran majoria fallen en el mateix punt crític: el *game feel*. Sovint el personatge no té pes, els salts semblen surar de manera imprecisa o la càmera es converteix en un autèntic malson que t'impedeix jugar bé. Amb aquest treball, ens hem proposat demostrar que es pot crear un títol sòlid, amb un control que respongui al mil·límetre al que el jugador té al cap, i amb un sistema de progressió que, tot i ser exigent, mai resulti injust.

El valor real d'aquest treball no és simplement fer uns models 3D que es desplacin per un escenari, sinó implementar mecàniques que facin que l'experiència sigui realment rejugable i, per sobre de tot, divertida. En lloc de dissenyar nivells infinits però buits de contingut on l'únic objectiu sigui córrer cap a la meta, hem apostat per un disseny molt més dens. El sistema de les tres estrelles i el límit estricte de tres vides no són elements posats a l'atzar; estan pensats per obligar el jugador a "llegir" l'entorn i a parar atenció als detalls. Això resol de soca-rel el problema de la monotonia que arrosseguen molts jocs senzills. Aquí, si vols aconseguir el 100%, t'has d'arriscar, has de dominar les mecàniques de *wallslide* i doble salt, i has de conèixer els riscos de cada plataforma. Aquesta combinació de dificultat calibrada i exploració activa és el que creiem que aporta un valor afegit diferencial respecte al que se sol veure en l'àmbit dels projectes acadèmics convencionals.

Des d'una vessant purament professional i informàtica, aquest projecte es justifica pel salt gegantí que ens ha suposat quant a aprenentatge tècnic. Passar de la teoria acadèmica de la programació a la pràctica real amb un motor tan potent com **Godot Engine** ens ha posat a prova constantment. Fer un joc en 3D ens ha obligat a gestionar problemes de físiques vectorials, lògica complexa de scripts en **GDScript**[4] i una optimització de recursos en temps real que no ens havíem plantejat mai. Per a un futur informàtic, barallar-se amb les col·lisions en un espai tridimensional mentre hi ha enemics patrullant i plataformes mòbils sincronitzades és una lliçó de resolució de problemes que no surt a cap llibre de text. Són problemes tècnics molt tediosos de solucionar, que requereixen hores de *debugging* i de picar-se els dits contra el teclat, però que ens han servit per consolidar coneixements de gestió d'estats i, el més important, per entendre com pensa realment un usuari final quan interactua amb un programari exigent.

Finalment, el resultat d'aquest projecte va molt més enllà d'un simple joc per passar l'estona; és una base tècnica i una metodologia de treball que ens serviran de trampolí per a futurs desenvolupaments molt més ambiciosos. Ens ha ensenyat el valor de la documentació d'errors (bugs), la importància de saber fer marxa enrere amb **GitHub** quan una funcionalitat nova "trencava" tot el joc, i sobretot, a treballar en equip sota una pressió i uns terminis realistes. Estem convençuts que aquesta és la millor justificació possible per a Skyline Chase: no només hem creat un producte que funciona i que és capaç d'entretenir,

sinó que tot el procés ens ha preparat molt millor per a un món laboral on la precisió tècnica, la capacitat d'adaptació i l'eficiència en el codi ho són absolutament tot.

1.6 Metodologia de treball

Perquè el desenvolupament de **Skyline Chase** no es convertís en un caos total o en un pou d'hores perdudes, nosaltres ens vam haver de marcar-nos una manera de treballar que fos clara i, sobretot, realista des del primer minut. El nostre enfocament no ha estat lineal, sinó que hem barrejat les fases clàssiques de planificació amb un desenvolupament iteratiu. Això, a la pràctica, vol dir que en lloc d'intentar fer tot el joc de cop cosa que ens hauria bloquejat de seguida, hem anat creant peces petites i funcionals que es podien testar per separat per després anar-les integrant al conjunt. Aquesta estratègia ens va permetre tenir un prototip jugable del cavaller ja des de la segona setmana, un fet que ens va donar una injecció de seguretat vital per seguir avançant sense la por constant que el projecte s'enfonsés per falta d'una base sòlida.

1.6.1 Enfocament de treball i disseny per capes

Hem tirat per un camí molt pragmàtic, centrant tots els esforços inicials en el que anomenem el "nucli" del joc: el moviment del personatge en l'entorn 3D. Teníem clarissim que fins que el cavaller no es mogués de forma fluida i respongués exactament com nosaltres volíem, no tenia cap sentit perdre el temps dissenyant nivells complicats ni polint l'apartat visual. Un cop aquesta base mecànica va estar lliure de bugs greus, vam començar a afegir capes de complexitat: primer les trampes i les plataformes mòbils, després la lògica dels enemics i, finalment, el sistema de persistència de les estrelles. Aquesta flexibilitat ens ha salvat en més d'una ocasió; si veiem que una mecànica, com el lliscament lateral, es tornava massa "punyetera" de programar a Godot, teníem marge per canviar el disseny i ajustar-lo ràpidament sense que tota la feina feta fins llavors se n'anés en orris.

1.6.2 Organització i coordinació del tàndem

Gestionar les tasques d'un equip de dues persones pot semblar fàcil, però requereix una coordinació constant per no repetir feina. Hem dividit el projecte en tres grans blocs: **Mecàniques i Físiques**, **Disseny de Nivells i Escenaris**, i **Interfície i Sistema de Progressió**. Cadascú de nosaltres es feia responsable d'una part, però sempre sota la supervisió de l'altre. Ens revisàvem els scripts i les escenes pràcticament cada dia, comentant els problemes que anàvem trobant (que no n'han estat pocs) i decidint si necessitàvem buscar una solució alternativa o si el que teníem ja complia amb els estàndards que ens havíem fixat. Aquest diàleg constant ens ha permès ser molt més crítics amb el resultat final.

1.6.3 Eines de seguiment i control de versions

Perquè tota aquesta organització no es quedés només en bones paraules, hem fet servir tres pilars tecnològics que han estat fonamentals en el nostre dia a dia:

- **Trello:** Ha estat el nostre tauler de control visual. Hi vam muntar columnes específiques com "Entregar", "En test" i "Finalitzat". Cada vegada que detectàvem un error o se'ns acudia una millora d'última hora, creàvem una targeta. Ens ha ajudat a no perdre el nord i a recordar detalls que sovint s'obliden, com el poliment dels sons o els ajustos de la interfície d'usuari.

- **GitHub:** Per a un projecte de programació, el control de versions és una qüestió de vida o mort. Gràcies a GitHub hem pogut treballar de forma simultània en diferents branques del codi sense por a trepitjar-nos els fitxers de l'escena de Godot. També ens ha servit de salvavides; més d'un cop hem hagut de recórrer a un "rollback" d'emergència perquè una funcionalitat nova havia rebentat part de la lògica del personatge que ja consideràvem acabada.
- **Gmail i Discord:** Més enllà de la gestió tècnica, la comunicació immediata ha estat a través de Discord. Allà compartíem captures de pantalla de l'editor de Godot amb errors incomprensibles o tutorials que ens ajudaven a resoldre dubtes concrets de GDScript[4].

En conjunt, aquest sistema ens ha permès mantenir una eficiència alta i la motivació intacta. No hi ha res que doni més satisfacció que veure com les targetes de Trello es van movent cap a la columna de "Finalitzat", confirmant que el projecte creixia i que, malgrat els obstacles tècnics, teníem el control total sobre la producció de **Skyline Chase**.

1.7 Estudi econòmic i pressupostari

El desenvolupament de videojocs en 3D no és com escriure un document de text; requereix uns equips amb una potència de processament i una capacitat gràfica mitjana-alta. Hem de tenir en compte que el motor **Godot Engine** ha d'estar obert simultàniament amb eines de modelatge com **Blender** o editors de codi, cosa que consumeix molts recursos.

- **Equips informàtics:** Per a aquest projecte, hem comptat amb ordinadors portàtils i de sobretaula tipus *gaming*. Hem establert com a estàndard l'ús de targetes gràfiques dedicades (família NVIDIA RTX o similars) per garantir un bon rendiment en el renderitzat de les escenes i el testatge del joc en temps real. Un punt innegociable ha estat disposar de, com a mínim, **16 GB de memòria RAM**, ja que és la quantitat que permet treballar amb fluïdesa sense que l'equip es col·lapsi quan compilem els *scripts* o gestionem textures d'alta resolució.
- **Perifèrics i ergonomia:** No només hem comptat la torre de l'ordinador. Hem integrat en el pressupost monitors amb una bona fidelitat de color per a les tasques de disseny visual i interfície, a més de teclats i ratolins ergonòmics. Sembla un detall menor, però després de jornades de sis o set hores picant codi, la comoditat és un factor de viabilitat real per no cremar l'equip de treball.

1.7.2 Programari i llicències (Software)

Aquest és l'apartat on hem aconseguit que el projecte sigui realment viable per a un equip independent gràcies a l'aposta pel programari de codi obert.

- **Motor de joc (Game Engine):** Aquí és on hem estalviat més. En lloc de triar motors que demanen *royalties* o subscripcions cares, hem optat per **Godot Engine**. En ser programari lliure i totalment gratuït (sota llicència MIT), ens permet reduir les despeses a zero en aquest camp, sense límit de facturació ni pagaments per instal·lació. Això fa que el pressupost de *software* es mantingui molt baix sense sacrificar potència tècnica.
- **Addons i Assets de suport:** Tot i que hem programat la major part des de zero, hem previst una petita partida pressupostària per a la compra de connectors específics o *shaders* preconfigurats. A vegades, comprar un *asset* de 20 € que et resol un sistema d'IA o de partícules et permet estalviar 40 hores de feina, la qual cosa és una decisió d'eficiència pura.

- **Eines de disseny i textures:** Per a les textures i els elements gràfics, hem contemplat tant la suite d'**Adobe** (Photoshop i Illustrator) com alternatives gratuïtes tipus **GIMP** o **Krita**. Per als models 3D, Blender ha estat l'eina estrella, de nou a cost zero.
- **Gestió i col·laboració en el núvol:** Finalment, per mantenir el projecte ordenat, hem utilitzat les versions gratuïtes de **GitHub** per al control de versions i **Trello** per a la gestió de tasques. Hem calculat que, per a la mida del nostre equip, no cal pagar les versions *Premium*, mantenint la viabilitat econòmica del projecte sota control.

Concepte	Descripció	Cost estimat (€)
Maquinari	Ordinadors i perifèrics	1.500€ - 3.000€
Llicències SW	Subscripcions anuals i llicències	500€ - 1.200€
Assets/Addons	Recursos comprats (botigues oficials)	200€ - 500€
Contingències	Imprevistos i altres despeses (10%)	250€
TOTAL		**2.450€ - 4.950€**

2 Descripció del projecte

2.1 Anàlisi de requisits

Per considerar el desenvolupament del videojoc Skyline Chase complet, s'han definit una sèrie de requisits que el sistema ha de complir.

2.1.1 Requisits funcionals

- **Control total dels personatges i salts:** El més bàsic era que els personatges respongués bé. Hem programat el moviment per tot el nivell i, sobretot, el sistema de salts entre plataformes, que és la base de tot.

- **Mecàniques de moviment extra:** Per no quedar-nos en un joc massa simple, hem afegit el **doble salt** i el **wallslide**. Això dóna molta més llibertat per moure's i fa que els nivells puguin ser més verticals.

- **Selecció de personatges i menús:** Volíem que el jugador pogués triar entre diferents personatges abans de començar. A més, hem enllaçat el menú de selecció de nivells amb el progrés d'estrelles, perquè sàpigues quantes en portes recollides a cada lloc.

- **El sistema de reptes i enemics:** Dins dels nivells hem repartit enemics i objectes col·leccionables (les estrelles). L'objectiu final sempre és arribar a la bandera, però pel camí t'has de preocupar de no perdre les vides.

- **Gestió de la partida i reinicis:** Hem posat un límit de **tres vides**. Si les perds totes o caus fora del mapa, el sistema reinicia el nivell i les estrelles d'aquella partida. Això ens serveix per posar una mica de pressió al jugador i que no sigui massa fàcil.

2.1.2 Requisits no funcionals

A banda de fer que els personatges saltésin i la base de dades funcionés, teníem una sèrie d'objectius pel que fa a la qualitat que ens preocupaven bastant. No volíem que el joc fos un "vull i no puc", així que ens vam centrar en punts molt concrets per garantir que l'experiència fos sòlida.

El primer que ens va portar cua va ser la **usabilitat**. Ens vam posar en la pell d'algú que no ha vist el joc mai i ens vam esforçar perquè els controls fossin el més naturals possibles; la idea és que comencis a jugar i en deu segons ja sàpigues com moure't sense haver de llegir cap tutorial pesat. També ens hem obsessionat una mica amb el **rendiment**. Godot és un motor lleuger, però tot i així hem anat amb compte de no carregar massa les escenes perquè el joc vagi fluid i no doni aquelles estrebades tan molestes que et fan perdre una vida sense voler.

Pel que fa al codi, com que som dos tocant fitxers constantment, l'**organització** ha estat vital. No ens podíem permetre tenir un codi "brut" perquè si un havia de corregir un error de l'altre, ens haguéssim tornat bojos. Ho hem estructurat tot de manera que, si en un futur ens vingués de gust afegir deu nivells més o noves mecàniques, la base ja estigui preparada per créixer sense haver de refer-ho tot. Finalment, ens hem matat a fer proves per assegurar l'**estabilitat**; el nostre objectiu era que el joc no petés mai ni es quedés penjat, buscant aquells *bugs* estranys que a vegades apareixen quan col·lisiones contra una paret o canvies de pantalla ràpidament.

2.2 Previsió de tasques d’investigació

Abans de poder veure ni tan sols el cavaller movent-se per la pantalla, ens vam haver de posar les piles de debò amb el motor **Godot**. Com que era una tecnologia que no havíem tocat mai a la vida, no ens va quedar una altra que dedicar les primeres setmanes a investigar i a "trastejar" per entendre com funcionava tot per dins. No volíem llançar-nos a programar a cegues, així que vam prioritzar entendre bé la lògica del motor i el seu llenguatge, el **GDScript**[4], abans de voler muntar res massa complicat.

Aquesta fase de recerca ha estat el pa de cada dia durant tot el projecte. Cada vegada que se’ns acudia una idea nova, primer ens tocava passar per una etapa de mirar tutorials, llegir la documentació (que a vegades era un embolic) i fer mil proves en escenes buides per veure si allò funcionava o no. Ha estat un procés d’aprenentatge constant: provàvem una cosa, vèiem per què petava i la tornàvem a ajustar fins que anava fina. Tot seguit, expliquem els punts més importants d'aquesta investigació que ens han permès tirar endavant el joc.

2.3 Tecnologies

2.3.1 Comparativa de les tecnologies valorades

Abans de prendre una decisió definitiva sobre quines farien servir com a ciment tecnològic per a **Skyline Chase**, vam posar sobre la taula dues opcions amb filosofies de desenvolupament totalment contraposades: **Roblox Studio** i **Godot Engine**. Teníem molt clar que aquesta tria no era una simple qüestió de gustos personals o de quina interfície ens semblava més maca a primera vista; implicava decidir quina llibertat real volíem tenir a l'hora de programar, com gestionaríem el codi font i, sobretot, quins canals de distribució tindríem a l'abast per fer arribar el nostre videojoc als usuaris.

Figura 3: Roblox i Godot Engine

D'una banda, l'opció de **Roblox Studio** tenia punts molt atractius que la feien temptadora d'entrada. És un entorn que es caracteritza per donar-te gairebé tota la infraestructura mastegada: inclou el servidor, la gestió d'usuaris i un sistema de publicació immediat que et permet connectar directament amb una audiència potencial de milions de jugadors actius. Com a eina de prototipatge ràpid, és fantàstica perquè compta amb llibreries i objectes preparats per aixecar un escenari en qüestió de tardes, i les mecàniques de xarxa bàsiques es resolen sense haver d'entrar en configuracions complexes.

Tot i aquestes facilitats, quan vam analitzar la plataforma amb ulls crítics, hi vam veure un inconvenient que consideràvem insalvable: el perill del *vendor lock-in* o dependència absoluta de l'ecosistema. Si desenvolupes un producte a Roblox, estàs condemnat a viure i morir dins de Roblox. El joc no es pot exportar com un executable independent per a PC (un fitxer `.exe`), ni es pot penjar en portals oberts com Itch.io de manera autònoma. A més, tot i que el llenguatge [3]**Luau** (la variant de Lua que utilitzen a aquest tipus de motor grafic) és eficient, el motor imposa una sèrie de restriccions de disseny i unes físiques preconfigurades que ens feien dubtar molt de si surtiria bé. Vam valorar que ens acabarien surtint problmes per la creativitat i que ens impedirien aconseguir aquest control fi i personalitzat que buscàvem per als nostres personatges.

D'altra banda, teníem la proposta de **Godot Engine**, que és la que finalment va guanyar el debat. El fet de ser un motor de codi obert (*open source*) i completament gratuït sota la llicència MIT ens oferia una independència i una transparència arquitectònica que Roblox no ens podia donar ni de lluny. El gran avantatge de Godot és la seva flexibilitat radical: en tenir accés lliure a la manera com gestiona els nodes i les col·lisions, podíem tocar qualsevol paràmetre per fer que el joc se sentís exactament com nosaltres havíem planificat. El seu llenguatge natiu, **GDScript**[4], tot i ser un territori nou per a nosaltres, ens va semblar molt agraït d'aprendre gràcies a la seva similitud amb Python, mostrant-se com una eina molt potent i neta per estructurar la lògica d'un plataformes tridimensional.

És completament cert que anar per la via de Godot implicava assumir certs riscos. No compta amb una base d'usuaris captive interconnectada com la de Roblox i, des del primer dia, hem hagut de picar molta més pedra per programar des de zero sistemes bàsics com la base de dades amb SQLite, els menús d'interfície o els reinicis de nivell. Malgrat que aquest camí requeria molt més esforç i hores d'estudi, nosaltres estàvem convençuts que

el resultat final valdria molt més la pena. Treballar amb Godot ha permès que el projecte final sigui realment "nostre", amb un format exportable estàndard i amb una presentació molt més madura i professional que encaixa perfectament amb el perfil d'un treball d'informàtica.

2.3.2 Tecnologies escollides

Després de posar totes les cartes sobre la taula, sospesar els pros i els contres de cada opció i analitzar el recorregut que volíem fer, ens vam decidir definitivament per **Godot Engine** per tirar endavant el desenvolupament de **Skyline Chase**. No va ser, ni de bon tros, una tria feta a l'atzar o per una simple simpatia cap al motor; el que realment ens va guanyar de Godot des del primer minut va ser la llibertat total i absoluta que ens brindava per dissenyar el joc completament des de zero. Ens permetia picar les nostres pròpies mecàniques en GDScript[4] sense haver de dependre de plantilles predeterminades o de configuracions tancades que ens vinguessin imposades pel programari. El més important per a nosaltres era no lligar-nos de mans a una plataforma de tercers que controlés el nostre producte; volíem ser els propietaris reals del nostre codi. A més, el fet que sigui un motor de codi obert (*open source*) i totalment gratuït (gràcies a la seva llicència MIT) ens va semblar l'opció més coherent, ètica i viable per a un projecte d'aquestes característiques acadèmiques.

Un altre punt totalment clau i decisiu en aquesta elecció va ser la gestió del control tècnic. Amb l'arquitectura basada en nodes i escenes que proposa Godot, sentíem que teníem el comandament real sobre la programació, el flux de dades i la jerarquia organitzativa dels nivells. No hi havia "caixes negres" ni processos ocults; si alguna cosa fallava en el moviment dels personatges o en la càmera, sabíem exactament a quin node havíem d'anar a buscar l'error. Això ens ha facilitat moltíssim les coses a l'hora d'assimilar i aprendre com funciona realment el procés complet, real i professional de creació d'un videojoc: des de la conceptualització d'una idea abstracta i la primera línia de codi en brut, fins a la compilació i l'exportació final en un fitxer executable independent. En definitiva, estem completament convencuts que Godot ens oferia un repte intel·lectual molt més enriquidor com a futurs informàtics, obligant-nos a madurar com a programadors i garantint un resultat final molt més robust, polit i professional.

2.4 Estructura del projecte

Perquè **Skyline Chase** funcioni de manera correcta i harmònica, no n'hi ha prou amb dissenyar el model del personatge i col·locar quatre plataformes als nivells; hem hagut de conceptualitzar i implementar una estructura interna robusta on diferents peces i mòduls de programari es parlen entre ells constantment en temps real. Si haguéssim d'analitzar i resumir com està organitzat el videojoc sota la part visual, hauríem de dividir el sistema en diversos blocs funcionals de vital importància com podem observar:

- Level (Part principal del joc)
- Cel (Tota la part visual i estetica del nivel)
- Killplane (Colisio on el jugador mor)
- Flag 3D (Es la meta on el jugador a d'arribar)

El reste ja son coses del nivell i de la seva jugabilitat com ara poden ser les plataformes o fin i tot la decoració.

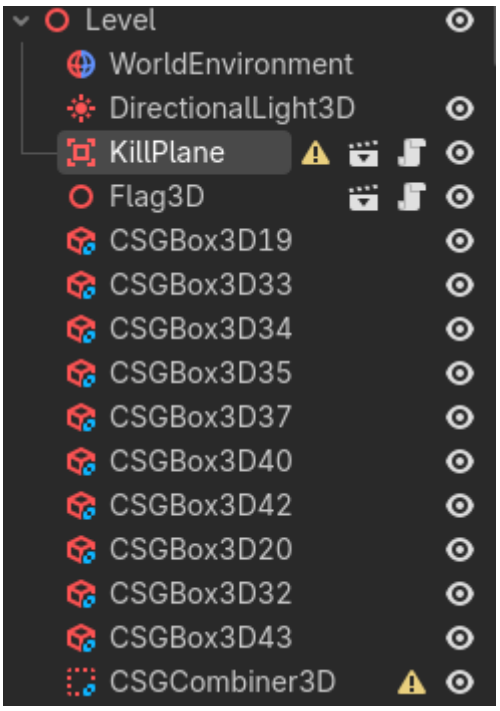

Figura 4: Estructura d'un nivell de Skyline Chase

2.4.1 El nucli del motor i la interacció de l'usuari

A la base de la piràmide hi trobem el motor natiu, **Godot Engine**, que actua com el ciment que ho aguanta absolutament tot. El motor s'encarrega de la part de computació més pesada: el renderitzat dels gràfics 3D, el càlcul de les caixes de col·lisió o també anomenades "hitboxes" per comprovar si el personatge està tocant el terra i la simulació de les lleis físiques que regeixen els salts i la gravetat. Just a sobre d'aquesta capa de baix nivell, hem programat el mòdul de control del personatge. Aquest sistema actua com un intèrpret: rep els *inputs* o senyals elèctrics que el jugador envia quan prem el teclat o el comandament, i els tradueix immediatament en vectors de moviment per al personatge i en rotacions precises per a l'angle de la càmera en terçera personal

2.4.2 La lògica de joc i el sistema de mòduls independents

Més enllà del moviment bàsic, la lògica interna del joc s'estructura en diferents mòduls independents que s'encarreguen de dictaminar què passa a la pantalla mentre s'executa la partida:

- **Mòdul de gestió de nivells:** És l'administrador d'escenes que s'encarrega de carregar i descarregar de la memòria RAM el mapa que toca a cada moment, controlant les transicions d'una pantalla a una altra.
- **Mòdul d'intel·ligència artificial (IA):** Controla el comportament dels enemics, obligant-los a fer rutes de patrulla definides i a reaccionar de forma automatitzada si el jugador es creua en el seu camí.
- **Component de control d'estat:** És el "cerebre" numèric de la partida. Gairebé sense que es noti, aquest script monitoritza constantment variables en temps real; és el que sap exactament quantes estrelles porta recollides el jugador en aquell nivell, si s'ha activat una *KillZone* i quants punts de vida s'han de restar de la barra d'interfície abans d'executar un reinici.

2.4.3 La persistència i el flux de dades

Finalment, hi ha una peça invisible per a l'usuari final, però que és la clau de volta de tota la nostra arquitectura: el **sistema de persistència de dades**. No volíem un joc clàssic tipus *arcade* on en tancar la sessió es perdés tot el progrés. Per això, vam connectar el codi de Godot amb una base de dades local mitjançant un sistema de guardat. Cada vegada que els personatges toca la bandera de final de nivell, el joc envia una consulta o instrucció per registrar de manera definitiva les estrelles obtingudes i el nivell desbloquejat. Així, quan el jugador torna al menú principal o obre el joc l'endemà, el sistema llegeix aquests fitxers i actualitza la interfície gràfica amb la informació correcta. Tot aquest entramat de sistemes interconnectats és el que garanteix que l'experiència final de Skyline Chase sigui fluida, no generi conflictes de codi i funcioni exactament com s'espera d'un programari seriós.

2.5 Descripció dels components

Per evitar que el desenvolupament de **Skyline Chase** se'ns anés de les mans amb un codi desordenat, inintel·ligible i ple de dependències creuades, ens ho vam plantejar des de l'inici una estructura basada en peces i mòduls totalment independents. La nostra intenció era fugir a tota costa del típic *script* gegant (l'anomenat "[2]codi espagueti") on tot està barrejat i on un simple canvi acaba provocant errors en cadena per tots costats. En el seu lloc, hem aprofitat al màxim la jerarquia de nodes i escenes que ofereix el motor **Godot Engine** per aplicar el principi de responsabilitat única: cada element s'encarrega exclusivament de la seva funció.

Aquesta decisió de disseny ens ha facilitat moltíssim la vida durant la fase de *debugging*: si de cop i volta el personatge deixava de saltar recte o reaccionava malament a les col·lisions, sabíem exactament en quin fitxer de **GDScript**[4] havíem de buscar l'error, sense la por constant d'haver trencat el sistema de guardat de les estrelles o el comportament dels menús per la línia de codi modificada.

A continuació, detallem com hem estructurat i desenvolupat aquests blocs principals perquè el videojoc funcioni de manera fluida, estable i eficient:

2.5.1 Sistema de moviment del personatge

Aquesta part és, sens dubte, la que més hores i maldecaps ens ha costat de tot el projecte, ja que teníem molt clar que si els personatges no es movien de manera intuïtiva, tot el joc perdria el seu valor de cop. Ens hem matat a polir el control de les físiques vectorials perquè la resposta del personatge als *inputs* del teclat fos immediata, evitant qualsevol sensació de retard o de pes fals en els salts.

A part de programar les accions bàsiques de caminar i saltar, ens vam motivar i vam voler anar un pas més enllà afegint mecàniques més avançades com el **doble salt** i el **wallslide** (el lliscament controlat per les parets). Estem convençuts que aquestes mecàniques fan que l'experiència sigui molt més dinàmica i divertida; d'aquesta manera, el jugador no es limita a avançar en un pla horitzontal de dreta a esquerra, sinó que es veu obligat a jugar constantment amb la verticalitat de l'escenari i a calcular els temps de reacció a l'aire.

2.5.2 Sistema de nivells

Aquí és on hem muntat, estructurat i ordenat tot el contingut interactiu que el jugador es va trobant al llarg de l'aventura. Aquest sistema s'encarrega de controlar activament el comportament de tots els elements de l'escenari: des de les plataformes estàtiques fins a les plataformes mòbils que segueixen una ruta amb keyframes, passant per aquelles superfícies que actuen com a trampes i que perden la col·lisió de cop.

Hem posat molts esforços a equilibrar la distribució d'obstacles, enemics i recompenses per aconseguir que el camí fins a la bandera final de cada nivell suposi un repte exigent pel jugador, però sense arribar mai a ser frustrant o impossible de superar. Bàsicament, aquest mòdul assegura que cada element de l'entorn reaccioni exactament quan i com toca segons la posició dels personatges.

2.5.3 Sistema de col·leccionables

Dins de cada nivell hem repartit una sèrie d'estrelles amb una filosofia de disseny molt clara: evitar que el joc es convertís en una simple carrera rectilínia cap a la meta. Les hem col·locat deliberadament en punts estratègics, racons amagats o zones de difícil accés per obligar qui juga a explorar l'entorn tridimensional i a treure el màxim profit de les habilitats del personatge, com el doble salt combinat amb el lliscament.

Des del punt de vista de la lògica del joc, aquestes estrelles són totalment opcionals és a dir, pots completar el nivell sense recollir-ne cap, però creiem que li donen molta més vida, riquesa i un punt extra de motivació al projecte. Tècnicament, hem programat un sistema de detecció d'àrees amb uns nodes de (*Area3D*) que, quan detecta la col·lisió amb el cos de cualsevol personatge, activa una funció que recull l'objecte, fa pujar el comptador de la interfície i envia la dada al sistema perquè es quedi guardada de forma definitiva.

2.5.4 Sistema de vides

Per donar-li tensió a la partida i evitar que el joc fos un passeig monòton on morir no tingués cap tipus de conseqüència, vam implementar un sistema de control d'estat amb un límit estricte de **tres vides**. Aquest mòdul monitoritza constantment l'estat del personatges:

vigila si entra en col·lisió amb els vectors d'atac d'un enemic o si cau per un forat del mapa directament cap a una KillZone.

Si el jugador comet un error tres vegades i el comptador intern arriba a zero, el sistema bloca les accions, executa l'animació de derrota i reinicia automàticament el nivell des del principi. Creiem que el número tres és el balanç perfecte dins del disseny de joc: ofereix un marge d'error just per aprendre dels teus propis fallos, però t'obliga a jugar amb cap, a fixar-te en els patrons dels enemics i a no saltar mai a la babalà.

2.6 Definició de les tasques

En aquest apartat expliquem com hem anat muntant les peces del joc perquè tot el que havíem pensat sobre el paper acabés funcionant dins de Godot. No ha estat només "picar codi", sinó buscar la millor manera de lligar cada sistema.

2.6.1 Sistema de Moviment i Físiques

Perquè el personatge es mogui com cal, hem fet servir el node **CharacterBody3D**. La clau per no tornar-nos bojos amb el codi ha estat crear una "màquina d'estats". Bàsicament, el programa mira tota l'estona si el cavaller està a terra, a l'aire o fregant una paret; segons on sigui, el **GDScript**[4] ens deixa fer el doble salt o el wallslide. Ara mateix, aquesta part la tenim **tancada i funciona súper bé**.

2.6.2 Sistema de Col·leccionables i SQLite

El tema de les estrelles ha estat un repte. No només volíem que desapareguessin en tocar-les, sinó que el joc recordés quines portaves. Per fer-ho, hem creat un script que, quan detecta la col·lisió, envia la dada directament a **SQLite**. Així, quan tornes al menú de nivells, ja surten les estrelles que has suat per aconseguir. Està **implementat al 100%**.

2.6.3 Sistema de vides i reinici del joc

Aquí no ens hem complicat gaire: tres vides i punt. Hem creat unes **"KillZones"** (com el buit o les zones de punxes) que, si les toques, el codi t'envia directament a l'inici de la pantalla i et resta una vida. Si el comptador arriba a zero, fem que el nivell es recarregui del tot. Aquesta part està **acabada** i és el que fa que hagis d'anar amb compte si no vols repetir-ho tot.

2.6.4 Funcionament dels enemics i les seves rutes

Els enemics funcionen amb una lògica de patrulla bastant senzilla però efectiva. Hem fet servir uns limits en les vores de les plataformes si l'enemic detecta que que té un mur davant, gira i segueix cap a l'altre costat. També hem programat que, si l'ataques i li treues tota la seva vida mori i s'elimini de l'escena. Està **tot llest i operatiu**.

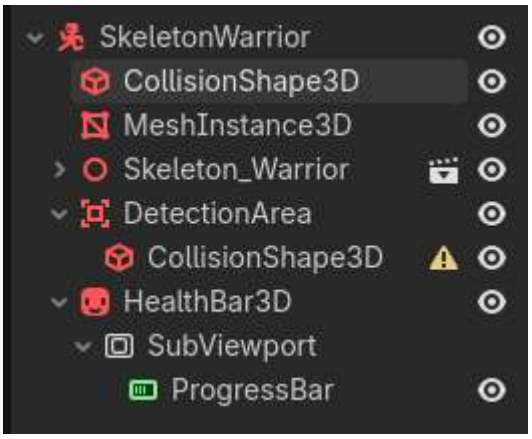

Figura 5: Estructura dels enemics del videjoc

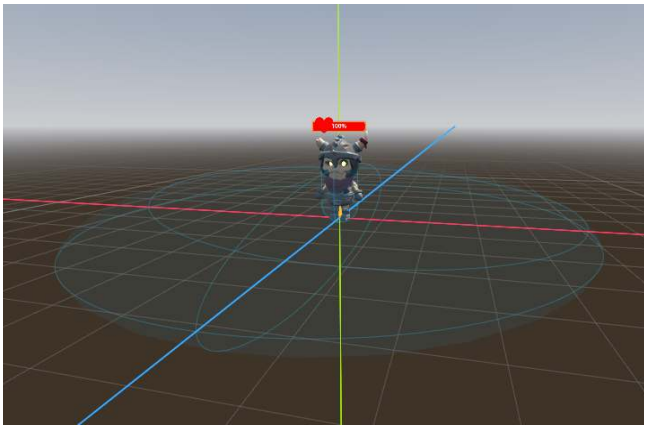

Figura 6**:** Foto enemic amb el seu àrea de detecció de personatge

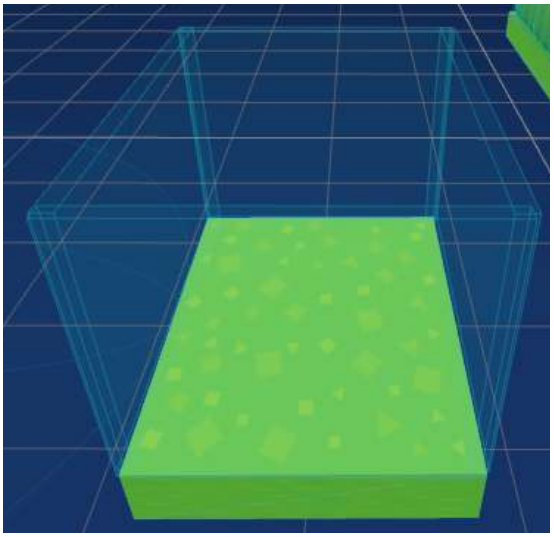

Figura 7: Estructura de collisions per els enemics

2.6.5 Plataformes que es mouen i obstacles

Perquè el joc tingués més "xixa", hem posat plataformes que es mouen. El més difícil va ser que els personatges no rellisqués en estar a sobre d'una plataforma en moviment, però ho vam solucionar amb les animacions de Godot. També n'hi ha algunes que cauen al cap d'un segon de trepitjar-les. Aquest sistema d'obstacles està **totalment integrat**.

2.6.6 Pantalles de menú i interfície visual

Aquesta és la part que lliga el joc. Hem muntat un **HUD** que t'ensenya a cada moment les vides i estrelles que portes. El menú per triar nivell també s'ha portat la seva feina, ja que ha de demanar a la base de dades quins nivells has passat per desbloquejar els següents. Ara mateix, tot aquest sistema de menús **funciona perfectament**.

Figura 8: Foto selector de nivell

2.6.7 Elecció de personatges

Finalment, hem deixat llesta la pantalla on pots triar el personatge. El que fem és canviar el model visual del protagonista es a dir el model de personatge però mantenint la mateixa lògica de moviment. D'aquesta manera, l'experiència és igual de justa triis qui triis. Aquesta part ja està **enllestida del tot**.

2.7 Definició de les funcionalitats

En aquest apartat explicarem com hem passat de les idees a la pràctica. No ha estat un camí directe; hem hagut de fer moltes proves a Godot fins que cada peça ha encaixat al seu lloc.

2.7.1 Moviment i control del personatges

Per a nosaltres, el més important era que el personatge no se sentís "pesat". Volíem un control àgil, així que vam agafar el node CharacterBody3D i el vam inflar a línies de GDScript[4]. El sistema que hem muntat sap on està els personatges a cada moment: si nota que toca una paret, t'activa el wallslide per lliscar, i si prems el salt mentre ets a l'aire, et deixa fer el segon impuls. Ha estat un repte deixar-ho ben polit, però ara mateix el control ja està tancat i funciona de luxe.

2.7.2 Sistema de vides i "Game Over"

No volíem que el joc fos un passeig, així que vam decidir posar només **tres vides** per posar una mica de pressió al jugador. El truc ha estat crear unes zones que anomenem KillZones (els forats i les trampes) que, en detectar al personatge utilitzat, li treuen un punt de vida i el tornen a l'inici del nivell de cop. Si la pifies tres vegades i arribes a zero, el joc es reinicia sol perquè ho hagis de tornar a intentar. Aquesta part ja està **totalment operativa**.

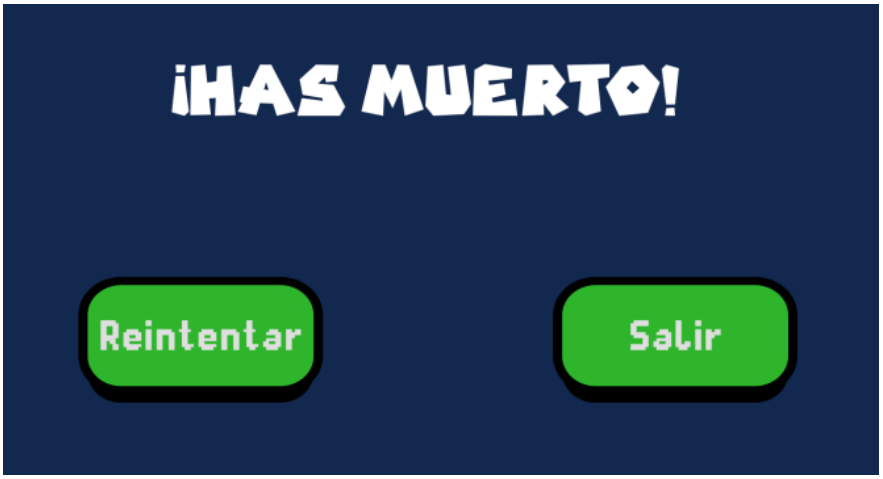

Figura 9: Pantalla de mort del videojoc

2.7.3 Col·leccionables i memòria (SQLite)

Una de les coses que més feina ens ha portat ha estat el tema de la memòria del joc. No volíem que les estrelles fossin només objectes que desapareixen, sinó que el joc recordés quines has agafat. Vam muntar un sistema amb **SQLite** perquè, cada cop que en reculls una, el senyal es guardi a la base de dades. Així, quan tornes al menú de selecció, el joc "recorda" el teu progrés de veritat i t'ensenya els teus rècords. Està **implementat al 100%**.

```
func guardar_progreso(nivel:int, vidas:int):
    db.query("DELETE FROM progreso;")
    db.query_with_bindings(
        "INSERT INTO progreso (nivel, vidas) VALUES (?, ?);",
        [nivel, vidas]
    )
func cargar_estrellas(nivel: int) -> int:
    db.query_with_bindings(
        "SELECT cantidad FROM estrellas WHERE nivel = ?;",
        [nivel]
    )
    if db.query_result.is_empty():
        return 0
    return db.query_result[0]["cantidad"]

func cargar_todas_estrellas() -> Dictionary:
    db.query("SELECT nivel, cantidad FROM estrellas;")
    var resultado := {}
    for fila in db.query_result:
        resultado[fila["nivel"]] = fila["cantidad"]
```

Figura 10: Estrelles i codi de la base de dades

2.7.4 Disseny de nivells i obstacles

Hem dissenyat cada nivell com si fos un trencaclosques, barrejant plataformes que es mouen i altres que cauen per fer-ho més entretingut. Tot acaba en una bandera que, en tocar-la, et canvia de pantalla o et torna al menú. També hem ficat enemics amb rutes senzilles perquè hagis de calcular bé on saltes. Aquest muntatge d'escenaris i trampes ja està **enllestit i llest per jugar**.

Figura 11: Diferents nivells de Skyline Chase

2.7.5 Selecció de personatges

Finalment, hem deixat llesta la pantalla on pots triar qui vols controlar. Bàsicament, el que fa el codi és carregar un model 3D o un altre segons el que triis al menú, però mantenint sempre les mateixes habilitats de salt perquè el joc no es torni més difícil per accident segons el personatge. Aquesta part visual ja està **completament finalitzada**.

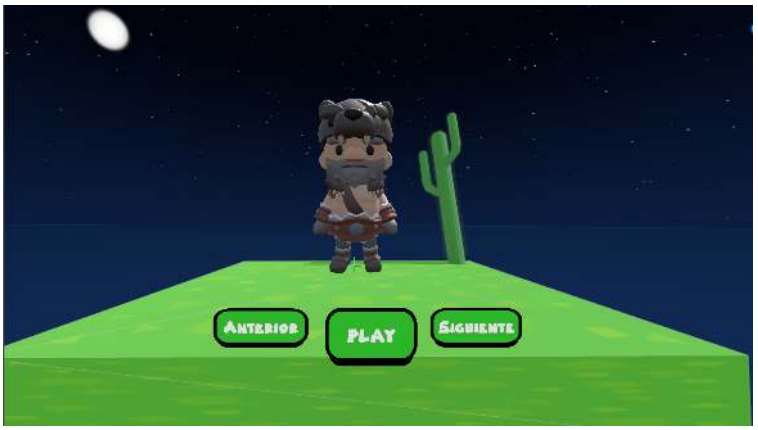

Figura 12: Pantalla del selector de personatges

3 Arquitectura del projecte

3.1 Estructura del sistema de fitxers

Abans de començar a crear nodes a la babalà, una de les primeres decisions de disseny que vam prendre va ser com organitzaríem la jerarquia de carpetes dins del directori res:// de Godot. Sembla una ximpleria, però en un projecte 3D amb tants actius diferents (models, textures, scripts, escenes, àudios), si no tens una estructura de fitxers neta des del primer dia, acabes perdent més temps buscant on has guardat un script que programant-lo.

Hem optat per una organització basada en la **naturalesa dels recursos**, però amb una separació clara per a les escenes principals. Aquesta és la lògica que hem seguit per a les carpetes del projecte:

- Carpeta d'Escenes (/escenas)**:** Aquí és on guardem els fitxers .tscn. Però no estan tots barrejats; ho hem dividit en subcarpetes com ***LVLS, GUI i Estrellas***. Això ens permet que, si hem de retocar el disseny del cavaller i dels demes personatges, anem directament a la seva carpeta i sabem que allà hi haurà l'escena del personatge i tot el que depèn d'ell.

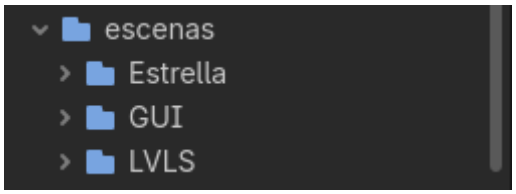

Figura 13: Carpeta on hi ha els nivells jugables

- Carpeta de Scripts (/Scripts): Tot el codi en GDScript[4] (.gd) està centralitzat aquí. I com abans també el tenim dividit en subcarpetes divits per funcionalitats com ***GUI*** (Interfaz de l'usuari), ***Platfroms*** (Movimiento de las plataformas), ***Player*** (Tots els scripts de moviment o mecaniques de tots els personatges) i ***Estrella*** (Tot sobre la estrella com el seu contador,moviment, etc.) estan divits d'aquesta manera per tal de tenir una visió clara de tota la lògica del joc. Dins d'aquesta carpeta, els scripts estan ordenats per la seva funció, els mes importants com son la base de dades (***DataBaseManager***) i el del joc principal (***game***) no estan en ninguna carpeta nomes estan dins de la de script.

Figura 14: Carpeta on hi ha els scripts

- Autoload (/autoload): Aquesta carpeta és fonamental per a l'arquitectura del joc. Conté els scripts que s'executen automàticament en iniciar el projecte o tambe anomenats "Singletons". Ens serveix per gestionar dades que han de persistir entre canvis de nivell, com ara la puntuació del jugador, el comptador d'estrelles o l'estat de les vides.

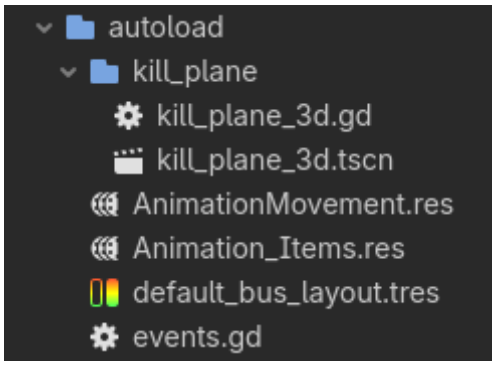

Figura 15: Carpeta on es troba el "Kill Plane"

- Fuentes i addons (/fuentes i /addons)**:** Finalment, la carpeta Fuentes conté totes les tipografies utilitzades per a la interfície d'usuari (UI), garantint una estètica coherent. A addons, hi guardem les extensions externes del motor que ens han ajudat a agilitzar el desenvolupament sense haver de programar funcionalitats genèriques des de zero como per exemple en el nostre cas **SQLite**.

Figura 16: Carpeta on estan les distintas tipografías i addons

- Player (/Players): Donat que el nostre joc es basa en un personatge principal en un entorn 3D, hem organitzat els models i les animacions en aquestes carpetes. Aquí gestionem des de les malles de tots els personatges que el jugador pot controlar fins als esquelets i totes les animacions de moviment. Tenir carpetes separades per als diferents tipus de personatges o enemics ens ajuda a localitzar ràpidament els actius 3D i les seves col·lisions.

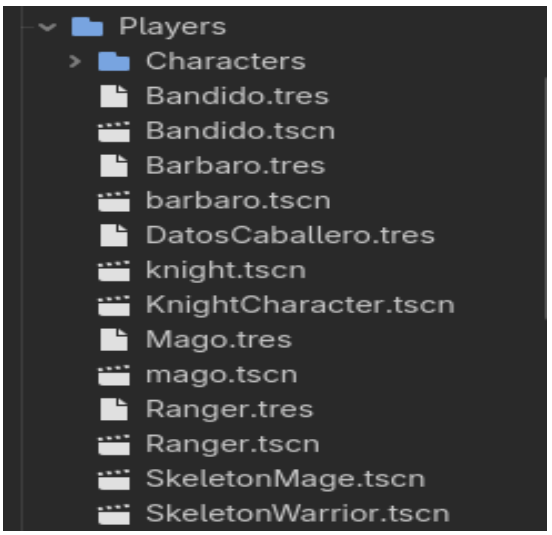

Figura 17: Carpeta on es troben tots els personatges

- Level (/level): És el cor visual del joc. Aquí no només guardem els escenaris, sinó que gestionem la il·luminació global, el cel (Sky) i els entorns. Aquesta carpeta conte tot tipus de decoracio usada al nivells i textures, també estan els nivells d'una forma natural es a dir esta el fitxer **.tscn** que es el que s'ha d'editar si es vol treballar o modificar el nivell.

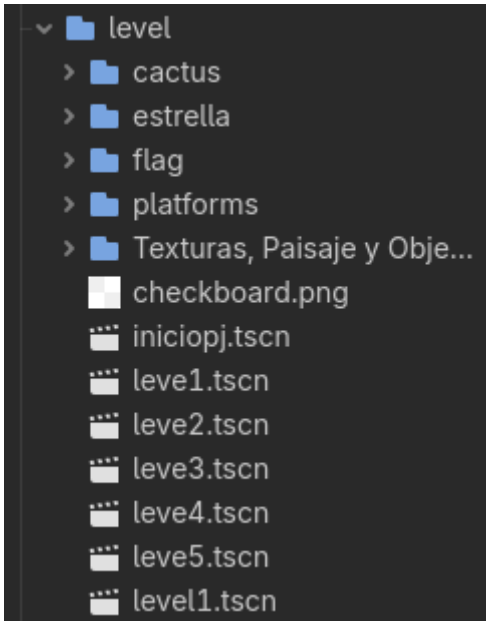

Figura 18: Carpeta on es troben els nivells editables

- WEB (/web): Aqui es gestiona tota la WEB del proejecte en aquesta podem torbar fins les instuccions de com es juga, el perque hem desenvolupat aquest projecte, la nostre planificació, una galeria, per probar-ho en una versió optimizada per HTML i un botó de descarrega el joc.

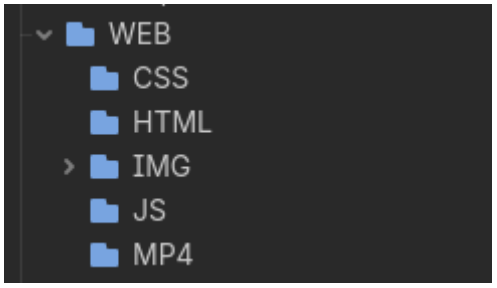

Figura 19: Carpeta on hi está ubicada la web

3.2 Jerarquia de les escenes

L'arquitectura de **Skyline Chase** s'ha construït seguint seriosament a la filosofia de disseny de Godot Engine, que es basa en una estructura de arbre amb directori principal (res://) que esta formada per un conjunt de nodes i escenes que poden ser heredades de altres. En lloc de programar el joc com un bloc de codi monolític i rígid, hem optat per un enfocament unic per cada tipus de situacio. Això vol dir que cada element del joc (els personatges, els enemics, les trampes o els col·leccionables) s'han desenvolupat i programat com una escena independent i totalment autònoma que després s'integra en l'escena dels nivells.

Aquesta jerarquia és fonamental per a la sostenibilitat del projecte. Per exemple, el nostre personatge principal (cavaller) no és un simple objecte dins d'un mapa, sinó que és una escena complexa que conté la seva pròpia lògica de moviment, les seves propies animacions i les mecaniques implementadas aquesta escena esta programada i editada fora de la principal que es la del nivell. L'escena del personatge te com a node pare un (**CharacterBody3D**). El seu esquelet d'animacions, la càmera que el segueix i els sensors de col·lisió. En tenir-ho així, podem fer canvis en el script del jugador o en la seva forma física i aquests es reflecteixen immediatament a tots els nivells. Aquesta capacitat d'herència estalvia una quantitat ingent de temps de desenvolupament i redueix a zero el risc de tenir comportaments diferents del personatge en funció de la pantalla on es trobi.

Un altre factor determinant en l'arquitectura del nostre projecte és el pes que hem donat a la modularitat durant tot el procés de disseny dels nivells. En lloc de tractar cada obstacle com un objecte únic i aïllat, hem optat per crear un sistema de "templates" o tambe anomenades escenes base. Aquest mètode ens permet, per exemple, que una plataforma mòbil no hagi de ser programada des de zero cada vegada que apareix en un mapa; el que fem és una **instanciació** de l'escena original, fet que ens permet reutilitzar tot el codi de moviment i les propietats físiques ja definides.

Aquesta metodologia de treball no és només una qüestió d'estalvi de temps, sinó que respon a una necessitat tècnica d'optimització. En utilitzar instàncies d'una mateixa escena, el motor de joc pot gestionar la memòria de manera molt més eficient, ja que només ha de carregar la lògica principal un sol cop i després aplicar les modificacions que hem aplicat (com la posició, la velocitat o el recorregut) a cada còpia. Això ens permet alliberar càrrega de treball tant a la **CPU** com a la **GPU**, un aspecte absolutament crític en un joc 3D on el càlcul de col·lisions en temps real, el renderitzat de objectes 3D suposen un cost computacional molt elevat.

Finalment, aquesta jerarquia de nodes ens facilita enormement la depuració d'errors. Si detectem per exemple que una estrella no s'agafa bé, sabem que l'error està localitzat en l'escena específica de l'objecte i no en el codi global del nivell.

3.3 Gestió d'esdeveniments i comunicació

L'arquitectura interna del nostre projecte no s'hauria pogut realitzar de manera eficient sense un sistema de comunicació sòlid. Per evitar el que en programació anomenem [2]***codi espagueti***, hem basat tota la lògica d'interacció en els **Signals** de Godot. L'ús d'aquesta metodologia ha facilitat la implementació d'un disseny basat en el desacoblament de components. Mitjançant aquest sistema, els objectes emeten notificacions davant qualsevol esdeveniment rellevant, permetent que altres mòduls reaccionin de forma independent. Això elimina la necessitat que l'emissor conegui la identitat o la ubicació del receptor, garantint una estructura molt més neta i modular.

Aquesta manera de treballar és la que fa moure tota la lògica del **Kill Plane** (el buit on el jugador mor) i de la **Meta**. Per exemple, quan el personatge toca l'àrea de la meta, no s'executa el canvi de nivell allà mateix de forma rígida. El que passa és que la meta emet un senyal que viatja fins a un script centralitzat anomenat **Events**. Aquest node funciona com un diccionari global d'esdeveniments. Gràcies a això, podem fer que al mateix moment passin tres coses diferents: que el **ScenesManager** prepari el següent nivell, que la interfície (UI) mostri una escena de victòria i que el sistema de so reprodueixi un efecte, tot sense que aquests tres sistemes hagin de dependre directament l'un de l'altre.

El cervell que gestiona totes aquestes ordres és el **ScenesManager**, que tenim configurat com un **Singleton o Autoload**. El GameManager s'encarrega del canvi d'escenes i de la persistència de les dades. Això és clau perquè, en un motor com Godot, quan canvies de nivell, tot el que hi ha a la memòria de l'escena anterior s'esborra. En tenir el GameManager fora de l'arbre de nivells, podem guardar informació crítica i important, com pot ser el número de vides restants o les estrelles que s'han recollit, per després poder mostrar-les correctament al selector de nivells.

Aquesta estructura de senyals també la fem servir per a tota la part dels botons i els menús del joc. Quan l'usuari prem un botó al menú de selecció o al de pausa, el botó envia un senyal `pressed()` que l'script de l'escena rep per saber què ha de fer a continuació: si ha de canviar d'escena o continuar amb el joc. Aquesta arquitectura ens ha aportat uns avantatges tècnics molt destacables:

- Robustesa**:** Si, per exemple, eliminem una estrella o un obstacle del nivell per fer una prova, el joc no es bloqueja ni dóna errors fatals de referències inexistents. El senyal simplement s'emet i, si ningú el rep, el programa segueix funcionant amb normalitat.

- Optimització real**:** El processador no ha de comprovar cada mil·lisegon si el jugador ha mort o si ha arribat al final. El codi només s'activa exactament quan es produeix l'esdeveniment, evitant així que l'ordinador gasti recursos constantment per verificar aquests processos.

- Facilitat de modificació**:** Com que tots aquests processos estan controlats de manera centralitzada, si volguéssim modificar qualsevol procés o afegir-ne de nous, seria tan fàcil com crear un nou senyal a l'script general. D'aquesta manera, no hi ha risc de fer malbé una part del codi que ja funciona correctament.

El nostre script de events

```
1   extends Node
2
3   signal kill_plane_touched(body: PhysicsBody3D)
4   signal flag_reached
5   signal player_attack(damage: int)
6   signal player_damaged(damage: int)
7   signal estrella_recogida_global
8   signal player_healed
```

Figura 20: Script dels events del videojoc

Script que controla el videojoc

```
extends Node

var nivel_actual := 1
var ultima_escena_jugada : String = ""
var proximo_nivel : String = ""
var siguiente_numero = nivel_actual + 1
var selected_character: CharacterData
signal game_paused(paused: bool)

func pause_game(paused: bool) -> void:
    if get_tree().paused == paused:
        return
    get_tree().paused = paused
    game_paused.emit(paused)

func ir_a_nivel(numero: int, path: String) -> void:
    Input.mouse_mode = Input.MOUSE_MODE_HIDDEN
    nivel_actual = numero # <--- ESTO ES LA CLAVE
    get_tree().change_scene_to_file(path)

func ir_a_selector() -> void:
    Input.mouse_mode = Input.MOUSE_MODE_VISIBLE
    get_tree().change_scene_to_file("res://escenas/GUI/SelectorNivel.tscn")

func ir_a_victoria():
    Input.mouse_mode = Input.MOUSE_MODE_VISIBLE
    get_tree().change_scene_to_file("res://escenas/GUI/victoria.tscn")
```

Figura 21: Script que fa control del videojoc

Senyals

```
func _on_salir_pressed() -> void:
    get_tree().quit()

func _on_start_game_pressed() -> void:
    get_tree().change_scene_to_file("res://escenas/GUI/SelectorNivel.tscn")

func _on_personajes_pressed() -> void:
    get_tree().change_scene_to_file("res://escenas/GUI/SelectorPersonajes.tscn")

func _on_tutorial_pressed() -> void:
    get_tree().change_scene_to_file("res://escenas/LVLS/tutorial.tscn")
```

Figura 22: Script amb algunas senyals

3.4 Arquitectura de la Interfície d'Usuari i sistemes de control

Dins de l'arquitectura del nostre joc, hem hagut de gestionar com es mostren les dades al jugador (com el nombre de vides o les estrelles) sense que aquestes interfereixin amb el món 3D. Per fer-ho, hem utilitzat diferents mecàniques que, com la resta del joc, formen part d'una escena instanciada. També hem utilitzat un sistema de nodes de tipus **CanvasLayer**. Aquest node és vital perquè crea una capa de dibuix independent; tot el que hi ha dins del CanvasLayer es renderitza sempre per sobre de la càmera del joc, assegurant que la interfície (HUD) estigui sempre visible i fixa a la pantalla. Tot aquest HUD es pinta en una dimensió 2D, ja que es tracta d'objectes plans.

Per a la construcció dels menús i el HUD, hem evitat col·locar els elements de forma manual en posicions fixes, ja que això causaria problemes de visualització en diferents monitors o resolucions. En el seu lloc, hem utilitzat una jerarquia de **Nodes de Control** basada en contenidors dinàmics:

- **Margin Containers:** S'utilitzen per assegurar que els textos i icones mantinguin un marge de seguretat respecte a les vores de la pantalla. És a dir, si el joc està en mode finestra, es respecten els marges perquè tot es vegi correctament.
- **VBox Containers:** Serveixen per organitzar els botons i les icones de forma automàtica. Aquesta elecció arquitectònica és clau per a la responsivitat del joc; si el jugador canvia la resolució o posa el joc en pantalla completa, els elements es reajusten sols sense deformar-se.

La comunicació d'aquesta capa visual amb la lògica del joc es fa de manera passiva. És a dir, la interfície no "pregunta" constantment al sistema quantes vides li queden al jugador, ja que això consumiria recursos innecessaris i podria alentir el joc. En lloc d'això, la interfície es queda a l'espera que el **GameManager** o el node d'**Events** emeti un senyal de canvi per actualitzar-se i mostrar la nova informació. Quan el jugador perd una vida o recull una estrella, la UI rep el senyal i actualitza només el node visual corresponent (com la barra de vida o la puntuació).

Abans: **Després:**

Abans: **Després:**

Figura 23: Fotos que mostren les estrelles i els cors

Finalment, també hem implementat el sistema del menú de pausa que, com la resta d'elements, s'ha integrat en aquesta estructura d'una forma instanciada i programada en una escena a part. En aquest menú utilitzem la propietat **"Process Mode"**. Quan el jugador prem la tecla "ESC", enviem un senyal que posa el node del nivell 3D en estat *Paused*, bloquejant el moviment i les col·lisions, mentre que el **CanvasLayer (2D)** del HUD té assignat el procés *Always*. Això permet que l'usuari pugui seguir movent el ratolí i interactuant amb els botons de "Continuar" o "Sortir" mentre el món 3D es mante congelat.

Figura 24: Foto del menu de pausa

4 Conclusions

4.1 Conclusions generals del projecte

La veritat és que ara que ja hem tancat **Skyline Chase**, ens n'adonem que fer un videojoc des de zero ha estat un embolic molt més gran del que ens pensàvem allà al gener. Al principi ens pensàvem que la feina era només fer que el personatge es mogués i saltés per la pantalla, però ens hem trobat amb mil detalls de codi i de gestió de dades que no es veuen a simple vista i que ens han portat de cap durant setmanes.

A nivell de classe, el projecte ens ha anat de perles per aprendre a dominar Godot i el llenguatge GDScript[4] de veritat. Sobretot, hem perdut la por a coses que ens feien respecte al principi, com haver de muntar la base de dades amb SQLite per desar les estrelles.

Però si hem de ser sincers, el que més ens emportem de tot plegat és haver après a currar en equip. Ens hem hagut de repartir els marrons, ajudar-nos quan l'altre s'encallava amb un script i prendre totes les decisions a mitges. Veure que ara el joc funciona, que respon bé i que a sobre és divertit després de tantes hores tancats davant la pantalla, ens dona un subidón important. Ha estat dur i hem tingut moments de voler tirar la tovallola, però veure el resultat final fa que tot l'esforç hagi valgut la pena.

4.2 Consecució dels objectius

Si mirem enrere i comparem el resultat final amb els objectius que vam escriure al principi de tot, podem dir ben feliços que hem complert gairebé tot el que teníem al cap, tot i que pel camí hem hagut de fer algun retoc.

Per exemple, el tema del moviment del personatge era el nostre gran repte de la llista. Volíem que el cavaller es mogués ben fi, i després de passar-nos hores i hores fent proves i tocant variables, el doble salt i el wallslide han quedat clavats i funcionen molt millor del que esperàvem.

D'altra banda, tot el sistema de les estrelles i la connexió amb SQLite ens feia bastant respecte al principi, però al final ho hem pogut treure endavant sense problemes. Les dades es guarden la mar de bé i la base de dades no falla mai, que era el que més por ens feia de tot el treball.

Finalment, pel que fa als nivells, hem aconseguit dissenyar les pantalles de manera que la dificultat vagi pujant a poc a poc a mesura que avances, tal com havíem planejat en els objectius inicials.

4.3 Valoració de la metodologia i planificació

Si hem de ser sincers, el calendari que vam planificar al principi de tot ens ha servit de ben poc. Ens vam confiar bastant i no vam preveure que algunes coses aparentment senzilles, com polir les col·lisions perquè el personatge no es quedés clavat o fes coses rares contra les parets, ens acabarien robant moltíssimes més hores de les que havíem calculat. Això ens va obligar a canviar de plans i a anar improvisant la manera de treballar una mica sobre la marxa.

La veritat és que el que ens ha salvat el projecte ha estat la flexibilitat i el suport mutu entre nosaltres. Quan un de l'equip es cansadissa d'un script o es bloquejava amb un error de codi, l'altre s'ho mirava amb ulls nous per trobar la solució. Treballar així ens ha permès tirar endavant, encara que haguem hagut d'accelerar al màxim i tancar moltes hores a última hora per poder arribar a temps a la data de l'entrega.

4.4 Visió de futur

Tot i que ja hem donat per acabada aquesta primera versió del videojoc, nosaltres van tenim molt clar que ens agradaria continuar treballant-hi a futur. Ens han quedat moltes idees al tinter que no hem pogut posar ara, en part per la falta de temps que hem tingut a l'últim tram i en part per culpa d'aquella mala planificació del principi.

El primer que volem fer és acabar de polir el sistema de combat. Ara mateix tenim l'animació de l'atac, però la nostra intenció és que les armes facin mal de veritat i que es puguin fer diferents tipus de combos o atacs segons el polsador. A més, com que la base tècnica del disseny de nivells ja és sòlida i funciona, ens morim de ganes de crear nous mons. Hem pensat a fer un món de neu on el terra rellisqui i afecti les físiques del cavaller o dels personatges, o una zona de lava on hagis de calcular al mil·límetre on saltes.

Una altra assignatura pendent és l'apartat visual. Creiem que podem treballar molt més les textures del joc i jugar amb la il·luminació de Godot per aconseguir que tot plegat es vegi molt més professional i entri millor pels ulls. Finalment, una idea que ens fa molta il·lusió és intentar fer el port per a dispositius mòbils. Sabem que Godot posa les coses bastant fàcils per exportar a Android, i creiem que pel tipus de joc de plataformes que és **Skyline Chase**, tindria molt de potencial per jugar-se amb controls tàctils en una pantalla de mòbil o en una tauleta.

5. Glossari

Godot Engine: És el motor de videojocs que vam escollir per dissenyar tot l'entorn en 3D del nostre projecte, Skyline Chase, principalment perquè és programari lliure, multiplataforma i de codi obert.

GDScript[4]: El llenguatge de programació que utilitza de forma nativa aquest motor. Té una estructura que ens ha recordat molt a Python i l'hem fet servir per escriure tota la lògica i el funcionament del joc.

CharacterBody3D: Un tipus de node clau dins de Godot que està pensat per a objectes que es mouen mitjançant línies de codi; en el nostre cas, ens ha servit per programar des de zero el control i les físiques del personatge.

Wallslide: Una de les mecàniques de moviment que hem programat per als nostres personatges, la qual li permet enganxar-se temporalment a les parets verticals del mapa i lliscar cap avall a poc a poc.

KillZone: Una àrea que hem col·locat de forma invisible a la part inferior dels escenaris i a les zones de perill perquè, si el jugador comet un error i hi cau, el sistema li resti una vida i el reubiqui al darrer punt de control.

GUI / HUD: Són les sigles de la interfície gràfica que veu l'usuari directament a la pantalla mentre juga. Al nostre projecte, l'hem dissenyat per mostrar en temps real les vides en forma de cors i el comptador de les estrelles que es van recollint.

SQLite: Una base de dades interna i molt lleugera que vam integrar per desar el progrés a nivell local. Ens serveix perquè el joc recordi quins nivells s'han superat i quantes estrelles porta guardades el jugador sense haver de dependre d'un servidor extern.

6. Bibliografia

[1] Godot Engine. (2026). *Asset Library (Llibreria de recursos i connectors per al motor)*. Recuperat el 17 de maig de 2026, de https://godotengine.org/asset-library/asset

[2] Godot Engine. (2026). *Official Blog (Informació de desenvolupament i actualitzacions del motor)*. Recuperat el 17 de maig de 2026, de https://godotengine.org/blog/

[3] Godot Engine. (2026). *GDScript Documentation and Reference Guide*. Recuperat el 17 de maig de 2026, de https://gdscript.com/

[4] CristianDev. (2025). Cómo hacer un juego 3D en Godot 4: Tu Primer Proyecto . Recuperat el 10 de novembre de 2025, de https://www.youtube.com/watch?v=EToPfWz9YZ0

[5] PDEProgramación. (2024). Animaciones usando el Nodo AnimationTree [Vídeo]. YouTube. Recuperat el 17 de novembre de 2025, de https://www.youtube.com/watch?v=0GXJsh9RBAQ

[6] Feles Machina. (2025). Godot Animation Expressions (loo-code animation transitions!) . Recuperat el 17 de novembre de 2025, de https://www.youtube.com/watch?v=iosqXfNdJVw

[7] LuisCanary. (2025). Start MENU en Godot/Main Menu/ Facil y Sencillo para 3D y 2D [Vídeo]. YouTube. Recuperat el 17 de maig de 2026, de https://www.youtube.com/watch?v=hyTu5ZTR5Yk

[8] Hwaci. (2026). *SQLite Official Website and Documentation*. Recuperat el 17 de maig de 2026, de https://www.sqlite.org/

[9] Google. (2026). *Gemini (Model de Llenguatge Intel·ligència Artificial)*. Recuperat el 14 de gener de 2026, de https://gemini.google.com

[10] Free3D. (2026). *Modelos 3D Gratuitos*. Recuperat el 5 de gener de 2026, de https://free3d.com/es/

[11] Lousberg, K. (2025). *KayKit - Adventurers Pack* [Models 3D]. Itch.io. Recuperat el 12 de novembre de 2025, de https://kaylousberg.itch.io/kaykit-adventurers

[12] Lousberg, K. (2025). *KayKit - Skeletons Pack* [Models 3D]. Itch.io. Recuperat el 12 de novembre de 2025, de https://kaylousberg.itch.io/kaykit-skeletons

[13] Lousberg, K. (2025). *KayKit - Character Animations* [Animacions 3D]. Itch.io. Recuperat el 12 de novembre de 2025, de https://kaylousberg.itch.io/kaykit-character-animations

[14] Viquipèdia. (2026). *Código espagueti (Antipatrons de programació)*. Wikipedia, La enciclopedia libre. Recuperat el 17 de maig de 2026, de https://es.wikipedia.org/wiki/C%C3%B3digo_espagueti

7. Annexos

ANNEX I: MANUAL D'USUARI

1. Introducció i Propòsit del Programari

Aquest manual té com a objectiu guiar l'usuari final a través de les mecàniques, controls i configuracions de **Skyline Chase**. El joc ha estat dissenyat com una experiència de plataformes en tres dimensions on la precisió en el salt i el control del personatge són els elements fonamentals per a l'èxit.

A diferencia de altres jocs d'aquest tipus, el jugador no te cap limitació de temps fent així que el jugador pugi analitzar tranquilament totes les decoracions i camins que s'han creat per així poder tindre temps per planificar que camí has d'agafar per aconseguir les estrelles o si vols anar directament a la meta.

L'objectiu real és la superació de reptes arquitectònics i la recollida d'objectes col·leccionables sense perdre les vides disponibles. A través d'aquesta guia, s'explicaran els procediments per instal·lar el joc, gestionar els personatges i entendre la interfície per garantir una experiència òptima.

2. Requisits de Hardware i programari

Com a projecte desenvolupat sota el motor **Godot Engine v.4.6**, el joc està optimitzat per funcionar en una àmplia gamma d'equips gràcies al treball realitzat en la compressió de textures i l'ús de malles eficients.

- Sistemas operatius actuals: Windows 10/11, distribucions de linux actuals.
- Minim 4GB de RAM per garantir el funcionament del videojoc.
- Espai al disc dur: amb uns 400MB de espai lliure ya tens espai.

3. Mecàniques del videojoc

A **Skyline Chase** hi ha com a 4 mecàniques més el moviment bàsic del personatge, les mekanikes amb què compta Skyline Chase són "**Wall slide**", "**Atacar**", "**Plataformes en moviment i invisibles**" i per últim **"Doble Jump**", aquestes són totes més mecàniques amb les quals compte el nostre videojoc i que per a nostre videojoc i que per a nosaltres.

4. Controls de Skyline Chase

Els controls bàsics de **Skyline Chase**, com ara moure's, saltar i colpejar enemics, són molt senzills, ja que són els mateixos que tenen la majoria de videojocs. Per exemple, a l'hora de controlar el moviment del personatge, es pot fer amb les tecles bàsiques **WASD**: la **W** és per anar cap amunt, la **A** cap a l'esquerra, la **S** cap enrere i la **D** cap a la dreta. Per fer que el personatge pugui saltar, la tecla que s'ha d'utilitzar és l'espai i, per últim, per a que el personatge colpegi els enemics, s'haurà de fer amb el **clic esquerre del ratolí**.

5. Interfaz del videojoc

5.1 Menu principal

El que veurà el jugador en iniciar el nostre videojoc serà una pantalla de menú principal amb les següents opcions: "**Start**", "**Tutorial**", "**Sortir**" i "**Personatges**". Cada botó té una funció diferent; per exemple, el botó "**Start**" et porta a un selector de tots els nivells del joc. El botó "**Tutorial**" és un nivell extra per conèixer totes les mecàniques amb què compta Skyline Chase i saber com s'utilitza cada una. El botó "**Sortir**" és un botó que tanca el videojoc i, per últim, el botó "**Personatges**" és un selector entre els cinc personatges disponibles. Si no tries cap personatge i prems directament el botó "**Start**", començaràs amb el jugador principal, que és el Cavaller.

5.2 Nivells

Quan el jugador entri a un nivell, ja sigui el que hagi triat o el primer després de seleccionar els personatges, el que veurà serà el nivell amb els seus cors a la part superior dreta i les seves estrelles a l'esquerra.

Figura 25: Foto de un nivell

ANNEX II: MANUAL D'INSTALACIO

1. Com instalar el videojoc

Per instalar **Skyline chase** hi ha dos maneras de instalar, les dos maneras estan al github, la primera es directament descarregant el .zip al github que hi esta tot el videojoc si vols editar algo del videojoc al **Goodot Engine** que podras editar todo el juego.

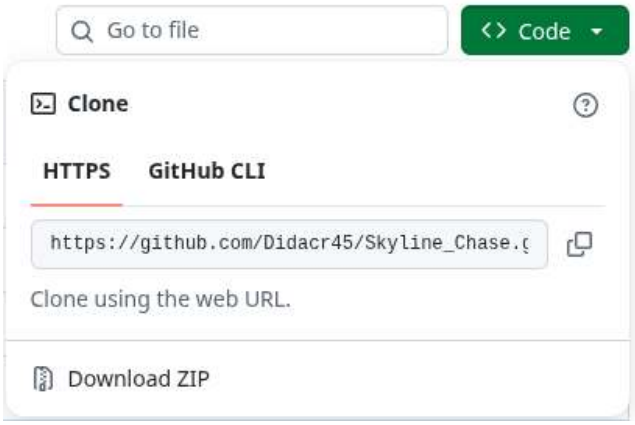

Figura 26: Foto del github

La segona manera de instalar el nostre videojoc es directament a les "**Releases**" per el teu sistema operatiu disponible, el videojoc estara disponible per **Windows** y **Ubuntu**, en aquest apartat estaran totas las versions disponibles del videojoc.

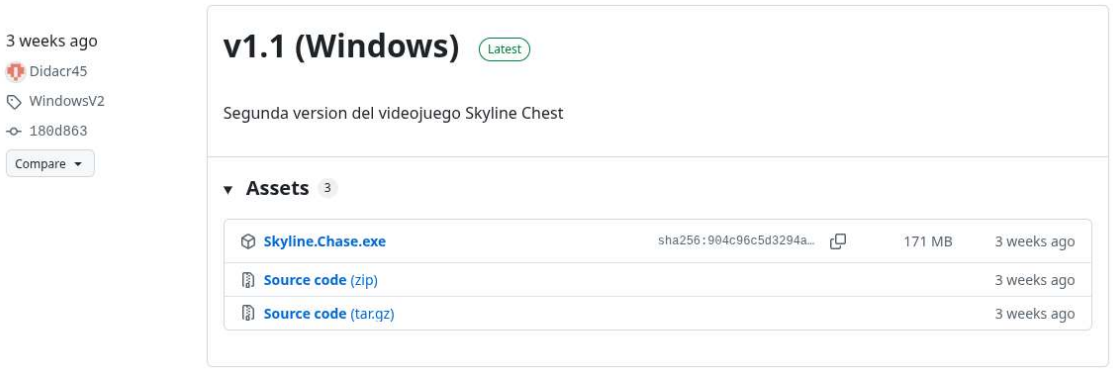

Figura 27: Foto de les "Releases" en el github

Despres hi ha una terçera manera de jugar al videojoc que es directament a la nostra pagina web ya que el videojoc tambe compta amb una versió disponible per paginas web.

Figura 28: Foto de la pagina web

Quan li dones clic al botó de “Jugar” entraras al videojoc per jugarlo directament.

8. Pie de documento

[1]sqlite: Gestor de base de dades relacional lleuger i autònom que s'executa localment en un sol fitxer de dades, sense la necessitat de dependre d'un programari servidor independent

[2]codi espagueti: Basicament es un metode on tot està connectat amb tot i al modificar alguna cosa es més probable trencar un altre.

[3]LUAU: Luau és un llenguatge de programació basat en Lua petit, ràpid i incrustat que afegeix un sistema de tipus gradual per millorar la seguretat i el rendiment del codi.

[4]GDSCRIPT: Llenguatge de programació d'alt nivell, dinàmic i optimitzat específicament per al motor Godot Engine. Utilitza una sintaxi molt similar a Python, la qual cosa facilita la seva lectura i aprenentatge