Universitat de Girona
Escola Politècnica Superior

Treball final de grau

Estudi: Grau en Disseny i Desenvolupament de Videojocs

Títol: Desenvolupament d'un videojoc de plataformes 2D amb habilitats

Document: Memòria

Alumne: Sergi González del Hoyo

Tutor: Gustavo Patow

Departament: Informàtica, Matemàtica Aplicada i Estadística

Àrea: Llenguatges i Sistemes Informàtics

Convocatòria (mes/any): Juny 2022

Índex

1. Introducció, motivacions, propòsit i objectius del projecte

1.1 Introducció i idea del joc

En general, en la majoria de videojocs en els quals l'objectiu del jugador és la superació de nivells i/o diferents escenaris, no sol ser gaire habitual que es premiï la creativitat i la innovació a l'hora de superar-los. El més habitual sol ser que es faci servir la estratègia dominant o la preferida pel jugador en funció del seu perfil.

En els videojocs de plataformes 2D aquesta dinàmica per part dels jugadors s'accentua encara més ja que la superació del nivell sol ser l'únic objectiu i les mecàniques i/o capacitats que té el jugador són les indicades pel nivell a completar.

Els jugadors acostumen a sentir-se atrets per aquests tipus de joc, ja que la forma de arribar fins a l'objectiu no és un (o dos) camin(s) rectes, sinó que cada jugador té la seva pròpia experiència. Això, a més, invita molt a la rejugabilitat i a tornar a viure una "nova" primera impressió per auto desafiar-se.

1.2 Objectius i motivacions

1.2.1 Profunditzar en el desenvolupament de mecàniques 2D

Per a la realització d'aquest videojoc haurem de profunditzar a fons en els videojocs de plataformes 2D. Necessitarem enriquir-nos en l'àmbit, veient com altres jocs del sector han exprimit aquest gènere i veure com podem fer viables totes les possibles mecàniques que aquests jocs ofereixen. Qualsevol idea nova que es pugui aplicar es tindrà en compte si és viable en aquest sector.

Figura 1.1: Super Mario Bros Deluxe

1.2.2 Entendre l'escenari amb el context d'aquestes mecàniques.

Per suposat, els nivells hauran de tenir en compte les possibles mecàniques, essent molt important que s'adaptin a aquestes en tot moment. La integració d'aquestes mecàniques en l'escenari és fonamental si volem tenir en compte el gran ventall de possibilitats que estem disposats a donar al jugador.

1.2.3 Posar-nos en el lloc del jugador tenint en compte el context

La diversitat i la innovació és el punt a reforçar i sobre el que ens recolzem per la idea de joc, però amb això s'ha de vigilar. Si aclaparem al jugador amb una gran quantitat de informació i mecàniques segurament acabi sentint que li és molt difícil acostumar-se a la dinàmica del joc.

Serà molt important tenir en compte com s'ensenya al jugador a jugar de forma introductòria i com, poc a poc, anem sumant afegits que insten a trobar aquesta diversitat i ganes de fer les coses de manera diferent. Aquest equilibri és molt important.

1.2.4 Implementar un prototip del joc

Un altre objectiu és implementar un prototip del joc o versió demo, on el jugador entengui la dinàmica. S'ha de presentar de manera correcta la dinàmica del joc, fent entendre que fer servir sempre les mateixes mecàniques no serà la manera més eficient. També haurem de presentar les mecàniques inicials i desenvolupar una primera idea de nivells que donin peu a la rejugabilitat dels mateixos, fent pensar al jugador quina pot ser la millor estratègia a l'hora de superar-los.

1.3 Quadre d'autovaloració

En el cas del videojoc complet, segurament, tot i que òbviament on es faria més ènfasis en el desenvolupament és en les mecàniques, segurament es podria fer d'una forma més repartida. Però en aquest cas que el videojoc ha estat desenvolupar per un sol estudiant clarament l'enfoc principal ha estat en el desenvolupament de mecàniques del joc.

A continuació es mostra una taula amb la valoració personal dels esforços realitzats en cadascun dels quatre elements del projecte:

Estètica	5%
Narrativa	0%
Mecàniques	55%
Tecnologia	40%

Clarament, com podem apreciar, el principal èmfasis ha estat en les mecàniques. No tan sols la programació i el seu funcionament, sinó la seva integració sobre l'entorn on es creen. Algunes hauran de ser semblants en la seva utilitat, ja que per invitar al jugador a que torni a jugar un nivell haurà de superar un mateix obstacle de manera diferent. Per tant necessitarem a part de desenvolupar-les, una gran quantitat de proves per verificar que són útils, complementàries i unes no donin més avantatges que les altres.

És necessari, per tant, que tecnologia i mecàniques vagin de la mà per la creació i integració de tots els elements fonamentals perquè el videojoc compleixi els objectius.

L'apartat estètic, tot i no ser el més important, té certa importància. En aquest primer desenvolupament sobretot s'ha tingut en compte alhora de donar feedback tant positiu com negatiu amb certs efectes sonors o visuals. Però en una versió final del joc es faria força més èmfasis ja que les capacitats en aquest entorn del creador difereixen força de la resta.

La narrativa no s'ha tingut en compte en aquesta versió del joc ja que el que busquem no és un avanç lineal i que pugui anar de la mà de una història o elements narratius, sinó que busquem sobretot la capacitat del jugador per fer les coses el millor possible en l'àmbit de superació dels nivells.

2. Estudi de viabilitat

Abans de començar a dissenyar i implementar un videojoc, cal valorar si el projecte és viable. En aquest apartat es calcularan els recursos necessaris tant humans com tècnics, es farà un estudi de mercat tenint en compte jocs similars i també sobre la tipologia dels jugadors.

2.1 Recursos necessaris i viabilitat

En aquest apartat es comentaran els recursos i eines utilitzats per a la realització del projecte. A més, es calcularà una estimació hipotètica sobre els costos totals del mateix, tenint en compte pressupostos ficticis dels desenvolupadors. Això ens donarà una idea sobre la viabilitat del projecte.

2.1.1 Recursos tècnics

Pel desenvolupament del projecte no és requereix una gran quantitat de recursos tècnics. Amb un ordinador mitjanament decent i un conjunt de software gratuït en fem prou.

Per desenvolupar-lo s'ha fet servir un ordinador portàtil amb les següents característiques:

- Processador Intel i7-8550U
- 8GB de memòria RAM
- Targeta gràfica NVIDIA GTX 1050.

Mentre que de software s'ha fet servir el següent:

- Unity Versió 2020.3.11 : Motor de joc utilitzat pel desenvolupament del videojoc
- Visual Studio 2019: Editor de codi utilitzat.

2.1.2 Recursos humans.

Pel desenvolupament d'un videojoc cal comptar amb un equip que compleixi els següents rols o perfils professionals:

- **Dissenyador de joc**: Encarregat de definir el joc, objectius, narrativa i mecàniques. És un paper imprescindible per saber què s'ha de fer. És el gran responsable del desenvolupament i que s'encarrega de repartir les tasques del mateix.
- **Programador**: Persona encarregada d'implementar els algorismes del joc. S'encarrega dels aspectes més tècnics.

- **Artista**: Encarregat de crear i dissenyar tot l'apartat artístic del videojoc.

En aquest cas el desenvolupament ha estat a càrrec d'una sola persona i durant un temps limitat, motiu pel qual la feina ha estat molt més enfocada en els dos primers rols, deixant més de costat el tercer.

2.1.3 Viabilitat econòmica

En aquest apartat es mostren els costos totals en cas d'haver de realitzar el pagament de tots els elements comentats exclusivament pel projecte del desenvolupament del videojoc. S'inclourà una aproximació dels pressupostos dels professionals per hores treballades.

RECURS	COST
Maquinària	800 €
Software	0 €
Sou Dissenyador del joc	14 € / hora
Sou Programador	13 € / hora
Sou Artista	13 € / hora

Una vegada s'hagi realitzat la planificació es podrà calcular el cost total del projecte, tenint en compte les hores de desenvolupament que necessitaria cada professional. Aquest seria un cas hipotètic ja que el projecte ha estat realitzat per un estudiant de manera gratuïta i per un cost de 0 euros.

2.2 Estudi de mercat

Quan es comença un projecte que suposa la creació d'un videojoc és molt important fer un bon estudi de mercat per saber com funcionen jocs amb idees semblants a la que tenim. Un cop fet aquest estudi de mercat és important també saber a quin públic potencial es dirigirà el joc.

2.2.1 Cerques realitzades

Primer de tot, caldrà realitzar un criteri per definir quins jocs son semblants al que es proposa. En aquest cas els criteris seran:

- Videojocs de Plataformes 2D
- Videojocs amb diferents possibilitats i/o habilitats
- Videojocs de superació de nivells

Seguint aquests criteris s'ha determinat que alguns dels jocs a tenir en compte per a comparar amb el nostre són els següents:

Trine 4: The Nightmare Prince

Aquest videojoc de plataformes 2D per PC i consoles és segurament l'exemple més proper a l'idea del projecte que tenim pensat. En aquest el jugador controla una sèrie de personatges entre els quals va alternant. Cadascun d'aquests personatges té habilitats diferents i les ha de utilitzar per poder superar escenaris i puzzles diferents. Podem veure un exemple d'escenari amb la cooperació dels personatges a la Figura 2.1.

Figura 2.1: Mostra Trine 4: The Nightmare Prince

Sonic Mania Plus

Un altre videojoc de Plataformes 2D important a tenir en compte és el Sonic Mania Plus, especialment pel sistema de rankings referent a la superació de nivells. En aquest joc, el jugador obté una puntuació al final del nivell en funció de factors com el temps que ha trigat, els anells recollits, etc. i classifica aquesta puntuació dins d'un ranking. Es pot veure una mostra a la Figura 2.2.

Figura 2.2: Mostra Sonic Mania Plus

Teslagrad

Aquest és un videojoc indie, també de plataformes 2D, que ha tingut molt d'èxit en els últims temps, disponible per a PC. També consisteix en superar una sèrie de puzzles on la clau són els poders electromagnètics del protagonista, els quals es manifesten de diferents formes. Es pot apreciar a la Figura 2.3.

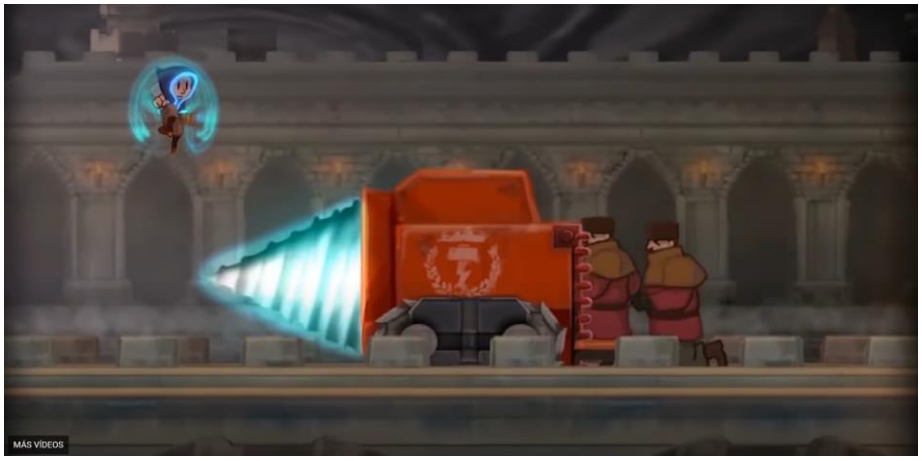


Figura 2.3: Mostra Teslagrad

2.2.2 Matriu de competivitat

Tenint en compte els resultats obtinguts de l'estudi de mercat, s'han comparat els esmentats videojocs amb el nostre tenint en compte els criteris diferencials que té el nostre videojoc. Els apartats més important són l'ús de varietat de mecàniques, sistema de puntuació dels nivells en funció de com s'han resolt i la dificultat.

En conseqüència dels criteris comentats i, tenint en compte els preus d'aquests en el mercat, determinarem el preu del joc. Es pot observar a la Taula 1.

	Varietat de mecàniques	Sistema de puntuació	Dificultat	Preu
Trine 4	Alta	Irrellevant	Normal	7,49 €
Sonic Mania Plus	Normal	Important	Normal-Alta	15,85 €
Teslagrad	Alta	Irrellevant	Normal-Alta	6,99 €
Projecte	Molt alta	Important	Alta	3 €

Taula 1: Matriu de competivitat

2.3 Públic objectiu i tipologia del jugador

A l'hora de desenvolupar un videojoc, és molt important tenir en compte cap a quin tipus de públic anirà enfocat i tenir un coneixement detallat sobre aquest perfil. A continuació es definiran els diferents tipus de jugador segons els objectius que intenten assolir quan juguen a un videojoc i posteriorment analitzarem de quin perfil serien als que es va dirigits aquest projecte:

- **Achiever**: Tenen com a objectiu resoldre la major quantitat de reptes que proposa el joc. Intenten aconseguir la màxima puntuació, recompenses i assoliments que aquest dona a l'abast.
- **Explorer**: Gaudeixen amb l'exploració i el descobriment de coses desconegudes dins del videojoc. Els interessa l'interacció amb el món.
- **Socializer**: Són els jugadors que senten atracció pels aspectes socials que per les possibles recompenses que pugui oferir el joc.
- **Killer**: Són els jugadors que s'enfoquen més en la competivitat i en guanyar els adversaris.

Per tant, tenint en compte aquests possibles públics objectius podem determinar que el que encaixa millor en el nostre joc és el **Achiever**, ja que el que volem aconseguir sobretot és la rejugabilitat dels nivells, invitant a que el jugador sigui creatiu, exprimint totes les possibles habilitats i aconseguint les recompenses tenint en compte aquest factor de ús de tot el que el joc ofereix. També podem tenir en consideració els jugadors **Killers**, ja que pel sistema de puntuació dels nivells es podria realitzar un sistema de rankings competitius per aconseguir la millor puntuació. A més, aconseguir-ho també afavoreix el progrés en el joc.

3. Planificació

3.1 Definició de les tasques a realitzar

En aquest apartat es comentarà la planificació realitzada del treball des de l'inici (meitat de Febrer de 2022) i final del projecte (Maig de 2022). Aquesta planificació es va realitzar juntament amb el tutor del Treball de Final de Grau des del moment en que es va comentar la idea del treball i es van començar a aportar idees.

1. **Pluja d'idees sobre les mecàniques a implementar i primer entorn**. El primer pas és, sobre la idea que es té, començar a pensar quines habilitats podem implementar com a primeres per a primers nivells senzills. S'ha de imaginar possibles primers entorns i idees de habilitats a implementar per superar-los. Així mateix es pot anar preparant un primer entorn de proves bàsic on hi hagi el personatge amb els moviments bàsics (córrer i saltar). Aquesta fase ha de durar unes dues setmanes.
2. **Implementar les habilitats.** Una vegada s'hagi fet una selecció entre totes les idees que s'hagin pensat a la fase anterior, es comencen a implementar les imprescindibles. S'ha de tenir en compte que aquest procés estarà subjecte a canvis durant tot el desenvolupament, ja que pot ser que quan dissenyem els nivells sigui necessari crear alguna nova o que calgui modificar les que tenim per algun bug que pugui sorgir.
3. **Crear un nivell de tutorial.** A continuació s'ha de crear un primer nivell que serveixi com a tutorial. És important tenir en compte que en aquest nivell s'haurà de deixar clara la idea del videojoc, fent que entengui la idea de superació de nivells amb combinacions d'habilitats. Això s'hauria de fer en unes dues setmanes
4. **Preparar menús i tria prèvia d'habilitats:** Una vegada tenim un primer escenari i més o menys les idees d'habilitats realitzades, s'ha de pensar com es farà la implementació per tal que el jugador accedeixi als nivells i faci la tria corresponent d'habilitats, posant unes com a disponibles i altres inhabilitades en cada moment. Això ha de trigar unes dues setmanes.
5. **Disseny de nivells:** Aquesta és, sens dubte, la part que ocuparà més temps en el desenvolupament. Implementar els nivells comporta realitzar l'escenari tenint en compte les habilitats que tindrà disponibles el jugador, el sistema de puntuació i les interfícies que hi haurà mentre es juguin. Això, per suposat, inclou una gran fase de proves i canvis en funció dels resultats en cada moment.
6. **Solució de bugs i afegir sons:** Reservem les darreres dues setmanes per solucionar problemes d'última hora i afegir sons a diverses parts del joc on poden encaixar bé.

S'ha de recordar en tot moment que aquesta planificació està subjecte a canvis, ja que és la primera vegada que s'implementa un joc en la seva totalitat i està subjecte a imprevistos que poden esdevenir.

3.2 Diagrama de Gantt

A continuació, a la taula 3.1, es mostra el Diagrama de Gantt corresponent a les tasques definides a l'apartat anterior.

	02/22 (2/2)	03/22 (1/2)	03/22 (2/2)	04/22 (1/2)	04/22 (2/2)	05/22 (1/2)	05/22 (2/2)
Pluja d'idees i preparar un primer entorn de proves							
Implementar mecàniques							
Preparar nivell tutorial							
Disseny de menús i tria prèvia d'habilitats							
Disseny de nivells							
Solucionar bugs i afegir sons							

Taula 3.1: Diagrama de Gantt

4. Marc de treball i conceptes previs.

4.1 Marc de treball

Fa referència al gènere del projecte, que ja s'ha tractat a l'Apartat 2, en el qual es comenta que es poden trobar altres videojocs d'un estil semblant que ja es troben al mercat i que es podrien comparar amb el realitzat en aquest projecte.

4.2 Conceptes previs

Abans de començar hem de tenir clars una sèrie de conceptes previs, ja que, tot i la experiència que determinada gent pugui tenir en l'àmbit dels videojocs, s'ha de tenir en compte que quan parlem de certs conceptes no són tan obvis per a tothom.

El concepte de Plataformes 2D fa referència a una modalitat de videojocs que es caracteritzen per poder caminar i saltar per damunt de plataformes. Que siguin en 2D significa que es veuen en dues dimensions, la de x i la de y, donant per fet que no hi ha profunditat en els escenaris sino que són plans i el moviment és freqüentment d'esquerra a dreta en horitzontal. El videojoc Super Mario Bros és un dels més mítics i referents en aquest sector. El podem veure a la Figura 4.1.

Figura 4.1: Exemple plataformes 2D Super Mario Bros

El concepte de nivells són els possibles escenaris que ens podem trobar dins del videojoc, els quals han de suposar un repte pel jugador. Aquest repte pot ser en forma de solucionar un puzzle plasmat en l'escenari, superar-lo a contratemps o realitzar una sèrie d'accions concretes. Un exemple clar d'aquest concepte poden ser els cadascun dels nivells del Candy Crush (Veure a la Figura 4.2).

Figura 4.2: Exemple nivells Candy Crush

Com ja hem comentat, als jocs de plataformes, les mecàniques comuns són córrer i saltar a través d'aquestes. Aquí entra el concepte d'habilitats. La peculiaritat d'aquest joc residirà en les més possibles mecàniques a part d'aquestes dos que podrà realitzar el jugador, que donaran noves possibilitats i capacitats per superar els nivells.

5. Disseny del videojoc

5.1 Mecàniques bàsiques

Les mecàniques bàsiques que ha de tenir el videojoc, al ser un plataformes 2D, són les fonamentals d'aquest gènere de jocs. Aquestes són, dins l'espai, poder moure'ns en les dues direccions segons l'eix x amb el nostre personatge i saltar sobre diferents plataformes. Veure Figura 5.1.

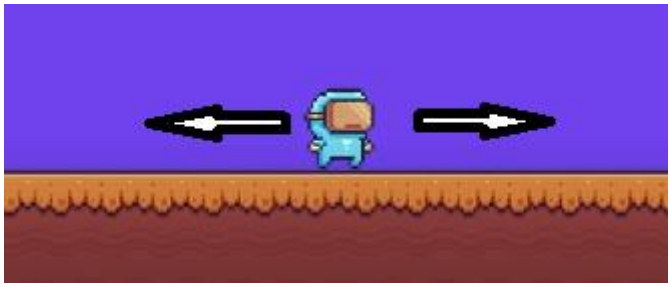


Figura 5.1: Pla horitzontal del joc

Això ho implementarem amb les tecles de les fletxes del teclat, ja que, al després cobrar més importància les diverses habilitats que podrem realitzar, millor deixarem la resta del teclat a fer servir. El salt estàndard que podrà realitzar el personatge el farem amb la tecla espai, ja que és més intuïtiu que amb la fletxa de direcció cap a dalt.

Figura 5.2: Fletxes i espai de teclat

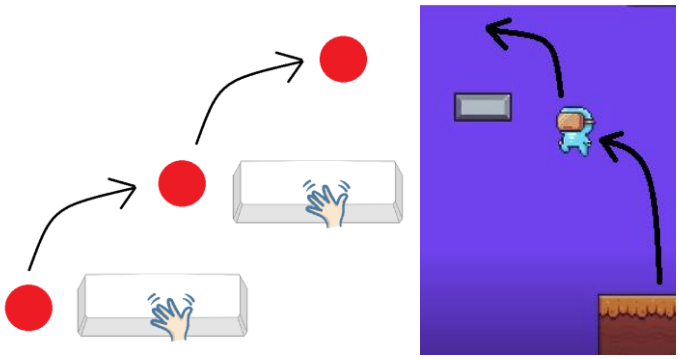
5.2 Habilitats

En aquest apartat es comentaran només les habilitats esmentades fins al moment i no es comentarà sobre idees d'implementació que es tenien i finalment no es van realitzar. Això quedarà a l'apartat ___?

Es farà menció de la tecla que tenen assignada per defecte cada habilitat, tot i que després, una vegada dins del joc, això és personalitzable pel jugador per tal que les ajusti al seu gust.

- **Doble salt:**

Permet realitzar un segon salt estant al aire després de haver saltat ja una vegada. No és repetible després del mateix salt. Es realitza al prémer de nou la tecla de espai una vegada el jugador ja és a l'aire. Permet arribar a llocs més lluny que amb el salt normal, tant en longituds horitzontals com verticals. Veure Figura 5.3



Figures 5.3: Controls i moviment doble salt

- **Dash:**

Es tracta d'un moviment d'avanç horitzontal molt ràpid, així com si el personatge s'impulsés de cop cap a davant. Es fa a gran velocitat però no és un teletransport. Està assignat a una tecla (per defecte Z) i es realitza immediatament al prémer-la. Veure Figura 5.4

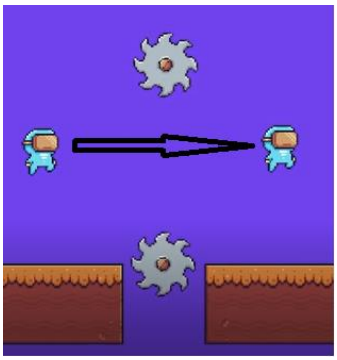


Figura 5.4: Moviment Dash

- **Super velocitat:**

Dona la capacitat al personatge de moure's a una velocitat aproximadament el triple que del normal durant un breu període de temps (1 segon). Aquesta velocitat es manté en els salts però és només horitzontal, pel que no pot fer salts més alts però sí més llargs. Per defecte està assignat a la tecla V i l'habilitat es manté fins que el jugador deixa de prémer la tecla o quan passa 1 segon.

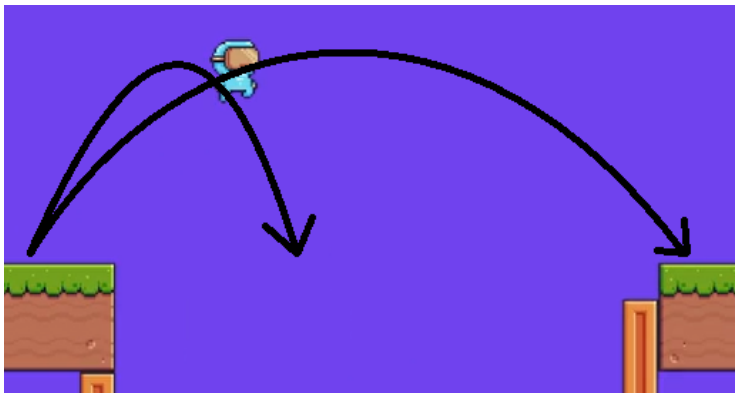


Figura 5.5: Diferència de salt amb super valocitat i sense

- **Planejar**

Obre un paracaigudes durant un breu període de temps. Permet al jugador flotar a l'aire així com si obrís un paracaigudes durant aquest temps, essent la velocitat horitzontal la mateixa que de normal. Això permet arribar a distàncies horitzontals més llunyanes. Està inspirada en la paravel.la del Zelda Breath of the Wild. A l'igual que l'anterior, es manté fins que el jugador deixa anar la tecla o fins que s'acaba el temps propi de l'habilitat. Per defecte està a la tecla X.

Figura 5.6: Habilitat planejar

- **Invisible:**

Permet traspassar alguns tipus de parets dins del joc. També té un temps màxim d'activació d' 1 segon. Al finalitzar l'habilitat, si el jugador està enmig de la paret surt rebutjat cap al costat més proper que té per sortir. A l'igual que les dues anteriors s'activa amb una tecla mantenint-la apretada. En aquest cas, per defecte és la tecla 1.

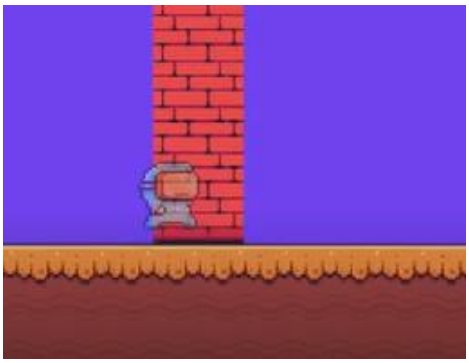


Figura 5.7: Habilitat Invisible

- **Construcció**:

Aquesta habilitat construeix un cub al costat cap al que mira el jugador, sempre que aquest no tingui cap objecte amb el que pugui colisionar en el lloc on suposadament es crearia el cub. Aquest cub no és gaire gran pero ens permet pujar-nos-hi per després saltar a una possible plataforma més alta. Inclús, permet acumular noves construccions unes sobre les altres. Aquesta habilitat ha estat inspirada en Fortnite. Més endavant en el videojoc, amb la creació d'un nou nivell, aquests cubs serveixen per activar polsadors. Es crea el cub al polsar la tecla 2 per defecte.

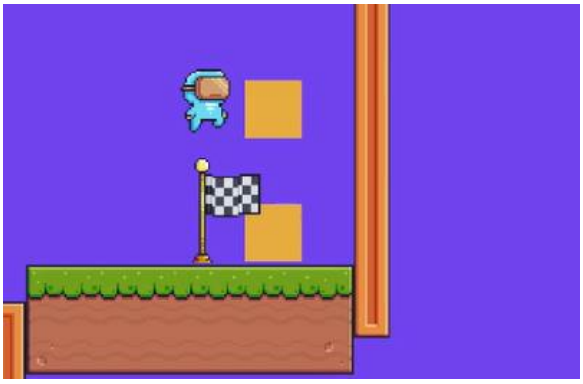


Figura 5.8: Habilitat construcció

- **Imant:**

Aquesta habilitat és un tant peculiar. Ens permet fer servir un imant que té el jugador per moure objectes “metàl·lics”. Realment no és un imant ja que només consistiria en atraure els objectes, però la gràcia és poder moure a distància un objecte que es pot imantar. L’objecte que s’imanta (si hi ha més d’un) és el que es troba més proper al jugador al moment de activar l’habilitat. S’activa l’habilitat amb una tecla i es mou l’objecte amb altres 4 mentre es manté polsada la primera. Per defecte la tecla d’activació és la D i les de moviment TFGH en disposició com si fossin les fletxes del teclat.

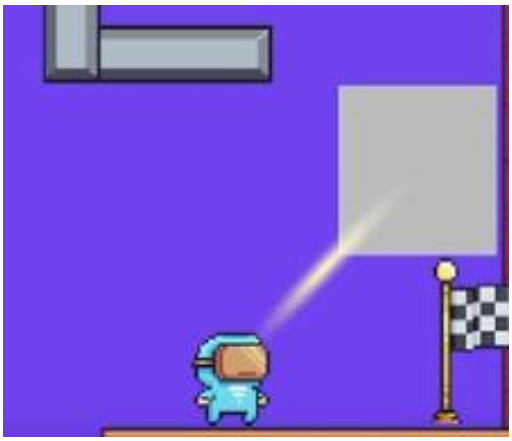


Figura 5.9: Habilitat Imant

- **Escalar**:

Permet al jugador escalar per una paret mantenint polsada una tecla. Té un límit d’ús de 2 segons, per tant no es pot escalar il·limitadament. A més, la paret ha de ser totalment vertical. També es fa servir mantenint la tecla assignada (per defecte A), i es deixa de fer servir una vegada exhaurit el temps o quan el jugador la deixa d’apretar la tecla.

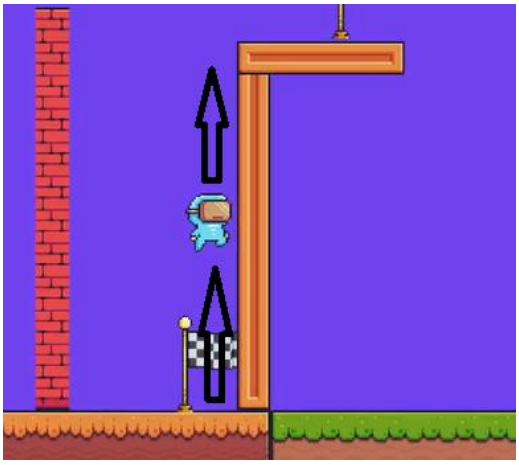


Figura 5.10: Habilitat Escalar

5.3 Espai de joc

L'espai del joc són petits mapes 2D d'on el jugador no pot escapar fent servir només el moviment bàsic. El jugador ha de completar un objectiu dins d'aquests, on la tònica habitual serà anar tocant els checkpoints que s'anirà trobant mentre avança. Això comporta que les mecàniques que pot fer servir estan directa i expressament relacionades amb l'espai i els elements que es troben.

5.4 Nivells

En aquest prototip de joc s'han dissenyat quatre nivells, dos dels quals són d'aprenentatge i/o tutorial i els altres dos ja suposen un repte pel jugador, on ha de aconseguir uns objectius per avançar en el joc. A continuació es presenta la dinàmica de cadascún de aquests nivells:

- **Nivell tutorial:**

Aquest nivell és el tutorial més bàsic. El jugador es troba en un escenari petit i de primeres se li ensenyen els controls bàsics, així com una de les habilitats, el doble salt. El jugador pot completar el nivell passant pels 3 checkpoints amb aquesta habilitat sense problema. Al "finalitzar" el nivell es mostra la puntuació (que només és de 1 punt) i el joc recomana amb una finestra emergent que, per obtenir més puntuació es faci ús de dues habilitats més i es presenten les habilitats dash i planejar. El jugador torna al primer checkpoint i, si torna a superar els dos propers obstacles però amb les noves habilitats apreses, obtindrà 3 punts i finalitzarà el tutorial. La figura 5.11 mostra un diagrama de flux del tutorial i a la figura 5.12 es pot veure el nivell complet:

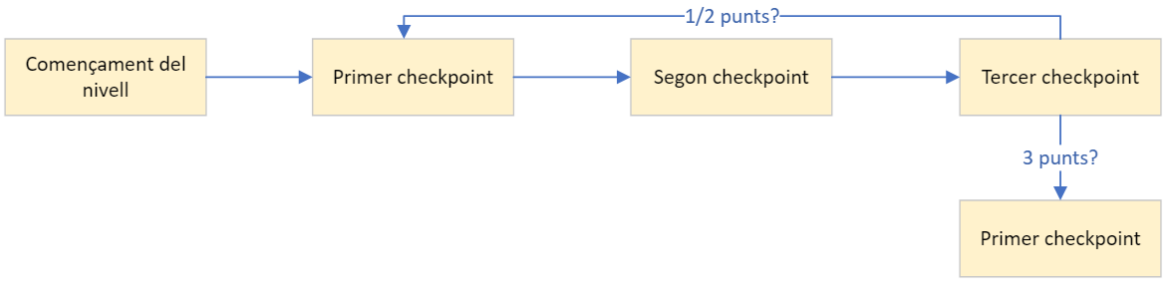


Figura 5.11: Diagrama de flux Nivell tutorial.

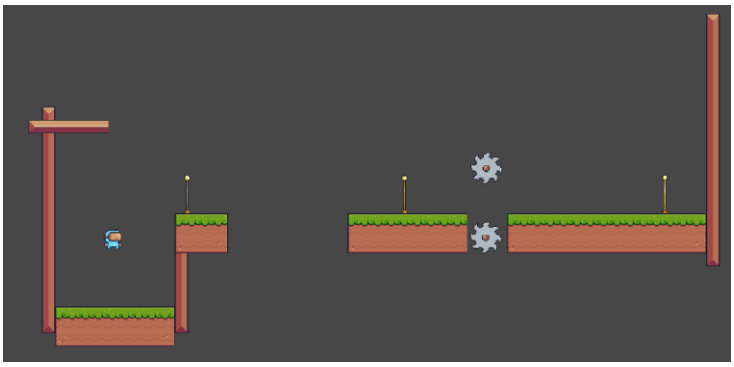


Figura 5.12: Nivell 1

- **Nivell proba d'habilitats:**

Aquest nivell també serveix com a aprenentatge pel jugador. En aquest aprendrà a fer servir, una a una, les habilitats que podrà fer al llarg del joc, per conèixer-les, provar-les i saber com fer-les servir quan les hagi de utilitzar en un dels nivells de repte més endavant. Al començar el nivell el jugador té una finestra emergent que indica l'habilitat que té disponible, amb la seva corresponent tecla. Cada vegada que arriba al següent checkpoint, es desactiva l'habilitat anterior i obté una nova adient per superar el proper obstacle que se li presenta. A la Figura 5.13 es mostra el diagrama de flux, i a la Figura 5.14 es mostra el disseny del nivell complet.

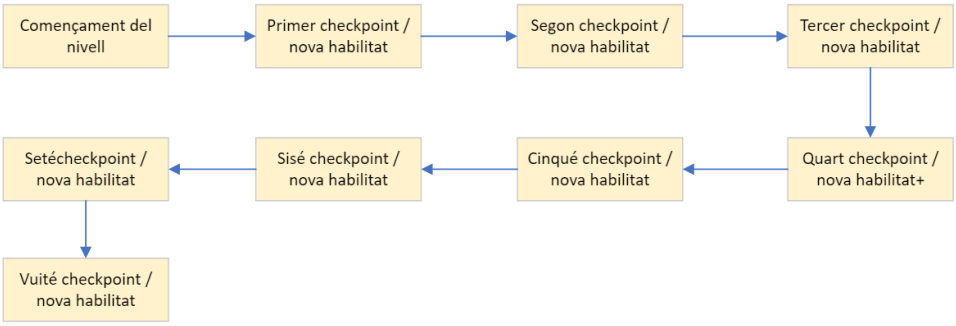


Figura 5.13: Diagrama de flux Nivell proba d'habilitats

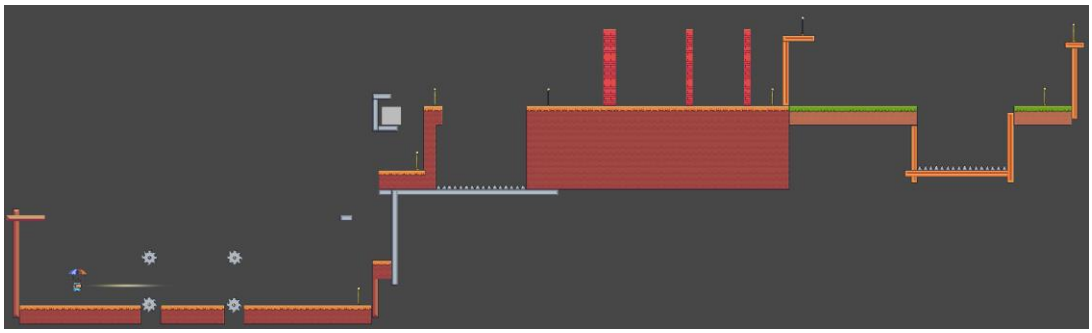


Figura 5.14: Nivell proba d'habilitats

- **Nivell 1:**

Es el primer nivell real puntuable del joc. Abans de començar-lo, al jugador se li deixarà escollir 3 habilitats entre les següents: Dash, doble salt, super velocitat i planejar. L'escenari és especialitzat i adaptat per aquestes quatre habilitats.

El jugador tindrà una interfície on se li mostrarà el número de vegades que ha fet servir cada habilitat així com una puntuació que comença en 2000 punts, la qual anirà baixant amb el temps. Aquestes interfícies es mostraran en a l'apartat 5.1.5, on es comenten més a fons.

L'objectiu del jugador serà passar pels 3 checkpoints del mapa fent un ús equilibrat de les habilitats. Fer-les servir de forma desequilibrada (unes molt i altres molt poc) es veurà reflectit en una penalització de la puntuació final del nivell que serà més gran quan més gran sigui la descompensació.

Cada vegada que el personatge mor en algun dels obstacles, o caient, torna a l'anterior checkpoint. A la puntuació final també es descomptaran 10 punts per cada mort. Veure al diagrama de flux de la Figura 5.15.

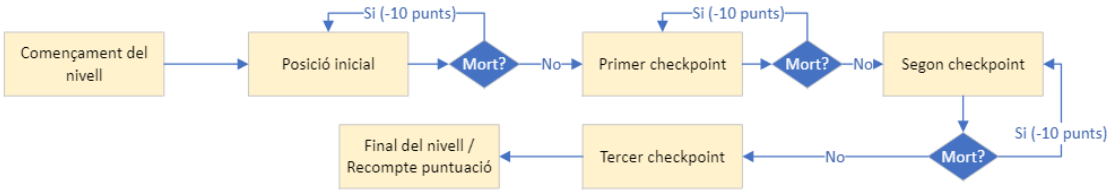


Figura 5.15: Diagrama de flux Nivell 1

També s'ha implementat un sistema anti-tramposos, el qual consisteix en un cooldown que no permet repetir una habilitat en un termini de 2 segons. Si es fa abans, el jugador tindrà una penalització de 5 punts, la qual augmentarà de 5 en 5 cada vegada que es cometi aquesta infracció, que es notificarà al jugador. Veure el nivell complet a la Figura 5.16 i Figura 5.17.

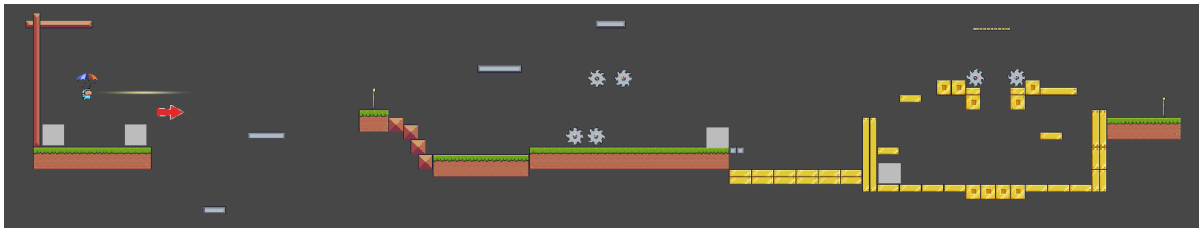


Figura 5.16: Nivell 1, part ½

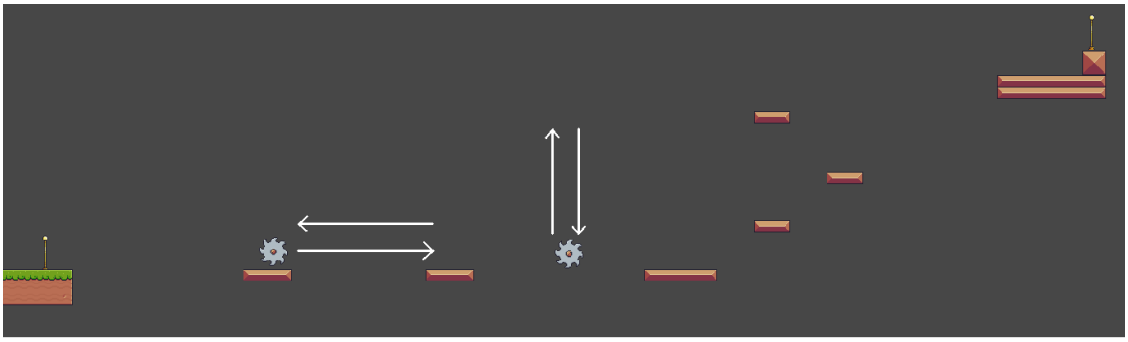


Figura 5.17: Nivell 1, part 2/2

- **Nivell 2:**

Nivell semblant a l'anterior en quan a l'objectiu a realitzar. En aquest cas el nivell està adaptat a les altres 4 habilitats, entre les quals es deixa triar 3 abans de començar: Imant, Construcció, invisible i escalar. En aquest cas, apart del temps, hi ha unes serres mecàniques darrera i la plataforma va avançant, de manera que no tenim més remei que avançar si no es volen acumular morts. El diagrama de flux és pràcticament igual que al nivell 1, es pot observar a la Figura 5.18.

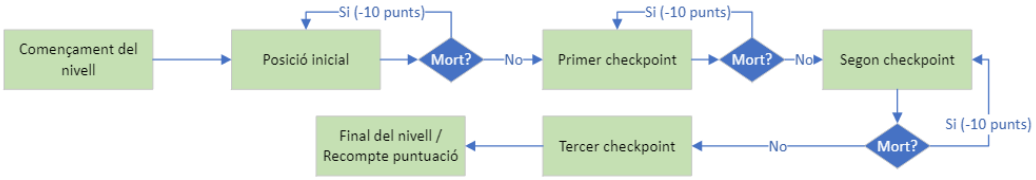


Figura 5.18: Diagrama de flux Nivell 2

En el cas de morir serà com una mort normal i es tornarà a un checkpoint anterior amb la situació anterior no estant la plataforma tant avançada fins aquell punt. Es pot observar el disseny del nivell a la Figura 5.19.

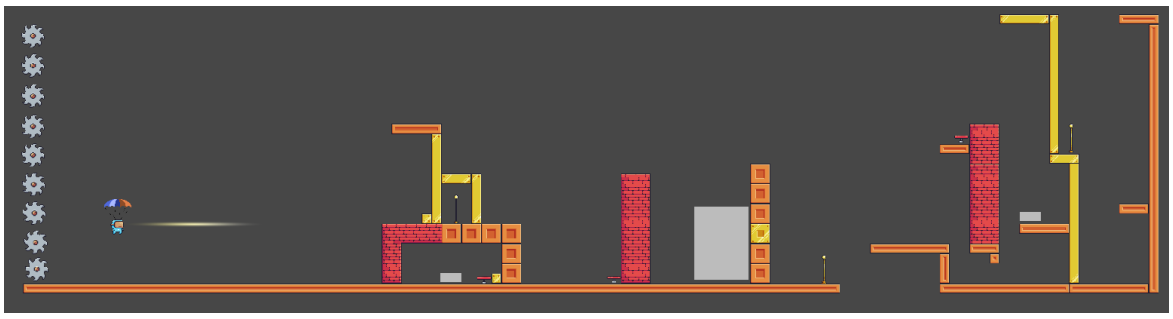


Figura 5.19: Nivell 2

5.5 Interfícies

En quan a interfícies durant el gameplay, com ja hem comentat explicant al nivell 3, tenim a la part superior esquerra la puntuació fins al moment, juntament un comptador de les vegades que hem fet servir cada habilitat fins al moment. A la part superior central tenim la penalització per repetició indicada amb una imatge de tipus filled, és a dir que es recarrega en forma de espiral en 360 graus. Això serveix per indicar el temps fins que es pot tornar a repetir l'habilitat. En el cas que cometem l'error, s'escolta un soroll d'error i s'actualitza aquest marcador amb el número de punts a sancionar.

La resta són les finestres emergents al fer pausa en el joc, o al finalitzar el nivell indicant la puntuació final. A les Figures 5.20, 5.21 i 5.22 es mostren aquestes interfícies de nivell.

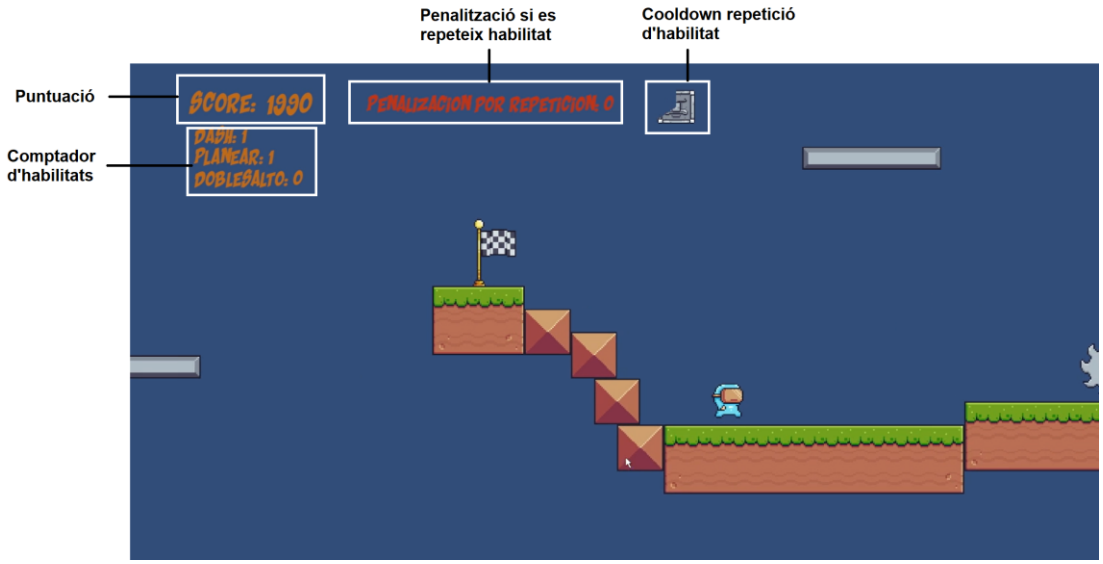


Figura 5.20: Interfícies de nivell

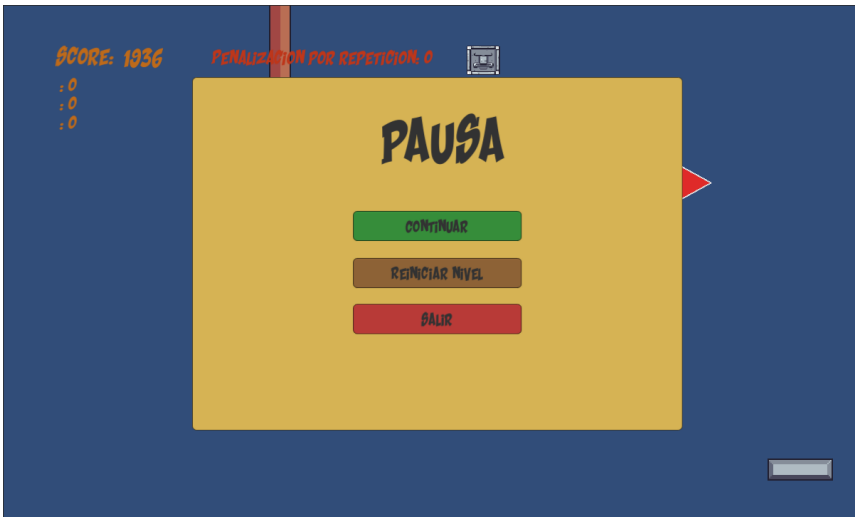


Figura 5.21: Menú de pausa

Figura 5.22: Finestra de puntuació final

5.6 Objectes dels nivells

- **Objectes metàl·lics:**

Són els que es poden moure amb l'habilitat imant. Es poden reconèixer per la seva textura metàl·lica i no es poden moure empenyent-los perquè són molt pesats. Poden ser de diferents mides. Poden accionar botons que obren portes, dels quals parlem més endavant. Veure exemples a la Figura 5.23.

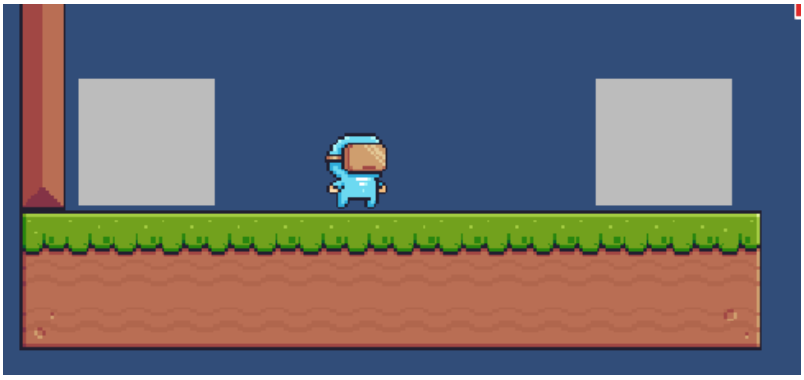


Figura 5.23: Objectes metàl·lics

- **Cubs de construcció:**

Són els que es generen quan el jugador fa servir l'habilitat de construcció. Són tots de la mateixa forma i mida i pesen molt poc, per tant el jugador els pot empènyer un cop creats. Veure exemples a la Figura 5.24.

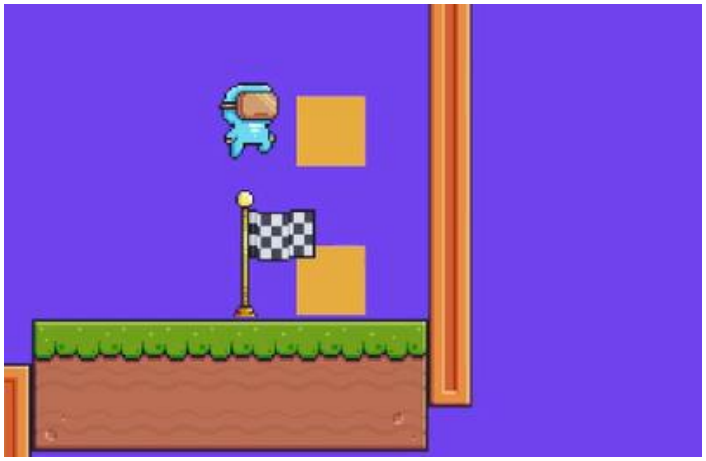


Figura 5.24: Cubs de construcció

- **Botons:**

Serveixen per obrir portes amagades pel mapa. Només poden ser accionats posant sobre algun dels dos objectes esmentats anteriorment. No es poden accionar saltant el jugador a sobre. Veure exemple a la Figura 5.25.

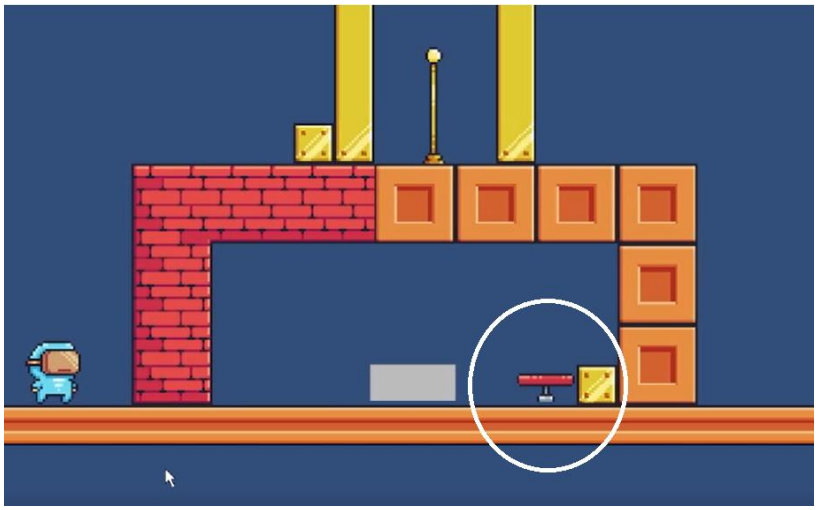


Figura 5.25: Botó interruptor.

- **Trampes o serres metàl·liques:**

Les úniques trampes que s'han creat de moment en aquest prototip del joc. Algunes tenen mobilitat pel mapa. Si el jugador entra en contacte amb una, torna automàticament a l'anterior checkpoint i se li conta com a mort pel global del nivell. Veure exemples a la Figura 5.26.

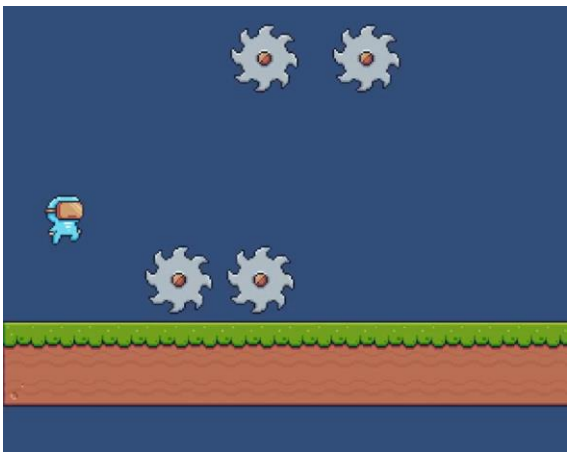


Figura 5.26: Serres metàl·liques

5.7 Menús del joc

- **Menú principal:**

El menú principal és molt bàsic en la versió del joc desenvolupada fins ara. Es disposa de diferents botons per realitzar tres possibles accions: Canviar els controls, reiniciar la partida o seleccionar un nivell a jugar. A l'entrar al joc per primera vegada, només tenim disponible el nivell tutorial i la resta estan bloquejats. Es veu com es mostra a la Figura 5.27:

Figura 5.27: Menú principal en nova partida.

Com es pot observar també a la Figura 5.27, al costat dels dos nivells puntuables del joc ens marca la puntuació màxima que hi ha aconseguit el jugador. Quan s'arriba a la puntuació mínima per desbloquejar el proper nivell, la tipografia d'aquests indicadors es torna de color verd. En aquest cas són 1000 punts. Si es clica el botó de "Reiniciar Partida", es torna a l'estat inicial del joc, bloquejant els nivells excepte el tutorial i tornant les puntuacions a 0. A continuació, a la Figura 5.28, es mostra com es visualitza al haver superat tots els nivells del joc:

Figura 5.28: Menú principal partida completada

- **Menú de controls de joc:**

Si es fa clic al botó de "Controles" del menú principal s'obre una finestra que ens permet triar els controls que volem per cada habilitat. Aquesta finestra té un botó de confirmar, que al clicar-lo confirmem que es jugarà amb els controls tal com estan en aquell moment. Per evitar configuracions dolentes, el botó de confirmar només està habilitat si es compleixen 2 condicions: Que no es repeteixi cap tecla amb cap de les altres, i que el text box on introduïm la tecla no tingui més d'un caràcter. Aquest algorisme per a que el botó estigui habilitat, s'explicarà a l'apartat d'aquest document d'Implementació i proves. El botó de l'habilitat doble salt és l'espai i no es pot canviar. Això és perquè, pel seu funcionament, queda més fluït si el jugador ha de deixar anar la tecla de espai i tornar-la a prémer per realitzar-lo. Si s'assigna a una altre tecla el jugador podria fer el doble salt res més saltar si està pressionant la tecla des d'abans de saltar. Llavors no estaria fent un doble salt i es contaria com a tal. És recomanable haver jugat abans el nivell de proba d'habilitats abans de configurar aquest apartat. Podem visualitzar aquest menú a la Figura 5.29:

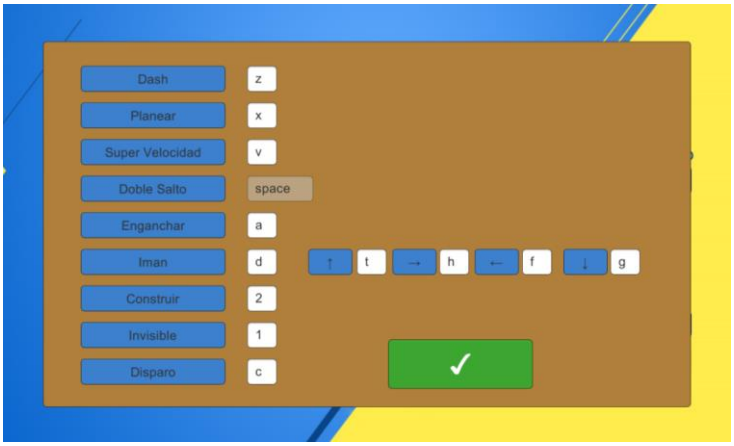


Figura 5.29: Menú de controls de joc

- **Menú previ als nivells:**

Quan seleccionem un dels dos nivells puntuables (el 1 o el 2) ens apareix un menú en forma de finestra previ a jugar el nivell en qüestió. En aquest trobem, a la part de a dalt, una breu descripció del nivell, amb l'objectiu de punts i els aspectes a tenir en compte al llarg de aquest. A la part del mig trobem els botons de selecció de habilitats, els quals, al polsar cadascun, podem visualitzar un vídeo de demostració del funcionament de la corresponent habilitat en el panell que es troba a la dreta. A la part inferior esquerra podem veure les habilitats que tenim seleccionades fins al moment. Amb el botó de confirmar (el qual només es pot fer servir al tenir-ne 3 seleccionades) comencem el nivell i amb el botó vermell esborrem les seleccionades per canviar la combinació. Finalment a dalt a l'esquerra del tot tenim un botó per tornar al menú principal. Tot això ho podem visualitzar a la Figura 5.30:

Figura 5.30: Menú previ a començar un nivell.

5.8 Estètica

Com ja s'ha comentat abans en aquest informe, l'apartat estètic no ha estat pràcticament treballat en aquest projecte. Per tant, s'ha optat per triar algun Asset per a jocs de plataformes 2D dels que podem trobar a la Unity Asset Store. El seleccionat ha estat Pixel Adventure, el qual és força popular entre les possibles opcions gratuïtes que ens ofereix la botiga de Unity. Podem veure exemples d'escenaris creats amb aquest Assets a les Figures 5.31 i 5.32.

Figura 5.31: Exemple 1 escenari Pixel Adventure

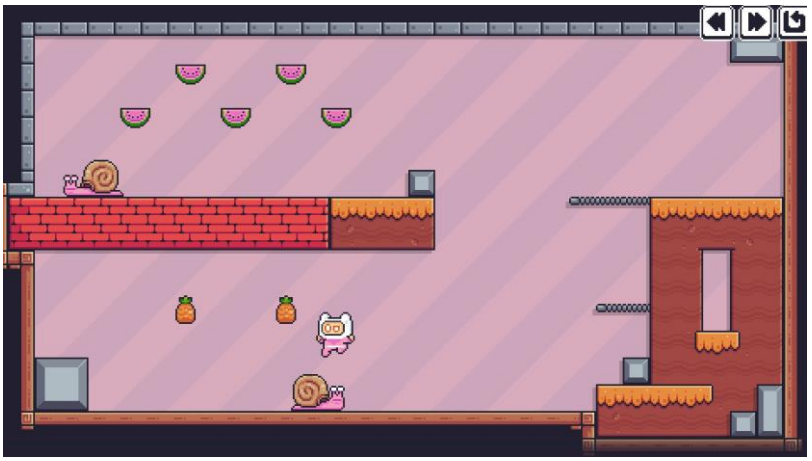


Figura 5.32: Exemple 2 escenari Pixel Adventure

5.9 Pes narratiu.

L'apartat narratiu tampoc té un pes important en el desenvolupament del joc en aquesta versió, ja que l'únic objectiu és la superació de nivells sense explicar cap història referent al personatge.

6. Implementació i proves

En aquest apartat s'explicarà tots els aspectes de programació implementats. Es comentaran tots els scripts que es fan servir en el joc i sobre quins objectes dels comentats a l'Apartat 5 d'aquest document estan implementats.

Però abans d'explicar que es fa i sobre quins objectes estan implementats cadascun, cal comentar que el llenguatge de programació utilitzat ha estat C#.

Finalment es farà referència als problemes apareguts durant la implementació i les solucions que s'han realitzat en conseqüència.

6.1 Implementació

6.1.1 Moviment del personatge

Els scripts implementats pel moviment del personatge han estat dos. Aquests son *playermove.cs* i *playermovetutorial.cs.* Aquests, òbviament, van vinculats al personatge del jugador. El *playermovetutorial.cs* està implementat només pel nivell tutorial, ja que en aquest nivell no es fan servir molts dels elements (variables, consultes, etc) que sí es fan servir en els altres nivells i limita al jugador a únicament les tres habilitats que li calen per superar-lo.

També cal destacar que aquest script fa servir alguna variable estàtica que es crida des d'altres scripts per si els objectes tenen algun tipus de interacció. Quan sigui el cas s'indicarà la referència.

6.1.1.2 Variables

- **Tecles:**

Totes les tecles les guardem al script en strings privats. Aquests els declarem com a strings buits i més endavant, al mètode Start(), el qual s'executa al crear-se el personatge al començar l'escena, es cridarà el respectiu PlayerPrefs de la tecla en qüestió. Cal recordar que els PlayerPrefs ens permeten guardar variables globals (enters, cadena de caràcters, booleans, etc), les quals es poden fer servir entre totes les escenes del joc. Aquest procediment el fem perquè, al consultar la tecla que s'ha apretat d'una habilitat, no s'hagi de cridar cada vegada el PlayerPref. Veiem com s'assignarien en el mètode Start() a la Figura 6.1.

```
void Start()
{
    // TECLAS
    TDash = PlayerPrefs.GetString("TDash", "z");
    TPlanear = PlayerPrefs.GetString("TPlanear", "x");
    TSuperSpeed = PlayerPrefs.GetString("TSuperSpeed", "v");
    TDobleSalto = PlayerPrefs.GetString("TDobleSalto", "space");
    TEscalar = PlayerPrefs.GetString("TEscalar", "a");
    //TSoltar = PlayerPrefs.GetString("TSoltar", "s");
    TIman = PlayerPrefs.GetString("TIman", "d");
    TImanArriba = PlayerPrefs.GetString("TImanArriba", "t");
    TImanDer = PlayerPrefs.GetString("TImanDer", "h");
    TImanIzq = PlayerPrefs.GetString("TImanIzq", "f");
    TImanAbajo = PlayerPrefs.GetString("TImanAbajo", "g");
    TConstruir = PlayerPrefs.GetString("TConstruir", "2");
    TInvisible = PlayerPrefs.GetString("TInvisible", "1");
```

Figura 6.1: Declaració de tecles del playermove.

- **Velocitats i variables generals:**

Necessitem dues variables per guardar velocitats (floats) per a les mecàniques bàsiques. A més, també necessitem guardar un booleà per saber si el jugador mira cap a la dreta o l'esquerra. En aquest cas li hem dit "right" i, si mira cap a l'esquerra, el booleà serà fals. També necessitem el RigidBody (el qual fem que l'objecte s'autoassigni al Start()) i el Renderer per les animacions. Veure a la Figura 6.2.

```
[Header("Velocidades:")]
public float runSpeed = 1.5f;
public float jumpSpeed = 4;
public bool right = true;
Rigidbody2D rb2D;
Renderer m_ObjectRenderer;
```

Figura 6.2: Inicialització variables globals

- **Variables corresponents a les habilitats:**

També necessitem guardar les variables per realitzar cada habilitat. No entrem en profunditat en aquestes de moment, ja que veurem per a què serveixen quan s'expliquin els mètodes corresponents a cada habilitat, realitzats en aquest mateix script.

- **Comptadors d'habilitats:**

Necessitem també guardar uns comptadors per a l'ús de les habilitats. Com ja s'ha comentat prèviament en aquest document, la puntuació final anirà en funció de l'ús equitatiu. Per tant, les guardem com a variables estàtiques per després poder enviar-les, sobretot, al *levelcontroller.cs*, el controlador del nivell i de la puntuació, entre d'altres coses.

```
[Header("Contadores:")]
public static int VecesDobleSalto;
public static int VecesDash;
public static int VecesPlanning;
public static int VecesEscalar;
public static int VecesIman;
public static int VecesConstruct;
public static int VecesInvisible;
public static int VecesSuperSpeed;
```

Figura 6.3: Inicialització comptadors

- **Sons:**

Finalment es necessiten guardar les variables pels sons. Per fer-ho, cal importar la llibreria AudioSource de Unity, la qual ens permet reproduir els sons que assignem des del panell de control de Unity. Per tant, cal que siguin variables públiques, per després poder assignar cada so des del panell de control. Veure a la Figura 6.4.

```
[Header("Sonidos:")]
public AudioSource sonidoSuperSpeed;
public AudioSource sonidoDash;
public AudioSource sonidoIman;
public AudioSource sonidoSaltar;
public AudioSource sonidoGolpe1;
public AudioSource sonidoConstruc;
public AudioSource sonidoParavela;
public AudioSource sonidoDisparo;
```

Figura 6.4: Inicialització variables de so

6.1.1.2 Mètode Start():

És el mètode que s'executa en el primer frame des que existeix l'objecte. Aquí inicialitzem tots els comptadors a 0, assignem als strings de tecles els playerprefs corresponents, assignem els rigidbody2d i renderer del mateix objecte amb els getComponents i inicialitzem cada variable corresponent a les habilitats segons necessitem.

6.1.1.3 Mètode FixedUpdate():

És el mètode que s'executa a cada frame després del start durant l'existència del objecte. Aquí es durà a terme pràcticament tot el que ha de fer el personatge al llarg del nivell: moviments bàsics, habilitats i la resta de coses a tenir en compte amb interaccions de altres objectes. A continuació s'expliquen aquests apartats en detall.

- **Moviment bàsic:**

Els moviments bàsics del personatge, com ja s'ha comentat, són el moviment horitzontal i el salt estàndard. Per fer-ho, es fa una escolta a les tecles corresponents amb l'Input.GetKey(). En el cas del moviment horitzontal, també es té en compte si s'està fent servir l'habilitat Dash, ja que el comportament del moviment del personatge en el moment que s'està realitzant és un moviment horitzontal molt més ràpid. Si no féssim aquest control no es podria realitzar l'habilitat Dash mentre el personatge es mou en horitzontal amb el moviment bàsic. Això passaria perquè el FixedUpdate faria que a l'instant es tornés a realitzar el moviment bàsic i no faria el Dash. En cada cas del moviment horitzontal el que es fa és aplicar la velocitat en l'eix de les x corresponent, canviar la variable "right" per tenir disponible la consulta de cap a quin costat mira el personatge (ja es veurà el per què), invertir l'eix de les x del personatge i aplicar la animació de córrer. En cas que no es faci ús de aquestes tecles, es desactiva la animació de córrer i s'aplica velocitat 0 a l'eix de X. Veure aquest codi la Figura 6.5:

```
if (Input.GetKey("right") && !Dash)
{
    right = true;
    rb2D.velocity = new Vector2(runSpeed, rb2D.velocity.y);
    spriteRenderer.flipX = false;
    animator.SetBool("Run", true);
}
else if (Input.GetKey("left") && !Dash) {
    right = false;
    rb2D.velocity = new Vector2(-runSpeed, rb2D.velocity.y);
    spriteRenderer.flipX = true;
    animator.SetBool("Run", true);
}
else
{
    rb2D.velocity = new Vector2(0, rb2D.velocity.y);
    animator.SetBool("Run", false);
}
```

Figura 6.5: Moviment bàsic horitzontal del jugador

El moviment de salt funciona de forma semblant, ja que el que es fa és mirar si es fa servir el botó de la barra d'espai i si es fa, s'aplica la velocitat sobre l'eix y. També s'ha de tenir en compte si el personatge està en contacte amb el terra. Això s'aconsegueix amb l'objecte CheckGround, que comprova que estigui en contacte amb el terra. Més endavant en aquest apartat es comentarà com funciona a l'Apartat 6.1.2. Veure Figura 6.6:

```
if(Input.GetKey("space"))
{
    if (CheckGround.isGrounded)
    {
        canDoubleJump = true;
        rb2D.velocity = new Vector2(rb2D.velocity.x, jumpSpeed);
    }
```

Figura 6.6: Moviment bàsic de salt

S'ha realitzat també un codi per a un salt millor, molt ben acceptat en els jocs de plataformes 2d, el qual fa un salt més o menys intens en funció del temps que es manté polsada la tecla de salt. Això s'ha fet amb dues variables implementades a la inicialització de variables i tenint en compte la velocitat momentània sobre l'eix y. En funció de això multipliquem el vector de velocitat per un vector normal vertical i fent servir dues variables inicialitzades prèviament per aquest funcionament. El codi és el que podem veure a continuació a la Figura 6.7:

```
if (betterJump)
{
    if (rb2D.velocity.y < 0)
    {
        rb2D.velocity += Vector2.up * Physics.gravity.y * (fallMultiplayer) * Time.deltaTime;
    }
    if (rb2D.velocity.y > 0 && !Input.GetKey("space"))
    {
        rb2D.velocity += Vector2.up * Physics.gravity.y * (lowJumpMultiplayer) * Time.deltaTime;
    }
}
```

Figura 6.7: Moviment de salt millorat

- **Animacions i reset de variables de habilitats:**

En el cas de que simplement toquem el terra, com hem vist abans, fem ús del Checkground.IsGrounded, tenim en compte diversos casos. Si no es compleix, és a dir, que el jugador no toca el terra, s'ha de canviar a l'animació de saltar. Per altra banda, si està sobre el terra, s'ha de canviar a l'animació de córrer i desactivar la de saltar.

Aquí mateix també inicialitzem variables importants de algunes habilitats. Aquestes són: les variables que permeten començar a fer l'habilitat Escalar i les que permeten fer l'habilitat Doble Salt. Això té sentit ja que aquestes habilitats no s'han de poder repetir successivament si el jugador no ha tornat a trepitjar el terra. Veure aquest codi a la Figura 6.8:

```
if (!CheckGround.isGrounded) {
    animator.SetBool("Jump", true);
    animator.SetBool("Run", false);
}
else {
    puedeEscalar = true;
    empiezaEscalar = false;
    JumpT = 0f;
    canDoubleJump = true;
    animator.SetBool("Jump", false);
    if (rb2D.velocity.x != 0)
    {
        animator.SetBool("Run", true);
    }
    else
    {
        animator.SetBool("Run", false);
        //animator.SetBool("Idle", true);
    }
}
```

Figura 6.8: Codi animacions i variables 1/2

També s'inicialitzen les variables corresponents a l'habilitat Escalar ("enganchado" i "puedeEscalar") tenint en compte si el jugador està en contacte amb la paret, i consultem cap a quin costat està orientat el jugador per saber quin check de construcció mantenim actiu. Finalment, al final del fixedUpdate, es crida el mètode que porta el funcionament de totes les habilitats que és el ComprobarHabilidades(). Veure a la Figura 6.9:

```
if (!checkwall.iswalled)
{
    enganchado = false;
    puedeEscalar = true;
}

if (right)
{
    CheckConsR.SetActive(true);
    CheckConsL.SetActive(false);
}
else
{
    CheckConsR.SetActive(false);
    CheckConsL.SetActive(true);
}

ComprobarHabilidades();
```

Figura 6.9: Codi animacions i variables 2/2

6.1.1.4 Mètodes d'entrada i sortida de col·lisió:

Hi ha una habilitat que necessitem tenir en compte amb els mètodes d'entrada i sortida de col·lisió. Aquesta és l'habilitat de l'imant. S'ha implementat que no s'ha de poder realitzar si el jugador està en contacte amb l'objecte que imanta. Això s'ha fet perquè, durant el desenvolupament, un jugador que va provar el joc va descobrir una forma que, amb l'imant, podia passar-se tot el nivell fent ús de aquesta habilitat, posant-se sobre del objecte imantable i saltant alhora que movia l'objecte. Així pràcticament podia superar-lo sobrevolant el mapa. Fent ús d'algunes variables podem controlar que no es faci servir aquesta habilitat a l'entrar en contacte amb el jugador. Veure al codi de la Figura 6.10:

```
private void OnCollisionEnter2D(Collision2D collision)
{
    if (collision.transform.CompareTag("Steel"))
    {
        SteelCollision = true;
        usingMagnet = false;
    }
    else
    {
        SteelCollision = false;
    }
}

private void OnCollisionExit2D(Collision2D collision)
{
    if (collision.transform.CompareTag("Steel"))
    {
        SteelCollision = false;
    }
}
```

Figura 6.10: Controladors de col·lisions del jugador

6.1.1.5 Comprovar habilitats:

El mètode ComprobarHabilidades() s'encarrega de llegir els PlayerPrefs on guardem les habilitats que té en ús el jugador, les quals haurà triat abans de començar el nivell. Es comprova, una a una, si alguna de les habilitats és la corresponent per executar el mètode de l'habilitat i, en cas que sigui així, l'executa. Recordem que aquest mètode s'executa al FixedUpdate, es a dir, que es comprova a cada moment. En la Figura 6.11 veiem només l'exemple amb la comprovació de l'habilitat Dash, ja que és molt llarg.

```
string Hab1()
{
    return PlayerPrefs.GetString("hab1", "");
}

string Hab2()
{
    return PlayerPrefs.GetString("hab2", "");
}

string Hab3()
{
    return PlayerPrefs.GetString("hab3", "");
}

void ComprobarHabilidades()
{
    //Debug.Log(Hab1() + ";" + Hab2() + ";" + Hab3());
    if (Hab1() == "Dash" || Hab2() == "Dash" || Hab3() == "Dash")
    {
        Dash_Skill();
    }
    else
    {
        Dash = false;
        Dash_T = 0;
    }
```

Figura 6.11: Mètode per comprovar l'ús de les habilitats.

6.1.1.6 Mètodes de cada habilitat:

Cada habilitat té implementat un mètode corresponent, el qual es crida o no gràcies al mètode vist anteriorment. A continuació s'explicarà com s'ha implementat cadascun.

- **Codi Habilitat Dash:**

L'habilitat Dash mira, en primer lloc, si s'està clicant la tecla del dash amb un Input.GetKey(TDash). Si s'ha polsat la tecla corresponent, el temps de dash s'incrementa (inicialitzat prèviament a 0), durant 0,05 segons, i si no s'està realitzant ja el dash, per exemple per no poder-los fer seguits, s'incrementa la velocitat 25 vegades cap al costat que estigui mirant el jugador (comprovem amb el booleà right que hem comentat abans). A més, cada vegada que es fa servir, actualitzem el comptador de dash, reproduim el so corresponent i actualitzem el PlayerPrefs de l'última habilitat que hem fet servir com a Dash. Una vegada acaba el temps, tornem a inicialitzar les variables per a un nou ús. Veure el codi a la Figura 6.12.

```
void Dash_Skill()
{
    if (Input.GetKey(TDash))
    {
        Dash_T += 1 * Time.deltaTime;
        if (Dash_T < 0.05f)
        {
            if (!Dash) { VecesDash++; sonidoDash.Play(); PlayerPrefs.SetString("LastHab", "Dash"); }
            Dash = true;
            if (right) { rb2D.velocity = new Vector2(runSpeed * 25, rb2D.velocity.y); }
            else { rb2D.velocity = new Vector2(-runSpeed * 25, rb2D.velocity.y); }
        }
        else
        {
            Dash = false;
        }
    }
    else
    {
        Dash = false;
        Dash_T = 0;
    }
}
```

Figura 6.12: Codi habilitat Dash.

- **Codi Habilitat Planejar:**

El codi implementat per l'habilitat planejar és semblant al del Dash en quant a la implementació i la comprovació alhora de realitzar-la, però els efectes són diferents. Per començar el temps màxim de l'habilitat pot arribar a ser d'un segon. Un altre factor important per realitzar-la és que el jugador no pot estar tocant el terra, ja que no tindria sentit. Per tant, es fa ús novament del CheckGround, en aquest cas per saber si es pot realitzar l'habilitat. A l'igual que a l'anterior, actualitzem comptador quan s'executa, PlayerPrefs d'última habilitat i activem el so corresponent. En aquest cas l'habilitat influeix en la velocitat de l'eix y, a la qual li apliquem la variable inicialitzada prèviament com a lowGravity (0,03f), la qual cosa fa que doni un efecte de planejar a l'aire. També hem d'activar el Sprite de la paravela mentre la fem servir i desactivar el sprite quan ja no es faci ús (Paravela.SetActive(true/false)). Veure el codi a la Figura 6.13:

```
void Planing_skill()
{
    if (Input.GetKey(TPlanear))
    {
        planning_T += 1 * Time.deltaTime;
        if (planning_T < 1.0f && !CheckGround.isGrounded)
        {
            if (!planning) { VecesPlanning++; sonidoParavela.Play(); PlayerPrefs.SetString("LastHab", "Planear"); }
            planning = true;
            Paravela.SetActive(true);
            if (CheckGround.isGrounded == false)
            {
                rb2D.velocity = new Vector2(rb2D.velocity.x, lowGravity);
            }
        }
        else
        {
            planning = false;
            Paravela.SetActive(false);
        }
    }
    else
    {
        planning = false;
        Paravela.SetActive(false);
        planning_T = 0;
    }
}
```

Figura 6.14: Codi habilitat Planejar

- **Codi Habilitat Doble Salt:**

El codi realitzat per l'habilitat del Doble Salt no s'ha fet en un mètode a part a l'igual que les altres. Això és per què l'habilitat Doble Salt ha de realitzar-se si polsem la tecla espai després d'haver saltat. És a dir, el salt normal s'ha de realitzar si es fa servir la tecla espai i el personatge està a terra (Checkground) i el Doble Salt ha de realitzar-se al fer clic a la tecla espai una vegada el personatge ja és a l'aire. Si això es compleix i el personatge té l'habilitat assignada, és quan es fa el Doble Salt. També, igual que a les anteriors habilitats, activem el so corresponent, actualitzem comptador i assignem a la darrera habilitat feta el Doble Salt. Veure codi a la Figura 6.15.

```
if(Input.GetKey("space"))
{
    if (CheckGround.isGrounded)
    {
        canDoubleJump = true;
        rb2D.velocity = new Vector2(rb2D.velocity.x, jumpSpeed);
    }
    else
    {
        if (Input.GetKeyDown("space") )
        {
            if (canDoubleJump && (Hab1() == "DobleSalto" || Hab2() == "DobleSalto" || Hab3() == "DobleSalto"))
            {
                sonidoSaltar.Play();
                rb2D.velocity = new Vector2(rb2D.velocity.x, jumpSpeed);
                VecesDobleSalto++;
                PlayerPrefs.SetString("LastHab", "DobleSalto");
                canDoubleJump = false;
            }
        }
    }
}
```

Figura 6.15: Codi Habilitat Doble Salt.

- **Codi Habilitat Super-Velocitat:**

L'habilitat de Super-Velocitat s'ha implementat de una forma molt semblant al Dash, ja que realment és el mateix concepte amb la diferència que la Super-Velocitat arriba a durar 1 segon i augmenta la velocitat de moviment bàsic horitzontal del personatge de 1,5 a 4 durant aquest breu període de temps. La Super-Velocitat, a més, deixa de tenir ús en el moment que deixem anar la tecla d'activació o, per contra, quan passa un segon des que s'ha començat a utilitzar. Per tant, la diferència fonamental amb l'habilitat Dash, és que, enlloc de aplicar la velocitat sobre el rigidbody2d directament, incrementem el float runSpeed durant aquest breu període de temps, que és el que s'utilitza amb el moviment bàsic. Veure el codi a la Figura 6.16.

```
void SuperSpeed_Skill()
{
    if (Input.GetKey(TSuperSpeed))
    {
        Superspeed_T += 1 * Time.deltaTime;
        if (Superspeed_T < 1f)
        {
            if (!isSuperSpeed) {
                isSuperSpeed = true;
                sonidoSuperSpeed.Play();
                VecesSuperSpeed++;
                PlayerPrefs.SetString("LastHab", "SuperSpeed");
            }
            runSpeed = 4f;
        }
        else
        {
            isSuperSpeed = false;
            runSpeed = 1.5f;
        }

    }
    else
    {
        isSuperSpeed = false;
        runSpeed = 1.5f;
        Superspeed_T = 0;
    }
}
```

Figura 6.16: Codi Habilitat Super-Velocitat

- **Codi Habilitat Invisible:**

L'habilitat invisible també té un temps d'ús màxim d'un segon mentre mantenim polsada la tecla corresponent. La idea d'aquesta habilitat és que el personatge pugui travessar parets mentre la fa servir. Per fer-ho, a l'apartat de inicialització de variables haurem declarat un TilemapCollider2D com a públic, de manera que se li haurà assignat el tilemap de parets. Aquest collider el desactivarem mentre s'estigui fent servir l'habilitat, de manera que, si no hi ha el collider, romandrà el sprite però el personatge la podrà travessar. A l'hora de la implementació, cal que les parets que volem que siguin travessables al fer ús de l'habilitat, les posem en un tilemap diferent al que serien la resta de parts del mapa. Això cal fer-ho de aquesta manera, ja que, si desactivéssim tot el Tilemap Collider del mapa, donaria lloc a interaccions que no ens interessen amb la idea d'aquesta habilitat. La més clara i evident és que es travessaria el terra i es cauria a sota del mapa. També es podria sortir d'aquest pels laterals. Una altre implementació que s'ha realitzat, per tal de que l'habilitat quedi visualment molt més creïble, és modificar la transparència del sprite del jugador mentre se n'està fent ús. Això ho podem fer amb el renderer del personatge. Veure codi a la Figura 6.16.

```
void Invisible_Skill()
{
    if (Input.GetKey(TInvisible))
    {
        Invisible_T += 1 * Time.deltaTime;
        if (Invisible_T < 1f)
        {
            if (!isInvisible) { isInvisible = true; VecesInvisible++; PlayerPrefs.SetString("LastHab", "Invisible"); }
            m_ObjectRenderer.material.color = new Color(1.0f, 1.0f, 1.0f, 0.5f);
            Walls.enabled = false;
        }
        else
        {
            isInvisible = false;
            m_ObjectRenderer.material.color = new Color(1.0f, 1.0f, 1.0f, 1.0f);
            Walls.enabled = true;
        }

    }
    else
    {
        isInvisible = false;
        m_ObjectRenderer.material.color = new Color(1.0f, 1.0f, 1.0f, 1.0f);
        Invisible_T = 0f;
        Walls.enabled = true;
    }
}
```

Figura 6.17: Codi Habilitat Invisible

- **Codi Habilitat Enganxar.**

L'habilitat d'enganxar ha de tenir en compte dos factors: que el personatge estigui a terra, consulta que podem fer, com prèviament s'ha vist, amb el Checkground, i si el jugador està tocant una paret. Aquesta darrera consulta l'executem amb la variable estàtica checkwall.iswalled, que s'explicarà més endavant a l'Apartat 6.1.3 i que serveix per saber si el personatge està tocant una paret. A partir d'aquestes dues consultes, i si no hem començat ja a escalar, comencem a realitzar l'habilitat, que és quan, com veiem al codi de la Figura 6.18, actualitzem el comptador. L'habilitat s'executa fins a un màxim de 2 segons o fins que el jugador deixa de polsar la tecla corresponent a l'habilitat. Si el jugador està escalant apliquem una velocitat vertical cap a dalt. També apliquem una petita velocitat horitzontal en sentit contrari de la paret que toca. Això es fa per donar un efecte visual millor, semblant un zig-zag al escalar per la paret. Veure el codi a la Figura 6.18.

```
void Engancha()
{
    if (Input.GetKey(TEscalar))
    {
        if (checkwall.iswalled && puedeEscalar)
        {
            if (!empiezaEscalar && !CheckGround.isGrounded)
            {
                VecesEscalar++;
                empiezaEscalar = true;
            }
            if (JumpT <= 2f)
            {
                JumpT += 1 * Time.deltaTime;
                if (right)
                {
                    rb2D.velocity = new Vector2(-0.3f, 1f);
                }
                else
                {
                    rb2D.velocity = new Vector2(0.3f, 1f);
                }
            }
            else
            {
                puedeEscalar = false;
            }
        }
    }
```

Figura 6.18: Codi Habilitat Enganxar

- **Codi Habilitat Construcció.**

En el cas de l'habilitat de construcció, el més important a tenir en compte en el desenvolupament és la posició del cub que es construeix i si realment es pot realitzar l'habilitat. El cub s'ha de construir lleugerament per davant del jugador, és a dir, que aquesta posició respecte del jugador canviarà segons si mira cap a la dreta o cap a l'esquerra. Per portar aquest control es fa servir el booleà right. En funció d'això la posició serà un vector2D que podrà ser (0,3 , 0) o (–0.3 , 0). Per comprovar que el jugador no tingui una paret just davant es fa servir una eina semblant a l'anteriorment comentada, iswalled, però aquesta funciona en certa forma diferent. Es comentarà la diferència en els respectius scripts als apartats 6.3 i 6.4. Una vegada comprovem que es pot construir a la posició on toca, instanciem el GameObject que haurem passat com una variable pública. Com a la resta d'habilitats, al realitzar-la, actualitzem comptador, cridem el corresponent so i actualitzem el PlayerPrefs de la darrera habilitat. Veure codi a la Figura 6.19.

```
void Construction_Skill()
{
    Debug.Log(Construction.iswalled);
    if (Input.GetKey(TConstruir))
    {
        //Debug.Log(Construction.iswalled);
        if (right)
        {
            bulletPost = transform.position;
            if (!Construction.iswalled && !Constructed)
            {
                //Debug.Log("Construct Right");
                bulletPost += new Vector2(0.3f, 0f);
                Instantiate(Construct, bulletPost, Quaternion.identity);
                sonidoConstruc.Play();
                VecesConstruct++;
                PlayerPrefs.SetString("LastHab", "Construct");
            }
        }
        else
        {
            bulletPost = transform.position;
            if (!Construction.iswalled && !Constructed)
            {
                //Debug.Log("Construct Left");
                bulletPost += new Vector2(-0.3f, 0f);
                Instantiate(Construct, bulletPost, Quaternion.identity);
                sonidoConstruc.Play();
                VecesConstruct++;
                PlayerPrefs.SetString("LastHab", "Construct");
            }
        }
        Constructed = true;
    }
    else
    {
        Constructed = false;
    }
}
```

Figura 6.19: Codi Habilitat Construcció

- **Codi Habilitat Imant.**

L'habilitat Imant és la més complexa i el funcionament es realitza des d'un altre objecte de joc que no és el del personatge del jugador. Per realitzar-la, a l'script del jugador el que fem és modificar un booleà estàtic a true (usingMagnet) quan polsem la tecla corresponent i es manté mentre la mantenim polsada. En aquesta part actualitzem el comptador, activem el so i actualitzem la darrera habilitat. També activa el gameobject que és semblant a un raig, que dona un efecte millor.

El booleà estàtic esmentat el rep un GameObject (SteelFinder), fill del GameObject del personatge, que realitza tot el procediment de buscar l'objecte sobre el qual es fa l'habilitat (recordem que ha de ser el més proper al personatge) i moure l'objecte en el moment que el booleà passa a ser true i mentre es manté així. El funcionament i la implementació d'aquest s'expliquen a l'Apartat 6.5. Veure el codi de l'habilitat realitzat al *playermove.cs* a la Figura 6.20

```
void Magnet()
{
    if (Input.GetKey(TIman) && !SteelCollision && CheckGround.isGrounded)
    {
        if (usingMagnet == false)
        {
            sonidoIman.Play();
            VecesIman++;
            PlayerPrefs.SetString("LastHab", "Iman");

        }
        RayoIman.SetActive(true);
        usingMagnet = true;
    }
    else
    {
        usingMagnet = false;
        RayoIman.SetActive(false);
        sonidoIman.Stop();
    }
    if (!usingMagnet)
    {
        RayoIman.SetActive(false);
        sonidoIman.Stop();
    }
}
```

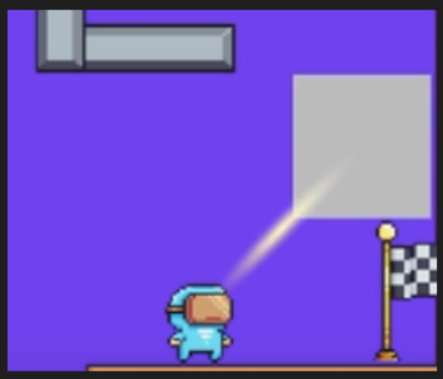
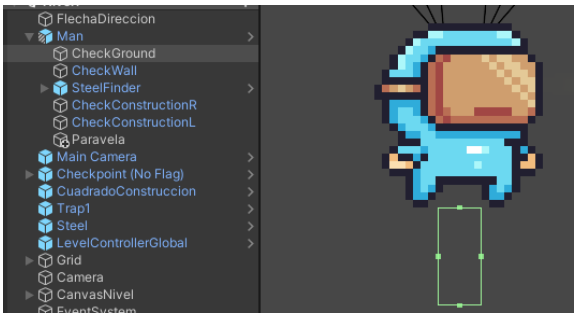


Figura 6.20: Codi Habilitat Imant

6.1.2 CheckGround

L'script *CheckGround.cs* és el que va vinculat a l'objecte CheckGround del personatge del jugador. Aquest objecte només consta d'un BoxCollider2D de tipus Trigger (que es pot traspassar) que es troba una mica més avall dels peus del personatge. Per tant, és un script de suport per altres, com ja hem vist al *Playermove.cs*, i que consta d'una variable estàtica booleana (isGrounded) que es pot fer servir en altres scripts per verificar si el personatge està tocant el terra. Consta dels mètodes OnTriggerEnter2D, OnTriggerStay2D i OnTriggerExit2D, els quals retornen la variable isGrounded com a true si el checkground està en intersecció amb un objecte amb els tags "Ground", "Steel" o "Wall". Es miren aquests tres tags de diferents objectes ja que són objectes els quals, si estem a sobre, es considera que el personatge ha de tenir un comportament com si estigués sobre qualsevol terra. Podem veure l'script a la Figura 6.21.




```
public class CheckGround : MonoBehaviour
{
    // Start is called before the first frame update
    public static bool isGrounded;

    private void OnTriggerEnter2D(Collider2D collision)
    {
        if (collision.transform.CompareTag("Ground") || collision.transform.CompareTag("Steel") || collision.transform.CompareTag("Wall"))
        {
            isGrounded = true;
        }
    }

    private void OnTriggerStay2D(Collider2D collision)
    {
        if (collision.transform.CompareTag("Ground") || collision.transform.CompareTag("Steel") || collision.transform.CompareTag("Wall"))
        {
            isGrounded = true;
        }
    }

    private void OnTriggerExit2D(Collider2D collision)
    {
        if (collision.transform.CompareTag("Ground") || collision.transform.CompareTag("Steel") || collision.transform.CompareTag("Wall"))
        {
            isGrounded = false;
        }
    }
}
```

Figura 6.21: Codi CheckGround.cs

6.1.3 CheckWall

L'script checkwall és el que va vinculat a l'objecte CheckWall, fill del personatge del jugador. El funcionament d'aquest és pràcticament igual al del CheckGround, l'únic que en aquest cas a l'objecte Checkwall hem assignat dos BoxColliders per mirar els dos costats del jugador del personatge. La variable estàtica que modifica és iswalled i en aquest cas no té en compte si l'objecte té algun tag, ja que com només es fa servir per l'habilitat escalar i no hi ha de moment cap objecte del joc que es consideri que no pot ser factible per realitzar-la. Veure el codi a la Figura 6.22.

```
public class checkwall : MonoBehaviour
{
    public static bool iswalled;
    // Start is called before the first frame update
    private void OnTriggerEnter2D(Collider2D collision)
    {
        iswalled = true;
    }

    private void OnTriggerStay2D(Collider2D collision)
    {
        iswalled = true;
    }

    // Update is called once per frame
    private void OnTriggerExit2D(Collider2D collision)
    {
        iswalled = false;
    }
}
```

Figura 6.22: Codi CheckWall.cs

6.1.4 Comprovar si es pot construir

L'script *Construction.cs* és el codi que va vinculat a dos objectes fill, el CheckConstructR i CheckConstructL. Això és per a que, per verificar si podem construir, ha de fer-se en funció del costat cap al que s'estigui mirant. Aquests objectes s'aniran activant i desactivant entre ells en funció de cap a quin costat mira el personatge. Es passen com a objectes públics al *playermove.cs* i es fa el procés al FixedUpdate().

La funció d'aquest script és també retornar un booleà estàtic que indiqui si es pot construir. Per tant, els BoxColliders son de la mateixa dimensió que l'objecte que es construiria i la posició és la mateixa de on es construïria. Per suposat, són Triggers (travessables). Per tant, aquest Script, a diferència del Checkwall, es fa servir per verificar que es pugui construir enlloc d'escalar i els boxcolliders són, en conseqüència, molt diferents. També aquí s'ha tingut en compte que l'objecte de colisions siguin objectes amb els tags "Ground" i "Wall". En aquest cas, sí que ho hem tingut en compte ja que, en futures implementacions, pot haver objectes amb els quals si que ens interessi aquesta col·lisió per determinades interaccions (parar el dispar d'un enemic, per exemple). Veure codi a la figura 6.23.

```
public class Construction : MonoBehaviour
{
    public static bool iswalled = false;

    private void OnTriggerStay2D(Collider2D collision)
    {
        if (collision.transform.CompareTag("Ground") || collision.transform.CompareTag("Wall")) {
            iswalled = true;
        }
    }

    // Update is called once per frame
    private void OnTriggerExit2D(Collider2D collision)
    {
        if (collision.transform.CompareTag("Ground") || collision.transform.CompareTag("Wall"))
        {
            iswalled = false;
        }
    }


}
```

Figura 6.23: Codi Construction.cs

6.1.5 Triar l'objecte a moure amb l'habilitat Imant

Com ja s'ha comentat prèviament, l'objecte que duu a terme l'habilitat Imant no és el propi personatge sinó que es realitza des d'un GameObject fill de aquest i que es troba a la mateixa posició. Aquest té assignat l'script SteelFinder. Aquest objecte s'encarrega de buscar el Objecte amb el tag "Steel" més proper. Això ho fem en un mètode que fa un recorregut per tots els objectes que tenen aquest tag i assigna a un GameObject privat el que es troba a menys distància del propi GameObject. Veure el codi a la Figura 6.24.

```
public Transform getClosestSteelObject()
{
    multipleSteelObjects = GameObject.FindGameObjectsWithTag("Steel");
    float closestDistance = Mathf.Infinity;

    Transform trans = null;

    foreach (GameObject go in multipleSteelObjects)
    {
        float currentDistance;
        currentDistance = Vector3.Distance(transform.position, go.transform.position);
        if (currentDistance < closestDistance)
        {
            closestDistance = currentDistance;
            trans = go.transform;
        }
    }
    return trans;
}
```

Figura 6.24: Mètode per trobar el objecte Steel més proper

A l'Update() d'aquesta classe, el que fem es consultar en tot moment el booleà estàtic que se'ns envia des del *playermove.cs.* Si aquest és false, el que fem és buscar continuament l'objecte Steel més proper i assignar-lo al RigidBody rb2D. Quan el el booleà estàtic usingMagnet passa a ser true, llavors donem la capacitat al jugador de moure amb les tecles assignades el darrer objecte assignat com al més proper. Necessitem també, tant la posició exacta del rigidBody del jugador, com de l'objecte, per dimensionar l'objecte visual que s'activa al fer servir l'habilitat, ja que és un lineRenderer. Veure el codi a la Figura 6.25.

```
void Update()
{

    if (playermove.usingMagnet)
    {
        lineRenderer.SetPosition(1, rb2D.position - rb2DPlayer.position);
        //rb2D.bodyType = RigidbodyType2D.Kinematic;
        if (Input.GetKey(TImanIzq))
        {
            rb2D.velocity = new Vector2(-moveSteel, rb2D.velocity.y);
        }
        if (Input.GetKey(TImanDer))
        {
            rb2D.velocity = new Vector2(moveSteel, rb2D.velocity.y);
        }
        if (Input.GetKey(TImanArriba))
        {
            rb2D.velocity = new Vector2(rb2D.velocity.x, moveSteel);
        }
        if (Input.GetKey(TImanAbajo))
        {
            rb2D.velocity = new Vector2(rb2D.velocity.x, -moveSteel);
        }
        if (!Input.GetKey(TImanAbajo) && !Input.GetKey(TImanArriba) && !Input.GetKey(TImanIzq) && !Input.GetKey(TImanDer))
        {
            rb2D.velocity = new Vector2(0.0f, 0.0f);
        }
    }
    else
    {
        rb2D.bodyType = RigidbodyType2D.Dynamic;
        closestSteelObject = getClosestSteelObject();
        rb2D = closestSteelObject.GetComponent<Rigidbody2D>();
    }
}
```

Figura 6.25: Codi del Update del SteelFinder.cs

6.1.6 Canviar d’escena

La forma de canviar d’escena en el joc és a través de botons en tots els casos. Per implementar-ho s'ha assignat a l’objecte botó que volem que faci el canvi d’escena, a la tasca d’OnClick() l'script ChangeScene.cs. Es selecciona el mètode corresponent i se li passa per escrit el nom de l’escena corresponent. Es pot veure aquesta assignació a la Figura 6.26.

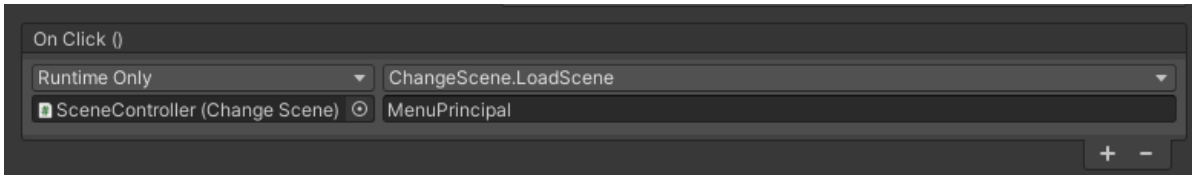


Figura 6.26: Assignació dl'script de canvi d’escena a l’objecte Botó

L’Script ChangeScene.cs s'ha realitzat simplement implementant el mètode LoadScene, el qual li passem el string amb el nom de la escena i fem servir la eina de Unity SceneManager per obrir la escena que ens arriba, que és la que haurem escrit al OnClick del botó. Veure script a la Figura 6.26.

```
using UnityEngine.SceneManagement;

public class ChangeScene : MonoBehaviour
{
    public void LoadScene(string sceneName)
    {
        SceneManager.LoadScene(sceneName);
    }
}
```

Figura 6.27: Codi pel canvi d’escena

6.1.7 Controlador de nivell

En aquest apartat es comenten els controladors dels nivells. Com ja s'ha explicat, hi ha quatre nivells en la versió del joc implementada i els funcionaments respectius són diferents (comprovar l'Apartat 5.4 del disseny dels nivells). Per tant, tenim diferents implementacions d'aquests controladors segons el nivell. En aquest apartat es mostraran pocs talls de codi en relació a la mida dels scripts, ja que les possibilitats varien segons el nivell i el que es fa en molts casos són coses molt específiques segons la situació, com moure el jugador a una posició específica segons on hagi mort, o actualitzar la puntuació segons unes normes concretes.

6.1.7.1 Nivell tutorial

En el control del nivell de tutorial hem de tenir diverses coses en compte. En primer lloc hem de controlar quants checkpoints hem activat i guardar la puntuació en funció d'aquests. També hem de poder controlar des d'aquí el jugador per, en cas que mori, canviar la seva posició en el mapa i tenir també guardats els checkpoints per consultar si s'han activat. A més hi ha 4 checkpoints que es controlen des de l'script, ja que al reiniciar el nivell la segona vegada s'han de restablir els dos últims però no el primer. El control dels checkpoints es fa gràcies al respectiu controlador, que es comenta a l'Apartat 6.15. També hem de controlar quan surt cadascun dels panells d'instruccions al llarg del nivell (canvas) i booleans que guardin si s'han realitzat cadascuna de les habilitats. Per fer-ho, consultarem les variables estàtiques del controlador del jugador i si són més grans de zero, es contarà.

Per tant, aquest script contempla les següents opcions:

- Si el jugador arriba al primer checkpoint guanya un punt.
- Si arriba al segon checkpoint i ha fet servir 2 habilitats obté 2 punts.
- Si arriba al tercer checkpoint i ha fet servir les 3 habilitats té 3 punts i acaba el tutorial. Si no els ha obtingut, surt el la finestra emergent que explica les habilitats Dash i Planejar i el jugador torna al Checkpoint 1, amb 1 punt. S'inhabiliten els checkpoints 2 i 3 i s'activen els altres dos, que es troben a la mateixa posició.
- Si es clica el botó R es reinicia el nivell.
- Si el jugador mor per caiguda i té un punt, torna a la posició del primer checkpoint. Si en té 2 torna al segon. La posició s'assigna de forma manual des de l'script.
- Si mor per un objecte que fa mal (les serres que giren) torna al checkpoint 2.
- Mostrar finestra emergent corresponent segons la part del nivell que toca.

6.1.7.2 Nivell prova d'habilitats

Aquest nivell funciona té un funcionament força diferent als altres, i alhora força més senzill. El jugador és incapaç d'arribar al següent checkpoint si no fa ús de l'habilitat que li toca, així que en la implementació no cal preocupar-se de situacions hipotètiques molt descarades, com per exemple, que pugui arribar al tercer checkpoint sense haver passat pel segon. Només necessitem controlar quants checkpoints ha arribat a tocar en tot moment. Tampoc hem de gestionar puntuació ja que l'únic objectiu d'aquest nivell és que el jugador faci ús de totes les habilitats passant per tots els checkpoints i, a l'arribar al últim, acabar el nivell.

Per tant, només necessitem 2 mètodes. Un ha de controlar quants checkpoints s'han activat i, en funció d'això s'assigna l'habilitat que toca al PlayerPrefs de l'habilitat 1, i desactivi el canvas anterior i activi el nou. L'altre mètode s'ha de encarregar de que, si el jugador mor o polsa la tecla R (de reiniciar nivell), torni a la posició de l'últim checkpoint tocat. Podem veure el codi a les Figures 6.28 i 6.29.

```
void Marcador()
{
    if (checkpoints == 1)
    {
        canvas1.SetActive(false);
        PlayerPrefs.SetString("hab1", "DobleSalto");
        canvas2.SetActive(true);
    }

    if (checkpoints == 2)
    {
        canvas2.SetActive(false);
        PlayerPrefs.SetString("hab1", "Iman");
        canvas3.SetActive(true);
    }
```

Figura 6.28: Mètode de canvi d'habilitat a cada checkpoint

```
void Reset()
{
    if (Input.GetKey("r") || damageObject.Dead)
    {
        if (checkpoints == 0)
        {
            Player.transform.position = new Vector3(0f, 0f, Player.transform.position.z);
        }
        if (checkpoints == 1)
        {
            Player.transform.position = new Vector3(7f, 0f, Player.transform.position.z);
        }
```

Figura 6.29: Mètode per controlar si el jugador mor o vol reiniciar

6.1.7.3 Nivell 1 i Nivell 2.

Aquests nivells estan plantejats per ser a contratemps pel jugador, i la idea més important a tenir en compte en els controladors és gestionar la puntuació tenint en compte diversos factors:

- El primer factor és el propi temps. La puntuació comença en els dos casos en 2000 i va disminuint a cada segon. Si la puntuació arriba a 0, apareix el menú de pausa amb el missatge de que el jugador ha perdut.
- El segon criteri són les habilitats. Com s'ha comentat a l'Apartat 5.4, s'ha de fer un ús equitatiu de les habilitats que s'han seleccionat abans de començar el nivell. Això suposarà un descompte de punts al finalitzar el nivell. Aquest criteri es basa en fer una mitja entre el nombre d'usos de totes les habilitats i calcular un percentatge entre aquesta mitja i l'habilitat menys feta servir. La fórmula és la següent:

$$Puntuació\ final = Puntuació * \left(\frac{Min(hab1, hab2, hab3)}{(hab1 + hab2 + hab3) / 3} \right)$$

- Un altre criteri són les morts. A part de tornar al checkpoint anterior, el jugador perdrà 10 punts si mor, ja sigui per caiguda del mapa o per tocar un dels objectes que maten.
- El factor final sobre el que repercuteix la puntuació és un sistema "anti-tramposos". El jugador podria arribar a la part final del nivell i posar-se a fer ús de les habilitats per igualar el nombre d'usos total. Per tant, aquest sistema el que fa es que si es repeteix una habilitat en un període inferior de 2 segons el jugador rep una penalització de 5 punts i aquesta va augmentant de 5 en 5 si aquesta infracció es repeteix al llarg del nivell. Amb aquest sistema, entre que el temps juga en contra del jugador i que es perd puntuació, no surt a compte realitzar aquesta pràctica. A més, en el nivell 1, la puntuació disminueix 2,5 vegades més ràpid des del penúltim a l'últim checkpoint. Al nivell 2 no és necessari implementar això ja que hi ha les serres que venen per darrera fan que el jugador s'hagi de donar pressa en superar el nivell.

A l'igual que en el nivell de tutorial, en aquests s'ha de tornar a col·locar al jugador en cas que mori a la posició de l'anterior checkpoint aconseguit. Una altre cosa a tenir en compte és el menú de pausa implementat per aquests nivells. Al clicar la tecla Esc s'ha de mostrar el canvas corresponent i s'ha de interrompre el descens progressiu de la puntuació, així com congelar totes les constants del jugador mentre aquest menú es manté per a que no pugui avançar amb la pausa posada. El botó continuar fa que es torni a estar com abans de posar pausa, amb el descens de puntuació i les constants del jugador igual. El botó de reiniciar nivell reinicia l'escena i el botó de sortir va a l'escena del menú principal.

Finalment, si el jugador arriba al final del nivell amb punts, es mostra la finestra corresponent, amb un resum dels punts aconseguits. Al fer clic al botó de ok, abans de tornar a l'escena del menú principal, cal guardar la puntuació obtinguda al PlayerPrefs corresponent a la puntuació del nivell si és més gran que la que ja té guardada.

6.1.8 Moviment de càmera

L'script *cameraMovement.cs* assignat a la càmera utilitzada als nivells permet seguir el personatge del jugador. Ja que la idea és que el segueixi, necessitarem un GameObject públic que serà el del personatge, i recuperar la posició continuament l'Update. També hem fet servir una variable float smooth, juntament amb el mètode Lerp característic de Unity. Aquest ens permet fer que el moviment de la càmera sigui molt més agradable pel jugador, ja que sinó seria massa brusc al moure's de forma uniforme. A aquest mètode li hem de passar la posició inicial (la de la càmera en cada moment), la posició final (la del jugador) i el darrer paràmetre és aquest factor smooth multiplicat pel temps de joc. Veure el codi a la Figura 6.30.

```
public class cameraMovement : MonoBehaviour
{

    public GameObject Target;
    private Vector3 TargetPos;

    public float smooth;
    // Start is called before the first frame update
    void Start()
    {
        smooth = 2f;
    }

    // Update is called once per frame
    void Update()
    {
        TargetPos = new Vector3(Target.transform.position.x, Target.transform.position.y, transform.position.z);

        TargetPos = new Vector3(TargetPos.x, TargetPos.y, transform.position.z);
        transform.position = Vector3.Lerp(transform.position, TargetPos, smooth * Time.deltaTime);
    }
}
```

Figura 6.30 : Codi del cameraMovement.cs

6.1.9 Objecte que fa mal / fa morir al jugador.

Els objectes que poden fer que el jugador mori en el joc i torni a l'anterior checkpoint (que pel moment, en aquesta versió, només són les serres giratòries) tenen assignat l'script *damageObject.cs*. Aquest té un booleà estàtic Dead, que s'envia al controlador del nivell per a que, si el jugador ha mort, realitzi la tasca que hagi de fer. Per determinar-ho es fa servir el tag de "Player" al comparar l'objecte amb el que ha col.lisionat. També s'ha implementat que, si l'objecte amb el qual col.lisiona té el tag "Steel", aquest sigui destruit. Això s'ha implementat alhora d'estar desenvolupant el Nivell 2, en el qual les serres giratòries avancen i es poden tocar amb objectes. Resultava un problema que els objectes Steel fossin arrossegats. Per això, i com que tampoc es dona el cas de col.lisió entre aquests en cap altre nivell, s'ha implementat així, però en cas que això no convingués només caldria fer el mateix script sense aquest tros de codi, o controlar que només es faci en els nivells que ens interessa. Una darrera implementació és el temps des que el personatge mor fins que Dead torna a ser false. Això s'ha implementat com a motiu d'un bug, que també es comentarà en aquest document a l'apartat de dificultats trobades. Aquest bug feia que el jugador, al morir, esigués reapareixent continuament. Per això, al controlador de nivell, es deixa un temps superior a 1 segon des que el jugador reapareix i pot tornar a morir i dead torna a ser false un segon després d'haver mort. Veure el codi a la Figura 6.31.

```
public static bool Dead = false;
public float TimeDead = 0f;

void Start()
{
    TimeDead = 0;
}

private void OnCollisionEnter2D(Collision2D collision)
{
    if (collision.transform.CompareTag("Steel")) {
        Destroy(collision.gameObject);
    }

    if (collision.transform.CompareTag("Player"))
    {
        Debug.Log("Player Damaged");
        Dead = true;
    }
    else
    {
        Dead = false;
    }
}
```

```
private void OnCollisionExit2D(Collision2D collision)
{
    if (collision.transform.CompareTag("Player"))
    {
        Dead = false;
    }
}

void Update()
{
    if (Dead && (TimeDead < 1f))
    {
        TimeDead += 1 * Time.deltaTime;
    }
    else
    {
        Dead = false;
        TimeDead = 0;
    }
}
```

Figura 6.31 : Codi damageObject.cs

6.1.10 Configurador de nivell

Aquest objecte és el que s'encarrega de gestionar el funcionament del panell previ als nivells 1 i 2, el qual s'ha explicat al final de l'Apartat 5.6. Ja que aquests nivells permeten 4 habilitats diferents, cadascun té un controlador diferent, però el funcionament és el mateix.

Pel seu funcionament, assignem a cadascun dels botons de tria d'habilitats una funció que es ChangeHab(), on cada botó passa un string diferent segons l'habilitat corresponent. En aquesta funció, el que es fa és assignar a un dels PlayerPrefs de les habilitats actives, el string que s'ha passat. Amb el suport del PlayerPrefs Nhabs, el qual guarda un enter, això es pot implementar molt més fàcil, tenint el control de quantes habilitats tenim ja assignades en el moment que passem una nova, fins a un màxim de 3. Veure el codi a la Figura 6.32

```
void ChangeHab(string Habilidad)
{
    //Output this to console when the Button2 is clicked
    Debug.Log(Habilidad);
    Nhabs = PlayerPrefs.GetInt("Nhabs", 0);
    if (Nhabs == 0)
    {
        PlayerPrefs.SetString("hab1", Habilidad);
        PlayerPrefs.SetInt("Nhabs", 1);
    }
    else if (Nhabs == 1 && PlayerPrefs.GetString("hab1", "") != Habilidad)
    {
        PlayerPrefs.SetString("hab2", Habilidad);
        PlayerPrefs.SetInt("Nhabs", 2);
    }
    else if (Nhabs == 2 && PlayerPrefs.GetString("hab1", "") != Habilidad && PlayerPrefs.GetString("hab2", "") != Habilidad)
    {
        PlayerPrefs.SetString("hab3", Habilidad);
        PlayerPrefs.SetInt("Nhabs", 3);
    }
}
```

Figura 6.32: Codi d'assignació d'habilitat activa

Al botó vermell amb la creu li assignem una funció que s'encarrega de esborrar les habilitats assignades dels PlayerPrefs, i torna a posar el de Nhabs a 0. El botó verd, en canvi, dona lloc a l'escena del nivell. Veure codi a la Figura 6.33.

```
void Clear()
{
    PlayerPrefs.SetInt("Nhabs", 0);
    PlayerPrefs.SetString("hab1", "");
    PlayerPrefs.SetString("hab2", "");
    PlayerPrefs.SetString("hab3", "");
    txtH1.text = "-";
    txtH2.text = "-";
    txtH3.text = "-";
}

void Confirm()
{
    SceneManager.LoadScene("Nivel1");
}
```

Figura 6.33: Codi botons confirmar i esborrar

Finalment en aquest controlador, a l'Update portem 2 controls: un que manté deshabilitat el botó de començar el nivell si no tenim assignades 3 habilitats (cosa que podem controlar amb Nhabs), i canviar els textos de la part inferior esquerra del panell, els quals ens indiquen les habilitats que hem triat fins al moment. Veure codi a la Figura 6.34.

```
void Update()
{
    hab1 = PlayerPrefs.GetString("hab1", "");
    hab2 = PlayerPrefs.GetString("hab2", "");
    hab3 = PlayerPrefs.GetString("hab3", "");
    txtH1.text = hab1;
    txtH2.text = hab2;
    txtH3.text = hab3;
    Nhabs = PlayerPrefs.GetInt("Nhabs", 0);

    if (Nhabs != 3)
    {
        Confirmar.interactable = false;
    }
    else
    {
        Confirmar.interactable = true;
    }
}
```

Figura 6.34: Update del controlador previ al nivell

6.1.11 Gestionar controls d'habilitats

L'objecte CambiarControles és el que s'encarrega de gestionar els controls que tenen les diferents habilitats. Per això, com ja s'ha explicat, tenim el panell de canviar controls de joc, esmentat a l'Apartat 5.6. Pel funcionament, el que s'ha fet és agafar tots els PlayerPrefs de les tecles d'habilitats i fer que, a l'acabar d'editar el cuadre de text corresponent a cada habilitat, es cridi la funció ValueChangeCheck. Això es pot fer declarant-lo com a InputField i passant-lo com a objecte públic.

El mètode ValueChangeCheck es crida al canviar un d'aquests textboxs i crea una llista d'strings amb totes les tecles de les habilitats assignades. Si no es repeteix cap tecla, es pot polsar el botó de confirmar els canvis. Si no, no es pot sortir del menú, ja que dues habilitats no poden tenir assignada la mateixa tecla. Si es posa més d'un caràcter al checkbox, automàticament es torna a posar com estava abans d'editar. Veure el codi a la Figura 6.35

```
void ValueChangeCheck(InputField input, string tecla)
{
    if (input.text.Length == 1)
    {
        Debug.Log(tecla + ": " + input.text);
        PlayerPrefs.SetString(tecla, input.text);
        List<string> mylist = new List<string>(new string[] { TDash.text, TPlanear.text, TSu
            TIman.text, TImanArriba.text, TImanDer.text, TImanIzq.text, TImanAbajo.text, TC

        if(Repetido(mylist))
        {
            Confirmar.interactable = false;
        }
        else
        {
            Confirmar.interactable = true;
        }
    }
    else
    {
        input.text = PlayerPrefs.GetString(tecla, "z");
    }
}
```

Figura 6.35 : Codi confirmar canvi tecla

El booleà Repetido que veiem a la Figura 6.36 és un altre mètode, el qual fa una cerca per tota la llista per mirar si algun dels elements està repetit. Aixo es fa amb dos for(s) i comparant element per element. Veure a la Figura 6.36

```
bool Repetido(List<string> list)
{
    bool repetido = false;
    for (int i = 0; i < list.Count - 1; i++)
    {
        for (int j = i + 1; j < list.Count; j++)
        {
            if (list[i].Equals(list[j])) {
                repetido = true;
                return repetido;
            }
        }
    }
    return repetido;
}
```

Figura 6.36 : Codi cerca tecla repetida

El botó de confirmar assigna a tots els PlayerPrefs la tecla del corresponent Textbox. A la Figura 6.37 es pot veure algun exemple d'assignació d'aquests PlayerPrefs. No es mostren tots ja que és moltes vegades el mateix però amb cada habilitat.

```
void TaskOnClick()
{
    PlayerPrefs.SetString("TDash", TDash.text);
    PlayerPrefs.SetString("TPlanear", TPlanear.text);
```

Figura 6.37: Assignació de tecles als PlayerPrefs.

6.1.12 Implementació del botó

Els botons són uns objectes que en aquesta versió només trobem en el Nivell 2. Aquests tenen la capacitat d'obrir portes o treure obstacles al ser polsats, o bé per un objecte amb el tag "Steel" (amb els quals el jugador por fer servir l'habilitat Imant) o amb els cubs de construcció (els quals pot crear el jugador amb la corresponent habilitat). Aquests tenen un boxCollider2D, que és un trigger (es pot traspassar) i té un funcionament molt semblant al del *checkGround* o *checkwall*, però en aquest cas, simplement el que fa és que, al detectar la col·lisió, ja desactiva la porta u objecte corresponent, el qual passem com a variable pública. Veure el codi a la Figura 6.38

```
public class btndoor : MonoBehaviour
{
    public GameObject Puerta;

    private void OnTriggerEnter2D(Collider2D collider)
    {
        if (collider.transform.CompareTag("Steel") || collider.transform.CompareTag("Ground"))
        {
            Puerta.SetActive(false);
        }

    }
}
```

Figura 6.38 : Codi btndoor.cs

6.1.13 Controlador del menú principal

El controlador del menú principal s'encarrega de gestionar la interfície del menú principal, sense incloure les finestres emergents com els menús previs de configuració de nivell ni la finestra de canviar els controls de les habilitats. Aquí declarem tres textos i quatre botons que són interfícies susceptibles de canvi en el menú, segons el progrés del jugador. Els textos són l'indicador de tutorial completat i les puntuacions màximes aconseguides als nivells 1 i 2, i els botons són els que es polsen per indicar el nivell a jugar i el de reiniciar partida. Per portar el control de quin punt es troba el jugador agafem el suport del PlayerPrefs "EstadoJuego". Aquest simplement guarda un enter que pot ser de 0 a 3 i augmenta de 0 a 1 al superar el tutorial, de 1 a 2 al superar el nivell de prova d'habilitats i finalment de 2 a 3 al superar el Nivell 1 amb almenys 1000 punts. Segons en quin es troba, estan disponibles els nivells que hi ha en endavant o no, i s'han d'inhabilitar els botons que permeten accedir-hi. També es canvia el color dels textos de puntuacions a l'arribar als 1000 punts, ja que és l'objectiu. A la Figura 6.39 podem veure com s'inicialitzen els elements de la UI (botons i textos) i a la Figura 6.40 veiem els exemples de com s'implementa al update el canvi segons l'estat del joc. No es mostren tots els estats ja que és codi molt repetitiu.

```
public Text TutCompleted;
public Button ReiniciarPartida;
public Button NivelTodasHabilidades;
public Button Nivel1;
public Button Nivel2;
public Text Score1;
public Text Score2;

// Start is called before the first frame update
void Start()
{
    Nivel1.onClick.AddListener(delegate { Clear(); });
    Nivel2.onClick.AddListener(delegate { Clear(); });
    ReiniciarPartida.onClick.AddListener(Reset);
    Score1.text = "PUNTUACIÓN: " + PlayerPrefs.GetInt("Score1", 0);
    if (PlayerPrefs.GetInt("Score1", 0) >= 1000)
    {
        Score1.color = Color.green;
    }
    Score2.text = "PUNTUACIÓN: " + PlayerPrefs.GetInt("Score2", 0);
    if (PlayerPrefs.GetInt("Score2", 0) >= 1000)
    {
        Score2.color = Color.green;
    }
}
```

Figura 6.39 : Inicialització de variables i crida dels botons a les funcions

```
void Update()
{
    if (PlayerPrefs.GetInt("EstadoJuego", 0) == 1)
    {
        TutCompleted.text = "Completado";
        NivelTodasHabilidades.interactable = true;
        Nivel1.interactable = false;
        Nivel2.interactable = false;
    }
    else if (PlayerPrefs.GetInt("EstadoJuego", 0) == 2)
    {
        TutCompleted.text = "Completado";
        Nivel1.interactable = true;
        Nivel2.interactable = false;
        NivelTodasHabilidades.interactable = true;
    }
```

```
Score1.text = "PUNTUACIÓN: " + PlayerPrefs.GetInt("Score1", 0);
Score2.text = "PUNTUACIÓN: " + PlayerPrefs.GetInt("Score2", 0);
if (PlayerPrefs.GetInt("Score1", 0) >= 1000)
{
    Score1.color = Color.green;
}
else
{
    Score1.color = Color.red;
}
```

Figura 6.40 : Control d'estat de joc i canvis del menú principal

Com també podem observar a la Figura 6.40 els botons criden funcions d'aquest mateix script. El mètode Clear() el que fa és que treu les habilitats que tenia assignades fins ara el jugador, de manera que, quan obri el menú de selecció de nivell, no li surtin les que tenia assignades anteriorment. La funció Reset és la més radical, ja que esborra tot el progrés de la partida, inicialitzant tots els PlayerPrefs com es troben a l'estat inicial del joc. És a dir, esborra puntuacions i reinicia l'estat del joc a 0. També torna a posar els controls de les habilitats com venen per defecte a l'iniciar-se el joc.

6.1.14 Reproduir Vídeo

En els menús previs als nivells 1 i 2 tenim un panell a la dreta. Aquest panell reprodueix un vídeo de mostra del funcionament de l'habilitat corresponent al darrer botó d'habilitat polsat, de manera que visualment ajuda més al jugador a fer la tria abans del nivell. Per aquest funcionament el procediment ha estat, primerament, col·locar els vídeos en GameObjects amb el component Video Player. Després es mantenen inhabilitats fins que els habilitem per codi. Veure a la Figura 6.41

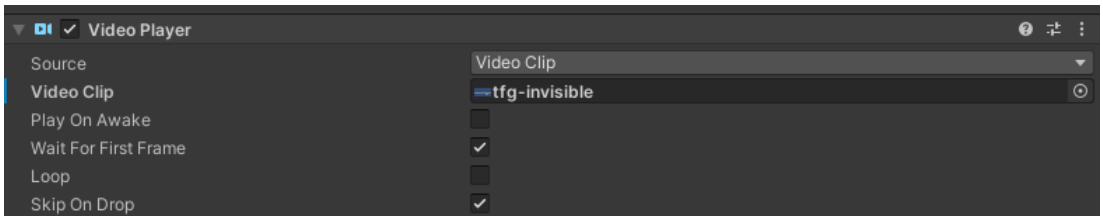


Figura 6.41: Implementació de Video Player al GameObject

Al afegir un component Video Player a l'objecte automàticament s'afegeix un Component Audio Source. Ja per codi, a l'script *streamvideo.cs*, el que fem es passar el RawImage (que és el tipus d'objecte del panell on es mostra el vídeo), el Vídeo Player i l'Àudio Source, de forma pública. Necessitem la llibreria de Unity UnityEngine.Video. Ara el que queda és cridar al Start() la funció que implementem per reproduir el vídeo, en aquest cas amb PlayVideo. Amb el següent codi de la Figura 6.42 es reproduirà al panell

```
public RawImage rawImage;
public VideoPlayer videoPlayer;
public AudioSource audioSource;
// Start is called before the first frame update
void Start()
{
    StartCoroutine(PlayVideo());
}
IEnumerator PlayVideo()
{
    videoPlayer.Prepare();
    WaitForSeconds waitForSeconds = new WaitForSeconds(1);
    while (!videoPlayer.isPrepared)
    {
        yield return waitForSeconds;
        break;
    }
    rawImage.texture = videoPlayer.texture;
    videoPlayer.Play();
    audioSource.Play();
}
```

Figura 6.42 : Codi streamvideo.cs

6.1.15 Controlador dels Checkpoints

Els checkpoints dels nivells tenen un script assignat que té un funcionament semblant al Checkground o Checkwall. En aquest cas, però, només interessa saber la primera vegada que el jugador el toca. Per tant, apart del booleà estàtic que el Checkpoint enviarà, en aquest cas, al controlador de nivell, guardarem un altre booleà per controlar que només s'envia un cop. Del contrari, puntuaria cada vegada que es toqués, cosa que no convindria. Aquests checkpoints tenen un objecte fill que conté l'sprite amb aquest activat. Veure el codi a la Figura 6.43 i la representació de l'objecte a la Figura 6.44

```
public static bool requestScore;
private bool check = false;

private void Start()
{
    check = false;
}

private void OnTriggerEnter2D(Collider2D collision)
{
    if(collision.CompareTag("Player") && !check)
    {
        GetComponent<SpriteRenderer>().enabled = false;
        gameObject.transform.GetChild(0).gameObject.SetActive(true);
        requestScore = true;
        check = true;
    }
}
```

Figura 6.43: Codi del Checkpoint.cs

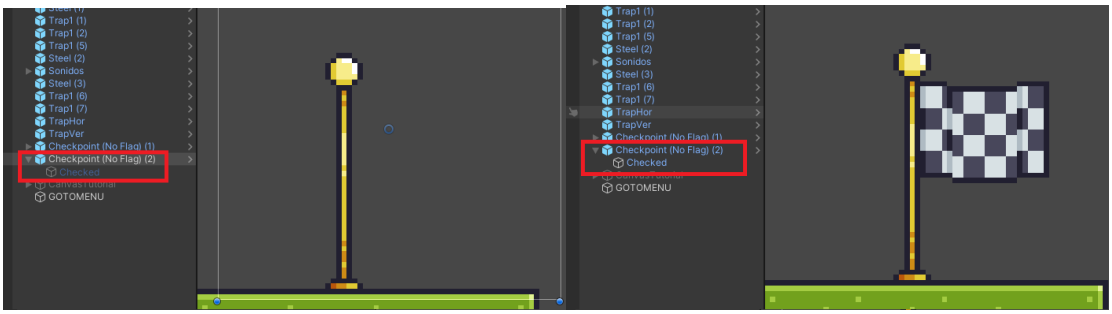


Figura 6.44: Objecte Checkpoint

6.2. Proves, dificultats trobades i solucions.

En l'apartat anterior s'ha descrit el que es fa en cada component del joc. En definitiva, com funcionen les mecàniques i tecnologies realitzades pel funcionament a la versió presentada en aquest projecte. En aquest apartat es comentaran les dificultats trobades en el desenvolupament d'algunes d'aquestes tecnologies, on s'ha hagut de fer canvis sobre el que hi havia previst o, directament, s'han descartat unes idees i s'han implementat d'altres. Alguns d'aquests canvis han estat conseqüència del feedback que han donat persones que han provat el joc.

També es comentarà alguna idea que s'ha implementat però que finalment s'ha optat per no afegir, ja que no encaixava amb el que s'ha implementat fins ara, però que es podria fer servir en futures versions.

6.2.1 Ús de totes les habilitats en el mateix nivell

La idea original per aquesta versió que, en definitiva, és una demo de cares a una futura implementació més gran, era que, després dels dos nivells que serveixen de tutorial, els nivells implementats tinguessin totes les habilitats disponibles. Això, al començar a implementar el nivell 1 em vaig donar compte que seria una implementació massa gran si es volien desenvolupar diversos nivells. El fet d'haver de contemplar totes les possibles combinacions de tres habilitats entre les possibles 8 a triar, feia que, a l'hora de posar un obstacle a superar, potser el jugador no podria superar-lo segons les habilitats que triés. Això tampoc em semblaria malament en una versió més completa, ja que el fet que el jugador hagi tornar a jugar el nivell fent ús de una combinació més eficient és una dels objectius a aconseguir. Però això podia fer que per aquesta versió algunes habilitats fossin totalment inútils per aquesta versió.

La primera idea en solució a això va ser realitzar dos camins alternatius dins d'un mateix nivell, cosa que hagués fet millors unes habilitats per un camí i unes altres per l'altre. Però aquesta idea finalment també es va descartar a causa que era molt difícil que un camí quedés molt més assequible que l'altre, tenint en compte el funcionament del sistema de puntuació.

Finalment es va optar per realitzar dos nivells, ja que no volia fer-ne només un perquè almenys quedés una idea de la progressió del jugador dins d'aquest i no fer un sol nivell que ocupés tot el temps de desenvolupament.

6.2.2 Dubtes sobre el funcionament de l'habilitat d'Escalar.

La idea original de l'habilitat d'escalar no era la que s'ha acabat implementant. Si bé és cert que el que es volia aconseguir era una mecànica que potenciés la capacitat del progrés en vertical, però el seu funcionament era diferent. La idea era que el jugador es pogués enganxar a la paret amb una tecla i que després pogués tornar a saltar. A l'estil de les parets blaves del Metroid Dread (veure Figura 8.1)

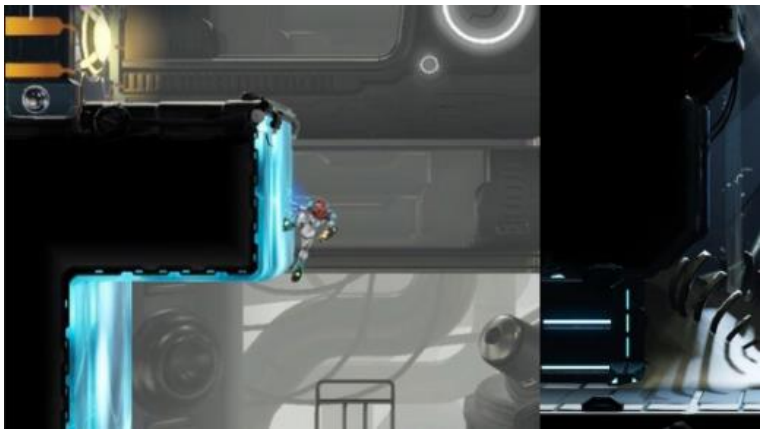


Figura 8.1: Mecànica parets blaves del Metroid Dread

De fet, l'habilitat estava implementada amb aquest funcionament però l'idea es va rebutjar després que altres persones provessin el joc. El problema residia en el funcionament i que era poc intuïtiva. Per enganxar-se el jugador havia de polsar repetidament la tecla i com que visualment tampoc hi havia una animació del personatge acord a la mecànica, no quedava clara la idea. Algunes de les persones que van provar el joc en primeres versions no van poder superar la part que s'havia de fer servir aquella habilitat en el nivell de prova d'habilitats, quan aquest nivell és un tutorial, a priori, senzill. Una segona idea va ser fer una espècie de ganxo, que visualment quedaria millor i més comprensible, però aquesta implementació era bastant més difícil per, a sobre, al moment del desenvolupament en que es va decidir canviar.

Finalment es va optar perquè simplement tingués el funcionament de apropar-se a la paret i escalar-la. El moviment en zig-zag fa que quedi més creible que si simplement s'arrossegués per la paret, i pel jugador és molt més intuïtiu.

6.2.3 Bug amb la reaparició

Com s'ha comentat a l'Apartat 6.9, durant el desenvolupament hi havia un problema a l'hora de reiniciar l'escena després de morir a causa d'un objecte que fa mal. El motiu pel que passava això, era que el booleà estàtic que es retorna des del *damageobject.cs* segueix essent true després de reiniciar-la. Per tant, el que es va fer va ser que al cap d'un segon torni a ser false, i als controladors de nivell es va crear una variable float Playing_time que no permet morir per aquest cas fins que no passen almenys 2 segons de joc. Això, de cares a la versió final del joc, no és un problema ja que no reiniciem l'escena directament, sino que canviem la posició del jugador a la del principi. Però de cares al disseny de nous nivells futurs que puguin tenir una implementació diferent (que si es mor tornes a començar directament) era un error important que ja s'ha solucionat.

6.2.4 Mecàniques no implementades.

Després de preparar un primer entorn bàsic de plataformes 2D, la feina a desenvolupar era començar a pensar quines habilitats implementar. Les que ja s'han vist eren algunes de les idees originals i d'altres (com ja s'ha comentat amb la d'Escalar) es van modificar respecte la idea original. Algunes de les pensades originalment que no es van arribar a implementar son:

- Atac cos a cos que permetés derrotar possibles enemics o trencar objectes, com els cubs de construcció, els imantables o altres a implementar.
- Paralitzar objectes per llençar-los després. Una mecànica semblant a la de Zelda Breath of the Wild.
- Interrompre salt. Una possible mecànica que enmig del salt fes un dash vertical cap a baix.
- Gravetat al revés. Una mecànica que canviés el sentit de la gravetat, tant pel personatge com per la resta d'objectes.

Una mecànica que també era una idea original era la de disparar a distància. Aquesta s'ha implementat però no es pot fer servir en el joc final, ja que seria útil per disparar a possibles enemics que tinguessin una intel·ligència artificial més complexa, cosa que finalment no es va implementar.

7. Resultats

7.1 Legislació i normativa vigent

El joc desenvolupat no presenta cap problema en aspectes legislatius. No es guarda informació de caràcter personal del jugador i, per tant, no és aplicable la Llei Orgànica de Protecció de dades.

El joc tampoc presenta cap problema de Copyright ja que els sons són d'ús lliure i els assets descarregats per a fer-lo són d'ús lliure pels desenvolupadors d'Unity.

7.2 Pegi

El sistema PEGI (Pan European Game Information) és un sistema de qualificació de videojocs d'Europa, que es fa servir per informar i aconsellar sobre el contingut i material interactiu dels mateixos a través d'etiquetes. Els principals fabricants de consoles com Nintendo, Sony o Microsoft respalden aquest sistema. A la següent Figura 7.1 podem veure les principals etiquetes que fan servir.

Figura 7.1: Etiquetes PEGI

En el cas del videojoc presentat en aquest projecte, el més correcte seria classificar-lo com a PEGI 3, a l'igual que molts jocs de Plataformes 2D actuals, ja que és apte per a totes les edats i no conté paraules malsonants ni escenes que no puguin presenciar nens petits.

7.3 Resultat.

A continuació es mostraran un seguit de Figures (de la 7.2 a la 7.9) on es poden veure tots els escenaris de joc, tal com seria tot el progrés, passant per tots els nivells i les diferents opcions de menú. A més es pot visualitzar un vídeo de mostra on es realitza un speedrun del joc realitzat pel creador al següent enllaç:

https://www.youtube.com/watch?v=CF2q48TtUMo

Figura 7.2: Menú principal principi del joc

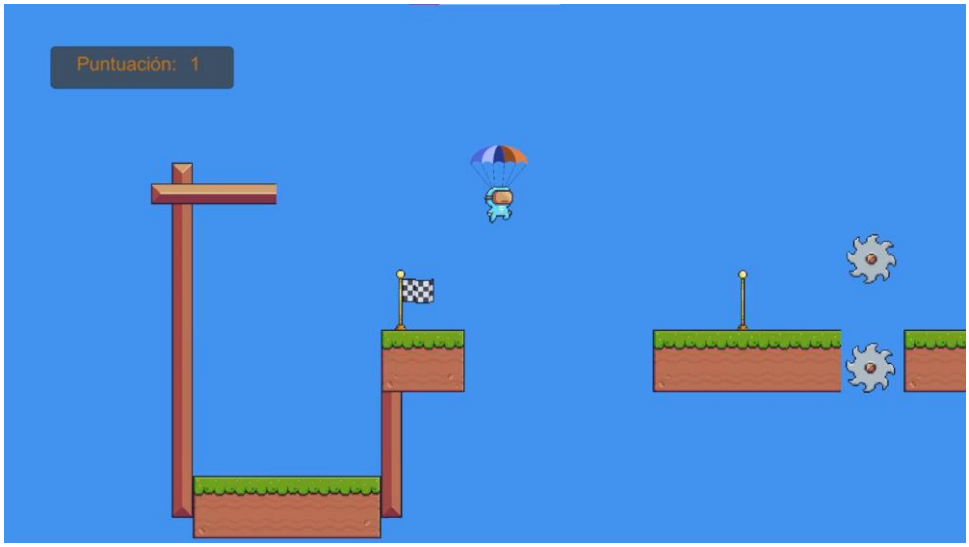


Figura 7.3: Nivell tutorial

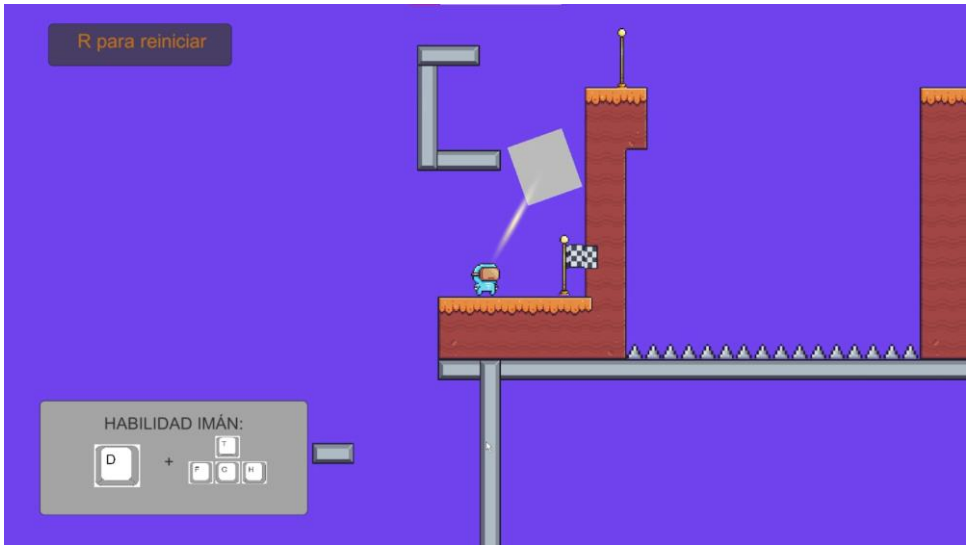


Figura 7.4: Nivell prova d'habilitats

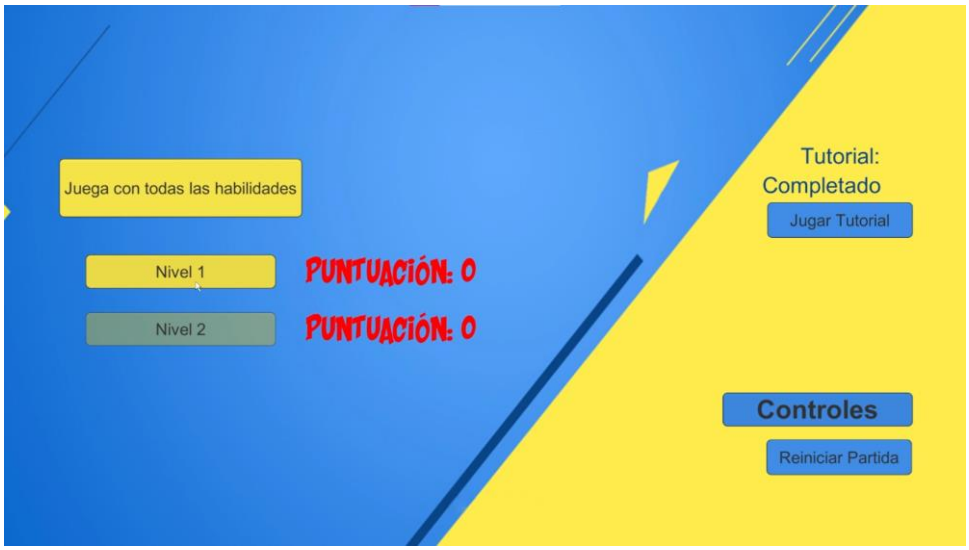


Figura 7.5: Menú principal al superar els dos nivells d'aprenentatge

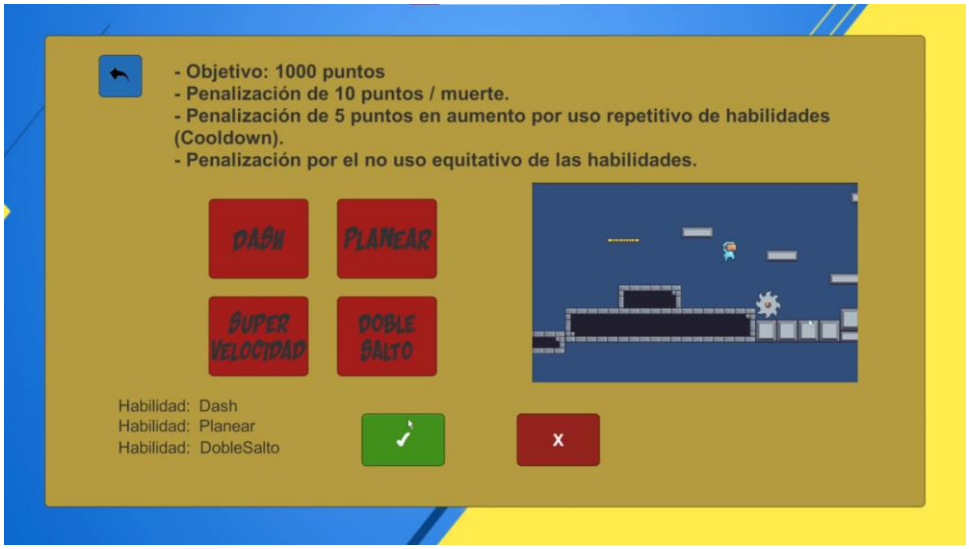


Figura 7.6: Menú de selecció habilitats nivell 1

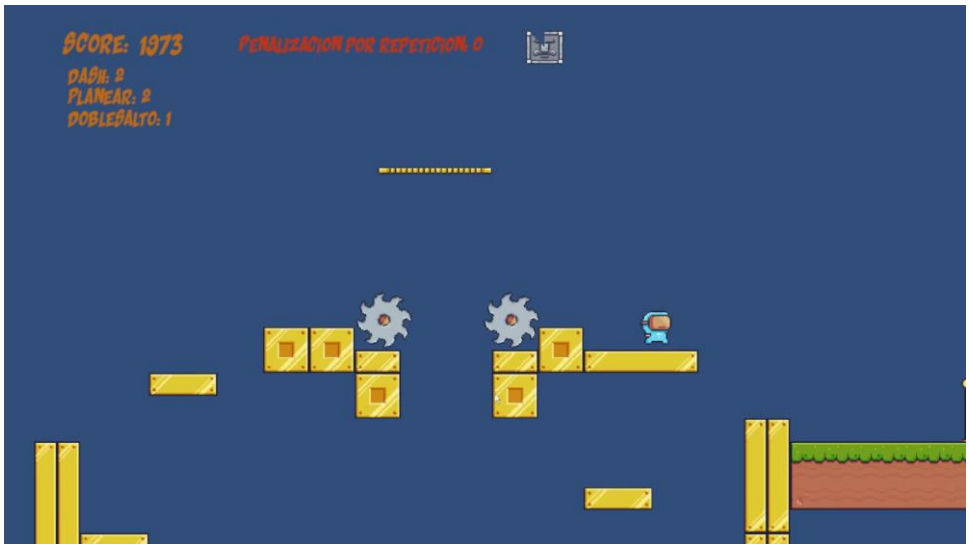


Figura 7.7: Nivell 1

Figura 7.8: Resum puntuació final nivell 1

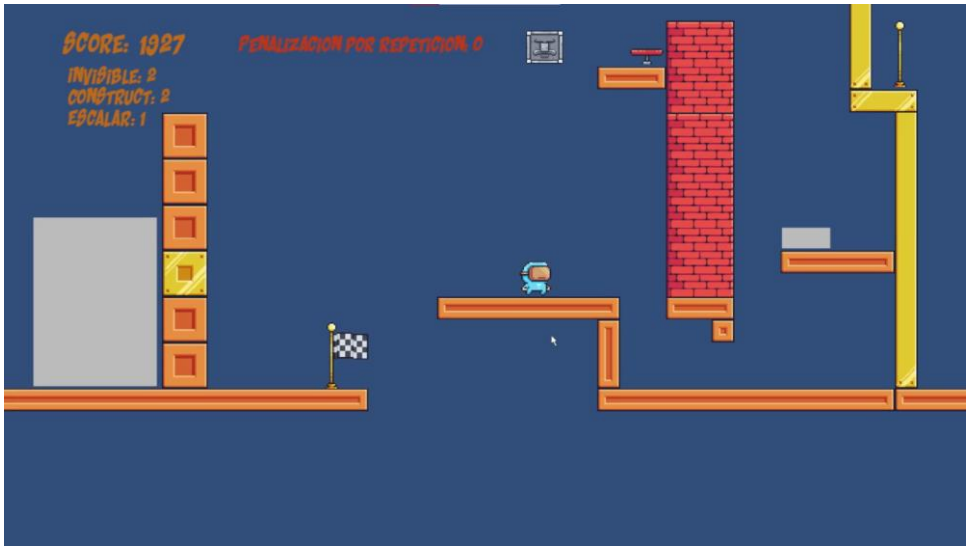


Figura 7.9: Nivell 2

8. Conclusions

8.1 Valoració del treball realitzat

A nivell personal i en línies generals, estic content amb la feina realitzada sobre aquest treball. La idea original era bastant més extensa de la que s'ha acabat realitzant, però en aquest tipus de projectes, i més treballant una sola persona, el temps és molt ajustat i el treball cal focalitzar-lo en els punts claus i en la idea principal que es vol que transmeti el joc, i això considero que ho he aconseguit.

El projecte ha estat sobretot enfocat a la part mecànica. Això fa que gran part del temps invertit, s'hagi de dedicar a molta prova i error. En principi pot semblar que no és gaire feina dissenyar una sola mecànica, però ajustar-la per a que sigui compatible amb l'espai de joc, amb la resta de mecàniques, que sigui eficient de la forma que desitja la persona que la programa i la solució de bugs i errors inesperats, treu molt de temps que, a priori, pot semblar que no seria tant.

Una de les grans conclusions que he tret és que, a l'hora de desenvolupar el joc, no és tan fàcil donar per fet que el jugador es sabrà adaptar. He tingut la sort que tres amics s'han ofert per provar-lo i m'han donat en diverses ocasions feedback de diferents elements que, a priori, per a mi eres coses òbvies, i al final no ho eren tant. Això ha tret temps de desenvolupament en refer certes coses que s'explicaran en el següent apartat, però m'ha ajudat a entendre de cares a futurs desenvolupaments que s'ha de tenir clar que no és el mateix pensar en el joc que es té al cap i com sabrà apreciar-lo el jugador.

Per tant, pels motius que he comentat, es pot treure en conclusió que la part negativa és pensar des d'un inici que el desenvolupament seria més senzill. Està clar que no és el mateix plantejar-se fer un videojoc amb tot el que això comporta (i això que no s'han tocat apartats com l'estètic o el narratiu), o fer algun treball de la universitat amb un motor de joc, ja que el fer el videojoc complet amb tot el que això comporta, té molts imprevistos i factors a tenir en compte.

8.2 Modificacions de la planificació inicial

La planificació inicial estava molt més enfocada a tractar tots els punts d'un videojoc. Gràcies a l'ajuda del meu tutor, he pogut donar-me compte a temps que m'havia d'enfocar sobretot en l'apartat mecànic i tecnològic. Llavors la idea va quedar en, que dels tres mesos de treball, la majoria del temps s'invertís en el disseny de nivells però, tot i que això s'ha complert, s'ha pogut dedicar menys temps de l'esperat.

També, com ja s'ha comentat, l'ajuda de altre gent que ha provat el joc m'ha ajudat a veure coses del joc que no funcionaven tant bé com jo creia. Això ha fet perdre temps en l'avanç del desenvolupament del joc, però canvis que, al cap i a la fi, s'han de realitzar. Per exemple, un jugador no era capaç de superar el nivell tutorial per culpa de la mecànica Escalar, que al principi era massa complicada. Això va suposar tornar a refer tota la mecànica, i adaptar-la als nivells que ja estaven fets, ja que no podia ser que un jugador no pogués superar el tutorial.

Per tant, la conclusió que se'n pot treure és que el temps de desenvolupament d'elements que no es consideraven tant importants o tant costosos en el global del projecte, ha fet que no hagi pogut dedicar tot el que plantejava al disseny de nivells. Tot i això, estic satisfet ja que s'ha pogut transmetre en gran mesura la que és la idea principal del joc i dona peu a futurs desenvolupaments que donen un ventall molt ample de possibilitats, treballant tant sobre les coses ja implementades, com incorporant-ne de noves.

9. Treball futur

La feina a futur a implementar en aquest joc seria força extensa, ja que les possibilitats del joc a partir de la idea de base són pràcticament exponencials. L'apartat visual i estètic també seria una part a millorar, ja que estaria bé donar una personalitat més pròpia al joc, cosa que, si es pogués fer juntament amb gent més especialitzada en aquest aspecte, seria de gran ajuda. Referent a les funcionalitats a afegir, podrien ser les següents:

- **Addició de nous nivells**: La versió actual del joc té nivells bastant curts i només tenen ús quatre habilitats a cadascun. Desenvolupar més nivells que donin la possibilitat d'utilitzar totes les habilitats i tinguin un ús equitatiu seria la principal tasca a dur a terme en futures implementacions. Per realitzar això, la idea dels camins alternatius que es comenta a l'Apartat 6.2.1 es podria reprendre. A més, en una versió definitiva, la idea seria començar el joc amb menys habilitats i a mida que s'avança en els nivells desbloquejar-ne de noves, però mantenint la idea original de poder fer-les fer servir totes en tots els nivells.
- **Nous objectes i/o enemics:** Els objectes que es troben pel mapa són imprescindibles per fer que el jugador aprofiti totes les habilitats. Segurament els primers a implementar podrien ser enemics amb intel·ligència artificial o, si més no, que segueixin algun patró.
- **Noves habilitats**: Els nous objectes o enemics donarien lloc a noves habilitats. Segurament realitzar atacs a distància o cos a cos serien les primeres a implementar per poder eliminar els enemics. També es podrien reprendre les idees comentades a l'Apartat 6.2.4, entre d'altres. Però en una implementació a gran escala s'haurien de pensar forces habilitats més.
- **Sistema de monetització i millora de les habilitats:** Un apartat important a implementar per la idea de rejugabilitat del joc seria un sistema de monetització, amb el qual el jugador obtenís una quantitat de monedes segons la puntuació obtenida en els nivells. O si es torna a rejugar amb diferents habilitats que es pogués obtenir més quantitat encara. La moneda es podria fer servir per desbloquejar noves habilitats, millorar les que ja es tenen o aconseguir elements estètics pel personatge.
- **Power ups dins dels nivells:** Altres elements que es podrien afegir dins els nivells podrien ser objectes que es poguessin recol.lectar per tenir temporalment una millora de les habilitats o recuperar puntuació. Això podria ajudar molt a fer els camins alternatius de manera que fossin equitatius i fos rentable agafar un camí que pot ser més llarg o més difícil.

10. Bibliografia

1. Unity technologies (2022) – *Unity Manual* *https://docs.unity3d.com/*
2. Unity technologies (2022) – *Unity Answers* *https://answers.unity.com/*
3. Unity technologies (2022) – *Unity Forum* *https://forum.unity.com/*
4. Game Maker's Toolkit (2022) – *Canal de Youtube* https://www.youtube.com/c/MarkBrownGMT
5. Mix and Jam (2022) – *Canal de Youtube: https://www.youtube.com/c/MixandJam*
6. Unity Store (2022) – *Pixel adventure 1* *https://assetstore.unity.com/packages/2d/characters/pixel-adventure-1-155360*
7. Unity Store (2022) – *Pixel adventure 2* *https://assetstore.unity.com/packages/2d/characters/pixel-adventure-2-155418*

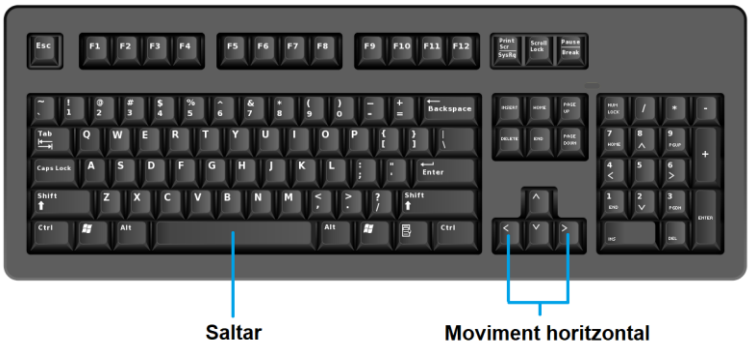
11 Annexos

Com a annexos tenim l'executable del videojoc i un vídeo de demostració de com superar el joc de forma relativament ràpida, realitzat pel propi creador del joc.

12. Manual d'usuari

A l'entrar en el joc podem visualitzar el menú tal com podem veure a la primera Figura de l'Apartat 7.3. A l'entrar el que s'ha de fer és realitzar el nivell de tutorial que es troba a la dreta de la pantalla. A partir d'aquí s'aniran desbloquejant els nivells disponibles dins d'aquesta versió del joc.

Ja dins de qualsevol dels nivells els controls bàsics són els que podem veure a la Figura 12.1



La resta dels controls per les diferents habilitats, tot i que es poden modificar al gust del jugador des del menú de controls, són els següents:

- Z: Dash
- X: Planejar
- V: Super-Velocitat
- A: Escalar
- Espai x2: Doble Salt
- 1: Invisible
- 2: Construcció
- D + T/F/G/H: Imant + moure

L'objectiu principal en aquesta versió és superar els Nivells 1 i 2 amb almenys 1000 punts, tot i que es poden tornar a jugar per intentar aconseguir la màxima puntuació possible.