
Universitat Oberta
de Catalunya

Modernització d'un clàssic: Sokoban

Autor: Marc Soler Baes
Tutor: Jordi Duch Gavaldà
Professor: Joan Arnedo Moreno

Màster en desenvolupament de videojocs
Disseny d'experiències de joc

18/06/2023

El present quadre de text té solament finalitats informatives i no ha de ser inclòs en la memòria de l'estudiant. Així mateix, aquesta pàgina tampoc ha de ser inclosa.

SOBRE ELS CONTINGUTS D'AQUEST DOCUMENT

Aquest document inclou estils predeterminats de text, exemples de cites bibliogràfiques, notes a peu de pàgina i inserció de figures (imatges i gràfics) i taules, així com secció de bibliografia i índexs automatitzats llests per usar.

SOBRE ELS CAPÍTOLS D'AQUEST DOCUMENT

Aquells apartats (i.e. capítols, apartats, subapartats, etc.) amb el títol en color negre són obligatoris per tots els TFP, mentre que aquells en color gris són opcionals, és a dir, susceptibles de ser inclosos en la memòria segons el tipus de TFP realitzat. És recomanable adaptar l'ordre dels capítols a la naturalesa del TFP a realitzar, i fins i tot combinar dos o més capítols en un si es considera oportú.

Tingueu en compte que el número màxim de pàgines que pot tenir la memòria és 90, incloent annexos i bibliografia.

Crèdits/Copyright

Una pàgina amb l'especificació de crèdits/copyright per al projecte (ja sigui aplicació d'una banda i documentació per l'altra, o unificadament), així com la de l'ús de marques, productes o serveis de tercers (inclusivament codis font). Si una persona diferent a l'autor va col·laborar en el projecte, ha de quedar explicitada la seva identitat i què va fer.

A continuació s'exemplifica el cas més habitual i una llista de possibles alternatives:

Aquesta obra està subjecta a una llicència de Reconeixement-NoComercial-SenseObraDerivada 3.0 Espanya de CreativeCommons

Llicències alternatives (triar alguna de les següents i substituir la llicència anterior)

A) CreativeCommons:

Aquesta obra està subjecta a una llicència de Reconeixement-NoComercial- SenseObraDerivada 3.0 Espanya de CreativeCommons

Aquesta obra està subjecta a una llicència de Reconeixement-NoComercial-CompartirIgual 3.0 Espanya de CreativeCommons

Aquesta obra està subjecta a una llicència de Reconeixement-NoComercial 3.0 Espanya de CreativeCommons

Aquesta obra està subjecta a una llicència de Reconeixement-SenseObraDerivada 3.0 Espanya de CreativeCommons

B) GNU Free Documentation License (GNU FDL)

FITXA DEL TREBALL FINAL

Títol del treball:	*Modernització d'un clàssic: Sokoban*
Nom de l'autor:	*Marc Soler Bages*
Nom del col·laborador/a docent:	*Jordi Duch Gavaldà*
Nom del PRA:	Joan Arnedo Moreno
Data de lliurament (mm/aaaa):	*06/2023*
Titulació o programa:	*Màster en disseny i programació de videojocs*
Àrea del Treball Final:	*Disseny d'experiències de joc*
Idioma del treball:	*Català*
Paraules clau	*Disseny, Modernització de jugabilitat, Adaptació*

Resum del Treball (màxim 250 paraules):

L'objectiu principal d'aquest treball és el de revisar les mecàniques del Sokoban, un joc de puzzles clàssic, per tal d'adaptar-les als estàndards de disseny actuals, permentent així veure i analitzar diferents maneres de modernitzar-les. La metodologia escollida per a desenvolupar aquesta adaptació del Sokoban és una variació del model Waterfall, on les fases de disseny, implementació i testeig aniran repetint-se per tal d'afegir mecàniques noves al producte amb cada iteració; començant amb un prototip similar al joc original i incrementant la diversitat de sistemes fins al moment de l'entrega. Quan arribem a la fase de desplegament, un dels objectius finals del projecte és posar-lo a la disposició del públic, i així veure com funciona el procés de publicació a dues plataformes en línia: Steam i la botiga de Google Play.
Vídeo: https://youtu.be/z3GmAneha-Q
Repositori: https://github.com/MSBeatles/TFM_SOKOBAN

Abstract (in English, 250 words or less):

The main goal of this project is to revise the mechanics of Sokoban, a classic puzzle game, in order to adapt them to current game design standards and to find and analyze the different ways of modernizing them. The chosen methodology to develop this adaptation of Sokoban is a slight variation of the Waterfall model, where the phases of design, implementation and testing will be iterated through, adding new mechanics to the product with each iteration; starting with a prototype similar to the original game and increasing the system diversity until the date of delivery. When we get to the deployment phase, one of the final goals of the project is to get it to the general public, thus learning how the publication process works for two on-line platforms: Steam and the Google Play Store.

Agraïments

Agraeixo molt l'ajuda que m'ha donat la meva germana Maria. Sense ella la part artística d'aquest treball hagués estat inexistent.

Abstract

L'objectiu principal d'aquest treball és el de revisar les mecàniques del Sokoban, un joc de puzzles clàssic, per tal d'adaptar-les als estàndards de disseny actuals, permentent així veure i analitzar diferents maneres de modernitzar-les. La metodologia escollida per a desenvolupar aquesta adaptació del Sokoban és una variació del model Waterfall, on les fases de disseny, implementació i testeig aniran repetint-se per tal d'afegir mecàniques noves al producte amb cada iteració; començant amb un prototip similar al joc original i incrementant la diversitat de sistemes fins al moment de l'entrega. Quan arribem a la fase de desplegament, un dels objectius finals del projecte és posar-lo a la disposició del públic, i així veure com funciona el procés de publicació a dues plataformes en línia: Steam i la botiga de Google Play.

Paraules clau

Disseny, Modernització de jugabilitat, Adaptació

Notacions i Convencions

Les paraules en idiomes que no són català estan en *cursiva*.

Els títols d'obres, a més d'estar en *cursiva*, estan *"entre cometes"*.

Els conceptes clau que cal destacar estan en **negreta**.

En la part d'implementació, els mètodes i scripts estan entre "entre cometes".

Índex

Figures i taules

1. Introducció

1.1. Introducció/Prefaci

«Sokoban» és un joc de puzzles dissenyat l'any 1981 per Hiroyuki Imabayashi, i publicat el desembre de l'any següent. El joc consisteix en desplaçar diverses caixes dins d'un magatzem en un patró seguint una graella, intentant fer que arribin a unes ubicacions marcades per punts. La graella és quadriculada, permetent al jugador moure's en quatre direccions: amunt, avall, a l'esquerra i a la dreta. Els moviments, a més, estan restringits per parets i les mateixes caixes que hem de moure: si intentem empènyer una caixa en una direcció i la següent casella no està buida, no podrem avançar.

A l'assignatura de **disseny de nivells**, una de les pràctiques consistia en programar un clon de «*Sokoban»* amb *Unity* per a crear 10 nivells diferents, cadascun més complicat que l'anterior i que ensenyés alguna estratègia nova dins de les mecàniques del joc. Fent aquesta pràctica em vaig adonar que, tot i ser un joc increïblement simple, hi havia molt de potencial per a afegir mecàniques noves que diversifiquessin les dinàmiques del *«Sokoban»* i creessin una jugabilitat més plena de possibilitats.

Actualment a la indústria s'han popularitzat la **remasterització** i el **rellançament** de diversos títols clàssics, com per exemple «*Resident Evil»*, *«Advanced Wars»*, «*Final Fantasy»* o «*Dead Space»*. Per tant, he pensat que era rellevant intentar portar aquest *trend* a l'extrem, fent el mateix però per a un joc encara més antic, on les mecàniques es veuen reduïdes al més bàsic que poden oferir els jocs de puzzles.

Finalment, els videojocs **són una forma d'art** que es troba en canvi constant, i això porta a que moltes obres caiguin en l'oblit o es perdin de l'existència degut al fàcil deteriorament del material necessari per jugar. Així doncs, pretenc ajudar a promoure el coneixement d'un joc clàssic mitjançant la creació d'aquesta remasterització, ja que considero que la **conservació** de la història del medi és essencial perquè aquest evolucioni en una bona direcció.

A la següent pàgina podem veure un esquema que representa un nivell de *«Sokoban»*. El que veiem representa:

·Cub blanc: Jugador

·Cubs negres: Parets

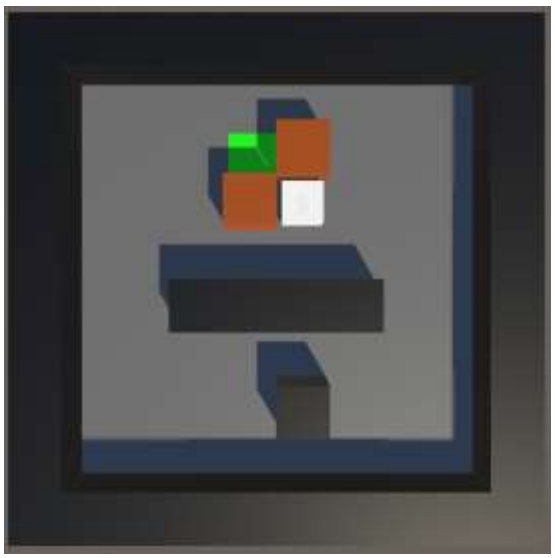
·Cubs marrons: Caixes

·Terra gris: Terra

·Terra verd: Metes

L'objectiu és moure les caixes fins que estiguin damunt de les metes.

Imatge 1: Nivell de "Sokoban" sense resoldre.



Imatge 2: Nivell de "Sokoban" mig resolt.

Imatge 3: Nivell de "Sokoban" resolt.

1.2. Descripció/Definició

El treball sorgeix de dues idees principals.

La primera és la de **modernitzar jocs clàssics** per a adaptar-los a la jugabilitat i al mercat actuals. Els últims anys, hem vist com la indústria dels videojocs virava cap a una tendència de **remasteritzacions i *remakes*** (*"Shadow of the Colossus"*, "*Resident Evil"*, "*The Last Of Us"*, "*Advanced Wars"*, etc). Aquestes noves interpretacions del joc original solen tenir una qualitat gràfica millorada i, a vegades, noves mecàniques que modifiquen el joc per a adaptar-lo als temps actuals. Una part important de l'èxit d'aquestes versions ve de la **nostàlgia** dels jugadors, que busquen reviure l'experiència de jugar un joc que recorden de quan eren més joves. L'altra part de l'èxit ve de nous jugadors que busquen poder viure el que persones més grans van viure amb la seva edat.

Aquest projecte no busca això exactament, sinó que intenta agafar aquesta tendència i dur-la a l'extrem reimaginant com podria haver sigut un joc clàssic sense les limitacions de l'època. No busca millorar-ne els gràfics, ni adaptar-ne les mecàniques, sinó que pretén expandir-les per tal d'incrementar exponencialment les possibilitats que ofereix la base del joc a l'hora de dissenyar-ne nivells.

La segona ve d'iniciatives com *Internet Archive*, i més específicament la *Video Game History Foundation* o la *Game Preservation Society*, associacions dedicades a **preservar**, celebrar i ensenyar la història dels videojocs. Una part important d'aquests projectes es centra en els videojocs *retro* (també coneguts com a videojocs clàssics), ja que tenen una característica que els fa molt peculiars: la diversitat del *hardware* necessari per a executar cada peça de *software*. Els videojocs són una forma d'art relativament nova, i increïblement canviant. Si establim una comparativa amb la literatura o el cinema, podem veure'n les diferències molt clarament:

·Al llarg de gairebé 150 anys d'història, el cinema ha passat per diverses fites i tecnologies, d'entre les quals en podem destacar el naixement del cinetoscopi l'any 1891 i el cinematògraf l'any 1895, la invenció del cinema en color i amb so, i l'arribada del cinema a la llar amb un seguit de tecnologies com els projectors, el *VCR*, el *LaserDisc*, el *VHS*, el *DVD* i, més recentment, el *Blu-Ray* o les plataformes de *streaming*.

·La literatura és una forma d'art encara més estàtica que el cinema. A part dels audiollibres i el llibre electrònic, nascuts a l'any 1932 i 1971 respectivament, la literatura sempre ha consistit de marques fetes sobre un material sòlid: ja sigui pedra, papir, canya de bambú o paper.

·El videojoc, tot i ser el més modern dels exemples utilitzats, és el que més canvis ha viscut i més ha evolucionat tecnològicament: des del *"tic-tac-toe"* creat per A.S. Douglas l'any 1952 fins a l'actual generació de consoles (que podríem resumir en *PlayStation5*, *Nintendo Switch* i *Xbox Series X/S*) i ordinadors podem trobar 9 generacions de consoles que contenen més d'un miler de sistemes diferents per a executar programari.

Això els converteix en un medi especialment **difícil de conservar**, ja que cada joc necessita una peça de maquinari específica per a executar-se, i aquestes es deterioren molt fàcilment amb el temps. A més, n'hi ha que van tenir molt poc èxit comercial, dificultant així la troballa d'unitats en bon estat.

Així doncs, la segona idea és que, mitjançant el rellançament d'un joc que ha anat caient en l'oblit del públic general, es **difongui el coneixement i la consciència** de la seva existència.

1.3. Objectius generals

1.3.1. Objectius principals

Objectius de l'aplicació:

- Adaptar un joc desenvolupat fa més de 40 anys als estàndards actuals de la indústria.
- Sortir al mercat amb un llançament a dues plataformes: *Steam* i *Google Play*.
- Ajudar a promoure la consciència sobre la conservació dels videojocs com a forma d'art.

Objectius per a l'usuari:

- Conèixer un joc que actualment no forma part de la cultura popular.
- Viure l'essència del joc original, però d'una manera completament nova.

Objectius personals de l'autor del TF:

- Créixer com a dissenyador de videojocs realitzant un exercici de pensament creatiu.
- Aprendre més sobre el procés de publicació de videojocs a les plataformes més populars del mercat.

1.3.2. Objectius secundaris

Objectius addicionals que enriqueixen el TF.

- Oferir un conjunt nou de possibilitats per a un joc que porta molt temps sense canvis.
- Donar a conèixer i revitalitzar l'interès per un joc que va marcar les bases del funcionament dels jocs de puzzles, però que ha quedat en l'oblit.
- Ajudar a la conservació en la memòria d'un videojoc clàssic que ha començat a caure en l'oblit.
- Aprendre a fer un projecte des de zero, i a treballar seguint una metodologia apropiada.

1.4. Metodologia i procés de treball

Per a desenvolupar aquest treball s'utilitzarà una modificació de la metodologia *waterfall*, on es repetiran les fases de disseny, implementació i testeig, utilitzant cada iteració per a afegir contingut al joc. La idea és la següent:

- **Primera iteració:** Implementació del joc original.
- **Segona iteració:** Disseny i implementació d'una mecànica nova.
- **Tercera iteració:** Disseny i implementació de nivells que utilitzin la mecànica nova.
- **Quarta iteració:** Disseny i implementació de la següent mecànica nova.
- **Cinquena iteració:** Disseny i implementació de nivells que utilitzin les mecàniques noves.
- **Etc.**

D'aquesta manera, adaptarem el producte original i el transformarem en el producte que volem aconseguir: un joc que mantingui l'essència de l'original però sentint-se fresc, actual i diferent.

Breument vaig considerar la modernització de jocs com *"Tetris"*, *"Bomberman"* o "*Space Invaders"*, però són jocs que ja han estat molt explotats i que són molt més coneguts per part del públic general.

Pel que fa a metodologies d'investigació, tot i que no són aplicables a un projecte així, sí que serà útil saber l'efectivitat i la originalitat de cada mecànica nova, així com la diversió que aporten al joc. Per això s'utilitzarà, una vegada acabat el treball, el *feedback* dels usuaris a les plataformes on es publicarà el producte.

Imatge 4: Representació gràfica de la metodologia Waterfall, extreta de indeed.com

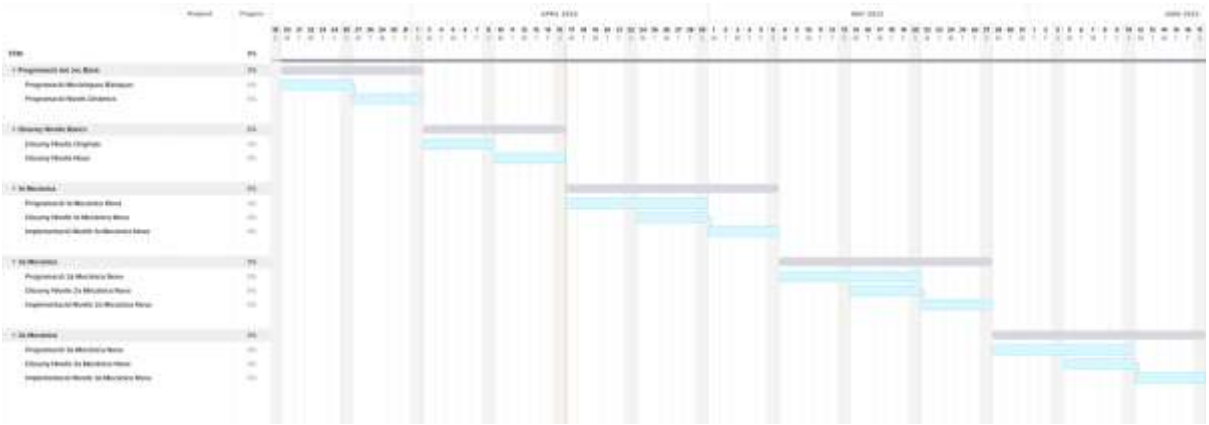
1.5. Planificació

La planificació del treball serà molt senzilla i seguint una estructura molt marcada per quinzenes: l'objectiu serà fer una iteració cada dues setmanes, començant per la setmana del 20 de març:

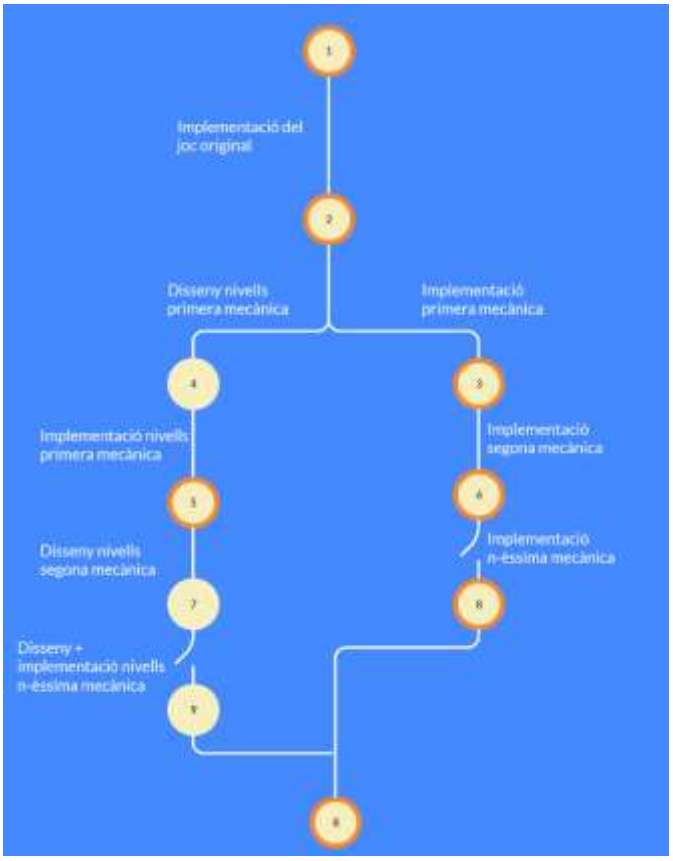
- **20/03-01/04:** Implementació del joc bàsic i una eina per generar els nivells dinàmicament
- **02/04-16/04:** Implementació dels nivells originals i 10 nivells nous
- **17/04-07/05:** Implementació de la primera mecànica nova, disseny de nivells corresponents
- **08/05-27/05:** Implementació de la segona mecànica nova, disseny de nivells corresponents.
- **28/05-17/06:** Implementació de la tercera mecànica nova, disseny de nivells corresponents.

Les fites (a part del final de cada iteració) seran les següents:

- **16/04:** Entrega de la PEC 2: Estat de l'art i primera versió del projecte
- **21/05:** Entrega de la PEC 3: Implementació de la versió jugable
- **18/06:** Entrega de la PEC 4: Memòria i productes finals
- **30/06:** Defensa del projecte



Imatge 5: Diagrama de Gantt del projecte..



Imatge 6: Diagrama de Pert del projecte.

1.6. Pressupost

Per al projecte de remasteritzar el *"Sokoban"*, podríem dividir el pressupost en 2 apartats: Material i Recursos Humans:

MATERIAL	PREU
Ordinador de sobretaula	1200 €
Pantalla	300 €
Ratolí	10 €
Teclat	40 €
	SUBTOTAL
	1550 €

HORES TREBALLADES	PREU/HORA
200h	15€
	SUBTOTAL
	3000€

PREU TOTAL	4550 €
PREU TOTAL (IVA INCLÒS)	5505,5 €

1.7. Estructura de la resta del document

La resta de la memòria seguirà la següent estructura:

Apartat 2: Anàlisi de mercat. En aquest apartat s'ha conduit una petita recerca per trobar xifres que recolzin el desenvolupament del projecte.
Apartat 3: Proposta. L'apartat 3 serveix per a proposar possibles implementacions que tindran lloc durant el desenvolupament del projecte.
Apartat 4: Disseny. Aquest apartat explicarà les decisions de disseny preses en l'aplicació pel que fa a l'UX i l'UI.
Apartat 5: Implementació. Aquest apartat és el més extens del projecte, i explica tota la part tècnica del projece.
Apartat 6: Demostració. En aquest apartat s'aclareixen termes que no han aparegut anteriorment.
Apartat 7: Conclusions. L'últim apartat és una reflexió sobre el treball i els punts que s'han aconseguit dels objectius inicials, i sobre l'après durant la realització del projecte.

2. Anàlisi de mercat

2.1. Públic objectiu (i.e. *target audience*) i perfils d'usuari

La versió que farem de *"Sokoban"* és un producte dirigit a tot el públic que té un ordinador o un mòbil. L'any 2021, el mercat dels videojocs va reportar un creixement de 180.3 mil milions de dòlars. Els jocs per a dispositius mòbils van representar més de la meitat dels beneficis totals de la indústria (un 52%, específicament) i els jocs per a ordinadors un 20%. És a dir, publicant el nostre videojoc per a mòbil i ordinador (tal i com hem proposat) estaríem arribant, potencialment, a un 72% de tots els usuaris de la indústria del videojoc.

A part de l'estat de creixement constant del mercat dels videojocs, *"Sokoban"* és un joc publicat a Japó, on va vendre més de 400.000 còpies. Si mirem a les mateixes xifres que abans, veurem que el 55% dels jugadors estan basats a l'àrea de l'est asiàtic (més concretament, la regió Àsia-Pacífic). A més, l'any passat Japó va ser el tercer país amb més ingressos en la indústria del videojoc, només per darrere de la Xina i dels EEUU, respectivament. Per tant, estarem publicant un joc que té especial interès a una de les regions del món amb més jugadors i ingressos al mercat.

Imatge 7: Mapamundi amb la regió Àsia-Pacífic destacada amb tonalitats verdes, extreta de la Viquipèdia*.*

Si ajuntem tot el ja vist amb el fet que publicarem una remasterització, podem arribar a la conclusió que el mercat està ple de potencials usuaris per al nostre joc, ja que, si mirem al que portem de l'any 2023, el 90% dels jocs amb millor puntuació de l'any són remasters i remakes. *"The Last Of Us Part I"*, per exemple, va ajudar a la sèrie a sobrepassar els 37 milions de còpies venudes dos mesos després del seu llançament. *"Resident Evil 4 Remake"* va vendre 3 milions de còpies en dos dies. Altres jocs com *"Dead Space"*, tot i

no tenir tant d'èxit com s'esperava, ha sigut top 3 jocs en vendes el mes que els van publicar. Si mirem els 10 jocs més venuts l'any 2023, 3 són remasteritzacions.

2.2. Competència, Antecedents i Estat de l'art

Actualment, *"Sokoban"* (com a franquícia) consta de 10 títols publicats, dels quals 2 han estat publicats únicament als EEUU, 6 al Japó i 2 a ambdós llocs. Després de la publicació del clàssic l'any 1982, diversos desenvolupadors han publicat fins a 9 versions oficials del joc, però cap d'elles ha innovat pel que fa a mecàniques. A la imatge 8 hi podem veure la llista. Després de l'última publicació l'any 1991, la saga va intentar ser rellançada per a tauletes i *smartphones* l'any 2016 amb *"Sokoban Touch"*, i aquest mateix joc va ser adaptat per a *Nintendo Switch* i *PlayStation4* els anys 2019 i 2021 (al Japó i als EEUU respectivament). Aquests rellançaments tampoc tenen cap mena d'innovació o canvi pel que fa a les mecàniques: Seguim tenint un mapa amb caixes, parets i quadrats on hem de posar les caixes, però no es fa cap variació sobre aquesta idea.

Year	Title	Country	Platform	Publisher	Media
1982	*Sokoban* (倉庫番)	Japan	NEC PC-8801	Thinking Rabbit	Cassette tape
1983	*Sokoban [Extra Edition]* (倉庫番[番外編][2][3])	Japan	NEC PC-8801	PCマガジン	Type-in program
1984	*Sokoban 2* (倉庫番2)	Japan	NEC PC-8801	Thinking Rabbit	Cassette tape
1988	*Soko-Ban*	US	IBM PC and compatibles	Spectrum HoloByte	Floppy
1989	*Soko-ban Perfect* (倉庫番Perfect)	Japan	NEC PC-9801	Thinking Rabbit	Floppy
1990	*Boxyboy*	Japan, US	Turbografx-16/PC Engine	NEC	HuCard
1991	*Soko-ban Revenge* (倉庫番Revenge)	Japan	NEC PC-9801	Thinking Rabbit	Floppy
2016	*Sokoban Touch* (倉庫番Touch)	Japan, US	Android and Apple iOS	Thinking Rabbit	Digital distribution
2019	*Minna No Sokoban* (みんなの倉庫番)	Japan	Nintendo Switch and PlayStation 4	Unbalance Corporation	Digital distribution
2021	The Sokoban	US	Nintendo Switch and PlayStation 4	Unbalance Corporation	Digital distribution

Imatge 8: Llistat d'edicions de "Sokoban" publicades, extret de la Viquipèdia.

En quant a jocs de puzzles, el sub-gènere està plagat de títols que destaquen en qualitat: *"Portal"* i la seva seqüela "*Portal 2"*, "*The Witness"* i "*Baba is you"* són 4 jocs que plantegen puzzles de manera original i creativa.

La mecànica principal de *"Portal"*, per exemple, és molt semblant al de "*Sokoban"*: Cal posar un cub damunt d'un botó. Aquesta mecànica, però, es veu revolucionada quan ens presenten la segona mecànica del joc: la pistola de portals.

Imatge 9: A "Portal" podem utilitzar una pistola per a disparar portals.

A "*The Witness"*, en canvi, el joc ens presenta puzzles que, tot i semblar simples laberints completament aïllats l'un de l'altre, estan relacionats amb el que veiem o sentim a l'escenari, i fins i tot amb accions nostres dins del món de joc.

Imatge 10: A "The Witness", l'escenari forma part dels puzzles.

Finalment, *"Baba is you"* és un joc de puzzles que utilitza frases per indicar-nos les accions que podem realitzar a cada nivell. La mecànica que el fa diferent és que ens permet moure les paraules d'aquestes frases per a modificar les propietats de cada element i així resoldre el puzzle. A la imatge d'exemple, les parets ens eviten arribar a la bandera, ja que *'Wall is stop'* (Paret és aturar). Per tant, per resoldre el puzzle podem redefinir les normes movent les paraules perquè, enlloc de posar *'Flag is Win'* (Bandera és guanyar), diguin *'You is win'* (Tu és guanyar), i guanyar sense haver d'arribar a la bandera.

Imatge 11: A "Baba Is You" es poden formar frases amb blocs i aquestes frases canvien la manera de funcionar del nivell.

Al llarg dels anys, els jocs de puzzles han evolucionat i crescut molt respecte a jocs com *"Sokoban"*. Aquest progrés només fa que el projecte que ens hem proposat sigui més interessant. Veure si els canvis i millores que implementarem aconsegueixen posar el *"Sokoban"* al nivell de complexitat de jocs com els anomenats anteriorment en aquesta secció serà un exercici fascinant.

2.3. Anàlisi DAFO

DEBILITATS	**FORTALESES**
·La mecànica base de *"Sokoban"* és extremadament simple ·És un joc del qual s'han fet moltes iteracions i això pot portar a falta d'interès ·S'ha de vigilar quins elements s'afegeixen: cal tenir en compte les sinergies entre ells	·El tenir una mecànica tan simple dóna molt marge de millora ·La programació base no hauria de ser extremadament complicada
OPORTUNITATS	**AMENACES**
·Si es planteja com una revolució/nova generació respecte de l'original pot generar expectativa ·Es pot aprofitar per a generar consciència sobre la conservació de l'original	·Si es genera expectativa però no s'aconsegueix complir pot impactar negativament de manera retrospectiva a l'original ·També pot provocar aversió per part del públic sobre la marca

3. Proposta

3.1. Definició d'objectius/especificacions del producte

L'objectiu d'aquesta remasterització de "*Sokoban"* és el d'innovar tot mantenint l'essència del joc original. Dit d'una altra manera: el jugador no haurà de realitzar cap acció diferent; podrà moure's amunt, avall, a l'esquerra i la dreta. El que modificarem serà les propietats del terreny per on es mou, afegint 3 dels següents canvis:

- **Caselles elementals:** Un conjunt de caselles relacionades amb elements de la natura que modificaran el comportament del jugador o de la caixa quan passi per damunt, entre els quals trobarem blocs de gel que fan que la caixa i el jugador llisquin, blocs de foc que eviten el pas del jugador (però no el de la caixa) i que cremen la caixa si està massa estona damunt, blocs amb aigua que permetran que una caixa passi per damunt d'un bloc de foc sense cremar-se gens, etc.
- **Caselles unidireccionals:** Un conjunt de caselles que només permetran el moviment en una direcció, evitant així que es creuin en direcció contrària, com si fossin rampes de moviment.
- **Trampes:** Làsers, punxes i més mecanismes que evitaran el pas del jugador a no ser que es compleixin certes condicions.
- **Portals:** Parets o caselles que transportaran al jugador o caixa que hi passi d'un lloc a un altre.
- **Duplicació de jugadors/caixes:** Nivells que tinguin dues entitats de jugador, que es mouran de manera sincronitzada.
- **Mapes dinàmics:** Mapes amb elements mòbils que es desplacin i obliguin al jugador a calcular el moment de moure's per travessar certs obstacles.

3.2. Estratègia de màrqueting

Pel que fa a l'enfocament del treball en qüestió de branding, la idea és crear publicacions a fòrums com *Reddit* per arribar a persones interessades en la resolució de puzzles o en *gaming* en general. També, i en menor mesura, s'escamparà el coneixement mitjançant el boca a boca d'amics i familiars, esperant arribar així a una audiència petita però fidel.

En quant a política de preus, depèn de les plataformes on es pujarà el joc: *Google Play* i *Steam*, que tindran dues estratègies diferents.

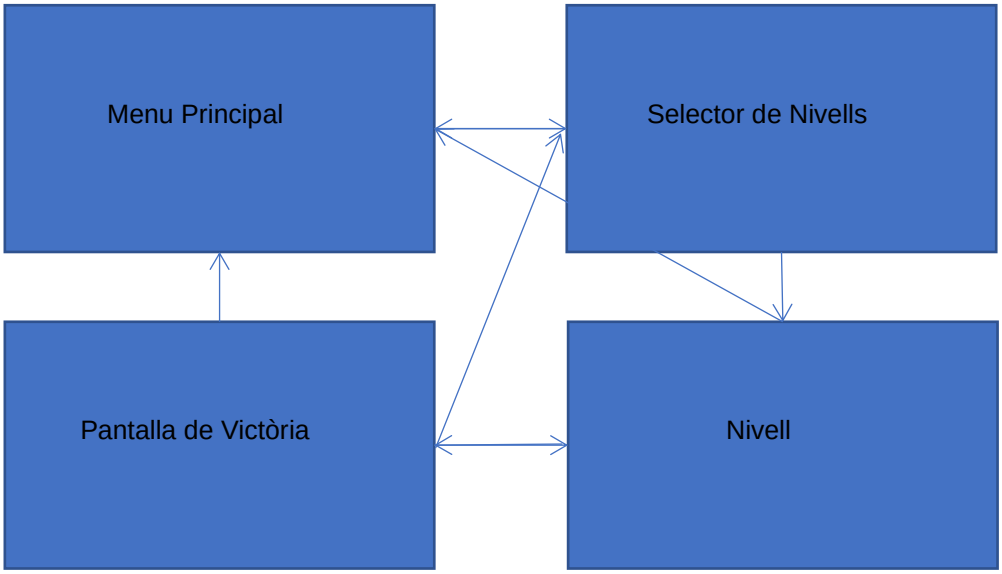
Steam: El joc estarà disponible per un preu entre 4.99€ i 14.99€, depèn de la quantitat de nivells i la qualitat del producte final. *Valve* obté una comissió del 30% per cada còpia venuda, així que obtindria entre 3.50€ i 10.50€ bruts per cada venda.

Google Play: El joc estarà disponible de manera gratuïta, però monetitzat amb anuncis no invasius (banners en menús,etc). Els beneficis dependran de les impressions d'anuncis que tingui el joc, i per tant dependran directament del nombre d'usuaris actius diàriament.

4. Disseny

4.1. Arquitectura general de l'aplicació/sistema/servei

El projecte dissenyat consisteix de diverses escenes, cadascuna amb una funcionalitat encapsulada. La primera escena és el menú principal, on el jugador podrà triar entre sortir de l'aplicació o passar a la pantalla de selecció de nivell. L'escena de selecció de nivell donarà una llista de botons al jugador, que podrà prémer el que vulgui per a entrar al nivell pertinent. Finalment, el nivell es generarà dinàmicament a partir d'un fitxer que en contindrà el disseny, i quan el jugador l'acabi apareixerà la pantalla de victòria. En qualsevol moment el jugador podrà tornar al menú principal.



Imatge 12: Diagrama de flux de l'aplicació.

4.2. Arquitectura de la informació i diagrames de navegació

La informació més important que es guarda en el projecte es troba en els fitxers que contenen el disseny dels nivells, emmagatzemats a la carpeta d'StreamingAssets del projecte de Unity. Per tal de generar els nivells dinàmicament s'ha implementat un script anomenat "LoadLevel.cs" que llegeix els fitxers i instancia els objectes necessaris per a crear el nivell.

A part d'això, hi ha un objecte anomenat "tiles". És una llista que guarda el contingut de cada casella a l'script que controla el moviment del jugador, i que serveix per a realitzar totes les verificacions necessàries a cada torn.

4.3. Disseny gràfic i interfícies

Com que el Sokoban és un joc tan genèric, es pot adaptar el seu disseny gràfic a moltes situacions. En el marc del projecte, i per falta de temps, només n'he creat una de predeterminada, però la idea seria, amb més temps i persones treballant-hi, donar monedes al jugador a mida que vagi passant nivells i ficar una tenda de "skins" dins del joc que li permetés comprar diferents aspectes gràfics per als nivells. Cadascun d'aquests aspectes podria tenir una paleta de colors determinada o una temàtica específica. Per a l'entrega del meu TFM, he decidit crear-ne una amb temàtica de platja i mar com a exemple d'una possibilitat, però la gràcia del Sokoban és que no té un setting concret i es podria adaptar a un nombre immens de situacions i temàtiques diferents.

Imatge 13: Nivell amb paleta de colors de temàtica de platja.

4.3.1. Usabilitat/UX

Per a facilitar la usabilitat del joc, s'han creat dos mapes de controls: Un per a teclat i ratolí i un per a comandament. A més, els menús són clars i concisos, intentant fer-los minimalistes per tal d'evitar confondre a l'usuari.

També s'ha creat una interfície d'usuari que compleix el principi de les 5 E's de l'experiència d'usuari:

- **Effective:** El disseny permet a l'usuari realitzar totes les funcions sense cap dificultat (excepte la implícita dels jocs de puzzles)
- **Efficient:** El disseny permet a l'usuari arribar a la pantalla objectiu amb el nombre mínim de botons premuts.
- **Engaging:** Com que l'estat mental necessari per a jugar a un joc de puzzles és molt relaxat, i solen ser jocs que no requereixen de reflexes ràpids ni provoquen frustració, sinó que només demanen agilitat mental i paciència, existeix la possibilitat que els jugadors s'avorreixin ràpid o perdin l'interès. Una manera de mantenir-los "engaged" és aportar un bon apartat gràfic, amb una interfície adaptable i personalitzable, perquè puguin veure el joc de la manera que més agradable els resulti en aquell moment.
- **Error tolerant:** El disseny del flux de l'aplicació permet a l'usuari desfer qualsevol acció en cas d'entrar a un menú al que no volia accedir. El mateix és aplicable al joc, que permetrà al jugador reiniciar un nivell prement R. (encara no implementat)
- **Easy to learn:** Els controls del joc són molt senzills i intuïtius, ja que l'únic que ha de fer el jugador és prémer la tecla de la direcció en la que es vol moure. A més, el disseny dels nivells està fet per tal que el jugador aprengui quin és l'objectiu del joc sense necessitat d'explicar-li, només guiant-se per el *layout* del nivell.

4.4. Llenguatges de programació i APIs utilitzats

Naturalment, per a la creació del projecte s'ha emprat Unity, ja que és l'única eina de desenvolupament de videojocs que hem après a utilitzar durant el màster. Per a la part de programació s'ha utilitzat C#.

4.5. Disseny de mecàniques de joc

Considero important comentar quines són les noves mecàniques que s'han dissenyat per a modernitzar Sokoban. Tot i que el resultat definitiu no conté totes les mecàniques possibles proposades originalment, s'han implementat les següents:

- **Portals:** La mecànica de portals funciona igual que al videojoc "Portal". Hi ha blocs que, enlloc de ser paret per les 4 bandes, tenen una superfície d'un color diferent a un lateral. El jugador pot entrar per aquestes superfícies laterals (o empènyer caixes a través d'elles) i sortir-ne per una altra, que està emparellada a la primera.
- **Caselles elementals:** Aquestes caselles poden ser de foc o de gel. Les de foc actuen com una paret per al jugador. En canvi, les caixes sí que poden passar a través d'elles. Les caselles de gel provoquen que qualsevol objecte que hi passi per damunt rellisqui fins que surti del gel o trobi un bloc que l'aturi.
- **Duplicació:** Hi ha situacions en les que el jugador apareix duplicat al mapa, i els dos jugadors es mouen alhora. Així es poden crear puzzles on s'ha de sincronitzar el moviment d'ambdós jugadors, o trobar un camí que funcioni per un i per l'altre per separat.

5. Implementació

Per a aquesta secció del treball considero que cal explicar les següents parts de l'aplicació en profunditat:

a) El creador de nivells dinàmic
b) El sistema d'esdeveniments
c) El moviment del personatge/mapa del nivell
d) Les mecàniques noves
e) Les interfícies gràfiques.

Així doncs, l'apartat d'implementació constarà de 5 subseccions que explicaran en detall cadascun dels punts llistats.

5.1. Creador de nivells dinàmic

Per a facilitar la part de disseny del joc, el primer que vaig fer va ser implementar un script anomenat "LoadLevel.cs", que s'encarrega de generar els nivells basant-se en uns fitxers de text que hi ha a la carpeta "StreamingAssets" del projecte.

Aquests fitxers consisteixen principalment de 3 elements, seguint sempre una estructura similar a la següent:

```
1   7
2   9
3   P F G W W W W
4   F F F G W W W
5   F F F F G W W
6   W B F F F G W
7   F F B F F F G
8   F F F B F F F
9   W F F F B F F
10  W W F F F B F
11  W W W F F W F
```

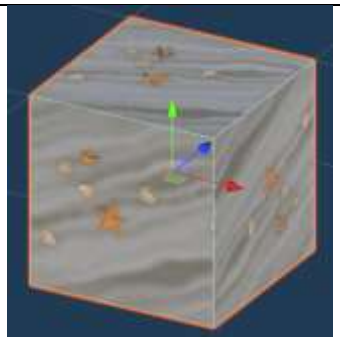
Imatge 14: Fitxer exemple d'un nivell del joc.

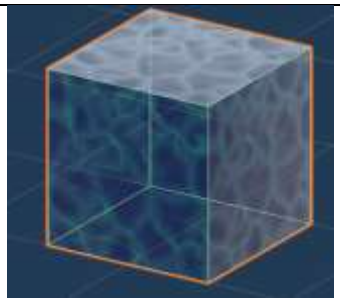
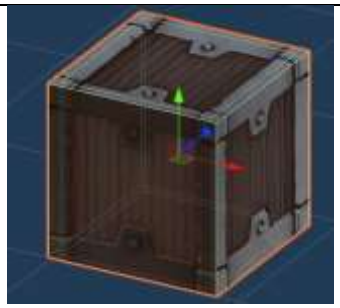
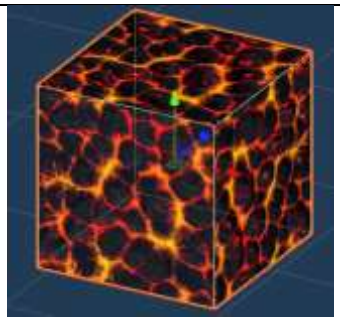
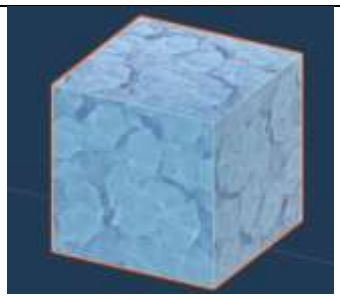
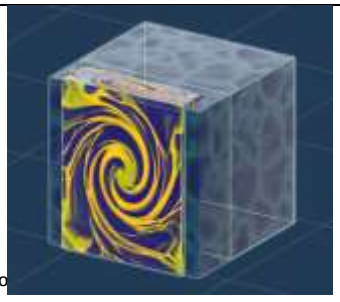
A la imatge podem veure que les dues primeres línies del fitxer són dos números enters. Aquests números diran a l'script "LoadLevel.cs" la quantitat de caselles que tindrà el nivell.

Podríem considerar-los les coordenades X i Y en un pla 2D. Després, el fitxer conté una matriu de lletres.

Aquesta matriu servirà per indicar al generador de nivells quin bloc cal posar a cada posició. Al principi, només hi havia 5 tipus de blocs: Player, Goal, Floor, Wall i Box. Quan el projecte va anar creixent i van aparèixer-hi noves mecàniques, va caldre afegir blocs nous.

Al final, cada lletra es correspon a un bloc diferent de la següent manera:

LLETRA	**BLOC**	**IMATGE**
"P"	Player	
"G"	Goal	
"F"	Floor	

"W"	Wall	
"B"	Box	
"Q"	Fire	
"I"	Ice	
"L"	Left Portal	
"R"	Right Portal	
"U"	Up Portal	
"D"	Down Portal	

Cal esmentar que no tots els blocs funcionen igual, i això porta a anomalies. Per exemple, quan al nivell hi ha un o més parells de portals, cal indicar com estan connectats. Per a fer-ho, cada portal va acompanyat de les coordenades de la seva parella. Aquesta implementació és especialment útil perquè dóna flexibilitat a l'hora de dissenyar els nivells, permetent-nos posar més d'un parell de portals, o fins i tot posar grups de més de dos portals que funcionen de manera rotatòria (és a dir, entrar pel portal A provoca que surtis pel portal B, entrar pel portal B fa que surtis pel C, i així successivament fins que entrar al portal X fa que surtis per l'A).

El resultat és que, en els nivells amb portals, acabem amb un fitxer d'aquest estil:

```
7
7
W W W U36 W W W
W W W G W W W
W W W W W W W
R63 G W P F B L03
W W W F W W W
W W W B W W W
W W W D30 W W W
```

Imatge 15: Fitxer exemple d'un nivell del joc amb portals.

Tornant a l'script "LoadLevel.cs", funciona d'una manera relativament senzilla un cop s'ha vist un fitxer de nivell. Primer mira a les "PlayerPrefs" quin nivell ha seleccionat l'usuari a la pantalla de selecció de nivell. Busca el fitxer associat a aquell nivell i comença a llegir-lo. Té dues variables que indiquen el nombre de files i columnes del nivell, i les assigna quan llegeix les dues primeres línies:

```
path = Application.dataPath + "/StreamingAssets/LevelDesigns/" + PlayerPrefs.GetString("ChosenLevel") + ".txt";
//Debug.Log(path);
StreamReader reader = new StreamReader(path, true);


//Primer de tot agafem maxX i maxY
string line;
line = reader.ReadLine();
int.TryParse(line, out maxX);
PlayerPrefs.SetFloat("Width", maxX);
onSetX.Raise(this, maxX);
line = reader.ReadLine();
int.TryParse(line, out maxY);
PlayerPrefs.SetFloat("Height", maxY);
onSetZ.Raise(this, maxY);
```

Imatge 16: Part de l'script LoadLevel.cs on es comença la lectura del fitxer de nivells.

A partir d'aquí, itera utilitzant aquesta informació i va instanciant prefabs dels blocs a les coordenades corresponents del nivell:

```
for (int j = 0; j < maxY; j++)
{
    int i = 0;
    line = "";
    line = reader.ReadLine();
    //Debug.Log(line);
    for (int k = 0; k < line.Length; k++)
    {
        if (line[k] == 'W')
        {
            //myObjects[i, j] = Instantiate(wall, new Vector3(i, 1.0f, j), Quaternion.identity);
            Instantiate(wall, new Vector3(i, 1.0f, j), Quaternion.identity);
            Instantiate(floor, new Vector3(i, 0.0f, j), Quaternion.identity);
            //tiles[i, j] = Tile.Wall;
            i++;
        }
        else if (line[k] == 'G')
        {
            //myObjects[i, j] = Instantiate(goal, new Vector3(i, 0.0f, j), Quaternion.identity);
            Instantiate(goal, new Vector3(i, 0.0f, j), Quaternion.identity);
            //tiles[i, j] = Tile.Goal;
            onGoalSet.Raise(this, 0);
            i++;
        }
        else if (line[k] == 'F')
        {
            //myObjects[i, j] = Instantiate(floor, new Vector3(i, 0.0f, j), Quaternion.identity);
            Instantiate(floor, new Vector3(i, 0.0f, j), Quaternion.identity);
            //tiles[i, j] = Tile.Floor;
            i++;

        }
```

Imatge 17: Part de "LoadLevel.cs" on es llegeix l'scipt, i s'instancien els objectes necessaris.

Un cop el generador de nivells ha llegit tot el fitxer, instancia un conjunt de terres i parets al voltant del nivell, creant un mur:

```
for (int j = -1; j <= maxY; j++)
{
    Instantiate(wall, new Vector3(-1, 1.0f, j), Quaternion.identity);
    Instantiate(wall, new Vector3(maxX, 1.0f, j), Quaternion.identity);
    Instantiate(floor, new Vector3(-1, 0.0f, j), Quaternion.identity);
    Instantiate(floor, new Vector3(maxX, 0.0f, j), Quaternion.identity);
}
for (int i = -1; i <= maxX; i++)
{
    Instantiate(wall, new Vector3(i, 1.0f, -1), Quaternion.identity);
    Instantiate(wall, new Vector3(i, 1.0f, maxY), Quaternion.identity);
    Instantiate(floor, new Vector3(i, 0.0f, -1), Quaternion.identity);
    Instantiate(floor, new Vector3(i, 0.0f, maxY), Quaternion.identity);
}
```

Imatge 18: Part de l'script "LoadLevel.cs" on s'instancien els objectes que envolten el nivell.

Vaig decidir implementar el generador de nivells dinàmic per diverses raons. Primer de tot, ens permet tenir una escena semi-buida anomenada "Level" i no cal tenir una escena per cada nivell. Segon, facilita la creació, el testeig i la modificació de nivells nous, ja que no és necessari crear-los a mà des de l'editor de Unity, simplement cal escriure un petit fitxer de text. Finalment, també simplificaria molt la implementació d'un creador de nivells que permetés al jugador fer els seus propis nivells i compartir-los amb altres jugadors de la comunitat. Aquesta opció no ha estat implementada per falta de relació amb l'objectiu del treball, però si el projecte segueix endavant serà un dels aspectes prioritaris.

5.2. El sistema d'esdeveniments

Per a mantenir un bon nivell d'abstracció i d'encapsulació dins del projecte, vaig implementar un sistema d'esdeveniments i listeners aprofitant les classes proporcionades per Unity. Aquest sistema consisteix en dues classes anomenades GameEvent i GameEventListener, que permeten que diversos objectes es comuniquin sense trencar l'abstracció del codi:

```
[CreateAssetMenu(menuName = "GameEvent")]
public class GameEvent : ScriptableObject
{
    public List<GameEventListener> listeners = new List<GameEventListener>();

    //Raise event through different methods signatures

    public void Raise(Component sender, object data)
    {
        for (int i = 0; i < listeners.Count; i++)
        {
            listeners[i].OnEventRaised(sender, data);
        }
    }


    //Manage listeners

    public void RegisterListener(GameEventListener listener)
    {
        if (!listeners.Contains(listener))
        {
            listeners.Add(listener);
        }
    }

    public void UnregisterListener(GameEventListener listener)
    {
        if (listeners.Contains(listener))
        {
            listeners.Remove(listener);
        }
    }

}
```

Imatge 19: Script on definim els GameEvents del joc.

```
6     [System.Serializable]
7     public class CustomGameEvent : UnityEvent<Component, object> {}
8
9     public class GameEventListener : MonoBehaviour
10    {
11        public GameEvent gameEvent;
12
13        public CustomGameEvent response;
14
15        private void OnEnable()
16        {
17            gameEvent.RegisterListener(this);
18        }
19
20        private void OnDisable()
21        {
22            gameEvent.UnregisterListener(this);
23        }
24
25        public void OnEventRaised(Component sender, object data)
26        {
27            response.Invoke(sender, data);
28        }
29    }
```

Imatge 20: Script on definim els escoltadors dels Events del joc.

5.3. El moviment del personatge i mapa del nivell

La part més complicada del projecte ha estat trobar una bona manera de comprobar quines accions podia realitzar el jugador i quines li estaven prohibides per les normes del joc. Aquest problema el vaig resoldre a l'script "PlayerMovement.cs". La primera implementación va ser molt senzilla, ja que només hi havia 5 tipus de bloc i les normes estaven reduides a 'el jugador sempre es pot moure en la direcció objectiu, a no ser que hi hagi una paret al següent bloc. Si hi ha una caixa al següent bloc, es podrà moure sempre i quan després de la caixa no hi hagi ni una paret ni una caixa.'

Així doncs, vaig crear una llista de dues dimensions anomenada "Tiles", que feia ús d'un Enum amb tots els tipus de casella per guardar una mena de "mapa" de la situació inicial del nivell, i vaig implementar unes funcions que comprovaven si el jugador es podia moure quan premia un botó. Aquestes funcions s'anomenaven "CanMoveRight()", "CanMoveLeft()", "CanMoveUp()" i "CanMoveDown()", i cadascuna comprovava la casella o les caselles corresponents. Per tal d'aconseguir-ho, feien ús de la llista Tiles. Si el jugador es volia moure a la dreta, la funció CanMoveRight() sumava 1 a la coordenada X i comprovava què hi havia a la posició (X + 1, Z) de la llista. Si es volia moure avall, la

funció CanMoveDown() restava 1 a la coordenada Z i comprovava (X, Z – 1) a la llista Tiles.

A més, cada vegada que el jugador intentava moure's i podia moure's, calia actualitzar la llista "Tiles" per a poder fer les comprovacions corresponents als següents torns del jugador, i per a això calia tornar a comprovar les caselles afectades.

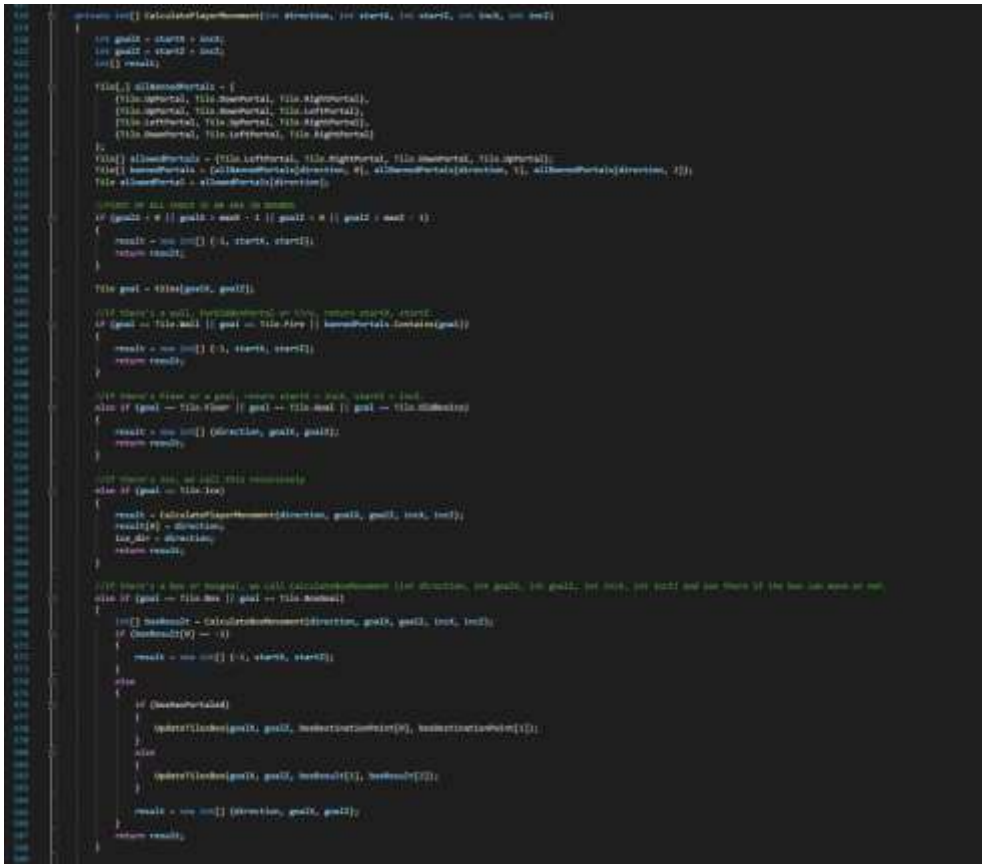
Aquesta implementació tenia dos problemes principals. El primer era que el codi quedava extremadament llarg, amb un seguit de condicionals per a comprovar totes les possibilitats que es podien donar cada vegada que el jugador volia moure's. El segon era que el codi estava repetit, ja que les quatre funcions CanMove esmentades anteriorment feien exactament el mateix: comprovar què hi havia en la direcció que es volia moure el jugador.

A més, la longitud i complexitat del codi escalava de manera exponencial cada vegada que volia afegir una mecànica nova. Per exemple: si hi ha un bloc de foc al terra s'afegeix una nova condició. Si hi ha un portal cal mirar què hi ha més enllà del portal i veure si bloqueja al jugador. Si hi ha una caixa seguida d'un portal també cal mirar més enllà del portal, però les condicions per a bloquejar el jugador són diferents. Si hi ha una caixa damunt d'un bloc de foc el jugador la pot empènyer però torna enrere automàticament perquè el foc el crema, bloquejant-li així la sortida del portal.

Finalment vaig decidir que hi havia d'haver una manera millor d'implementar el moviment del jugador i d'actualitzar la llista tiles, i vaig arribar a la conclusió que un algorisme recursiu faria que el codi anés millor i en reduïria la complexitat en gran mesura. La lògica final del moviment del jugador funciona amb dos mètodes anomenats "CalculatePlayerMovement" i "CalculateBoxMovement", als que passem com a paràmetres la direcció en que es mou el jugador (o la caixa, en el cas de CalculateBoxMovement), les seves coordenades actuals, i com es modificaran les coordenades si es pot moure. Calculem què hi ha a GoalTile (la casella on el jugador es vol desplaçar) i retornarem una llista amb {-1, X actual del jugador, Z actual del jugador} si veiem que a GoalTile hi ha un element que li barra el pas i {direcció del jugador, X objectiu, Z objectiu} si veiem que a GoalTile no hi ha res que li barri el pas. A més, si hi ha una caixa podrem cridar a CalculateBoxMovement amb la informació corresponent i retornar o la primera llista o la segona basant-nos amb la informació que ens arribi.

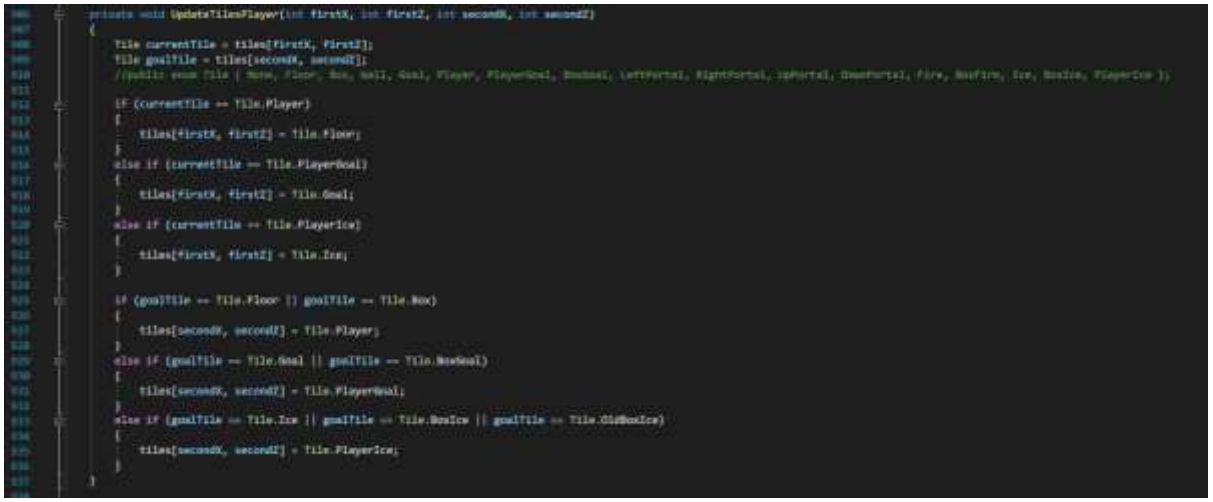
Aquesta implementació no només facilita molt el càlcul del moviment original, sinó que a més ens ajuda a l'hora de crear mecàniques noves. Per exemple, si hi ha un bloc de gel, el jugador relliscarà i seguirà movent-se fins que surti del gel o se'l bloquegi. Gràcies al

funcionament d'aquesta lògcia, podem cridar de manera recursiva a CalculatePlayerMovement cada vegada que hi hagi gel, i la implementació mateixa farà que la funció retorni el bloc final quan el moviment del jugador s'aturi:

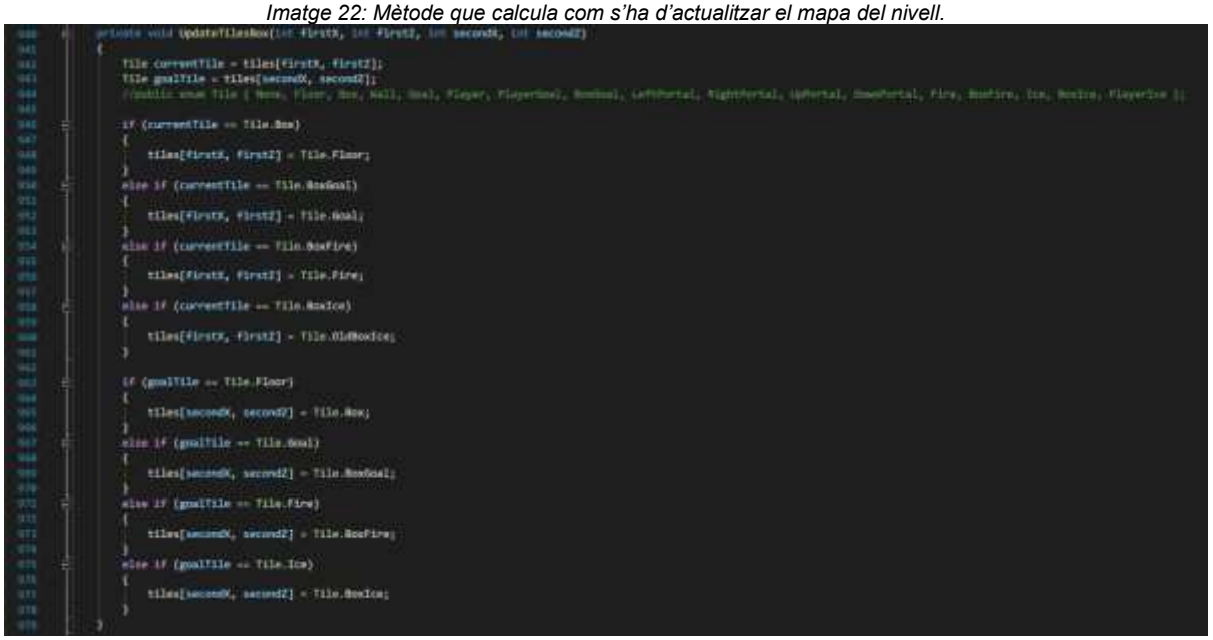


Imatge 21: Fragment del codi que calcula el moviment del jugador.

El mètode CalculteBoxMovement és molt similar, però amb alguna petita variació. Amb la nova implementació, a més, també vaig simplificar el mètode "UpdateTiles()", que va passar a actualitzar només les caselles que hi havia afectades, i a ser dos mètodes separats: UpdateTilesPlayer i UpdateTilesBox:



Imatge 22: Mètode que calcula com s'ha d'actualitzar el mapa del nivell.



Imatge 23: Mètode que calcula com s'ha d'actualitzar el mapa del nivell per a caixes.

El moviment del jugador té altres detalls, sobretot quan es tracta dels portals. La implementació dels portals és la que més dificultat ha tingut amb diferència, sobretot perquè el moviment a través dels portals havia de ser extremadament fluid. Per a això vaig crear una classe CoordPair, que conté les entrades i sortides d'un portal, i quan es

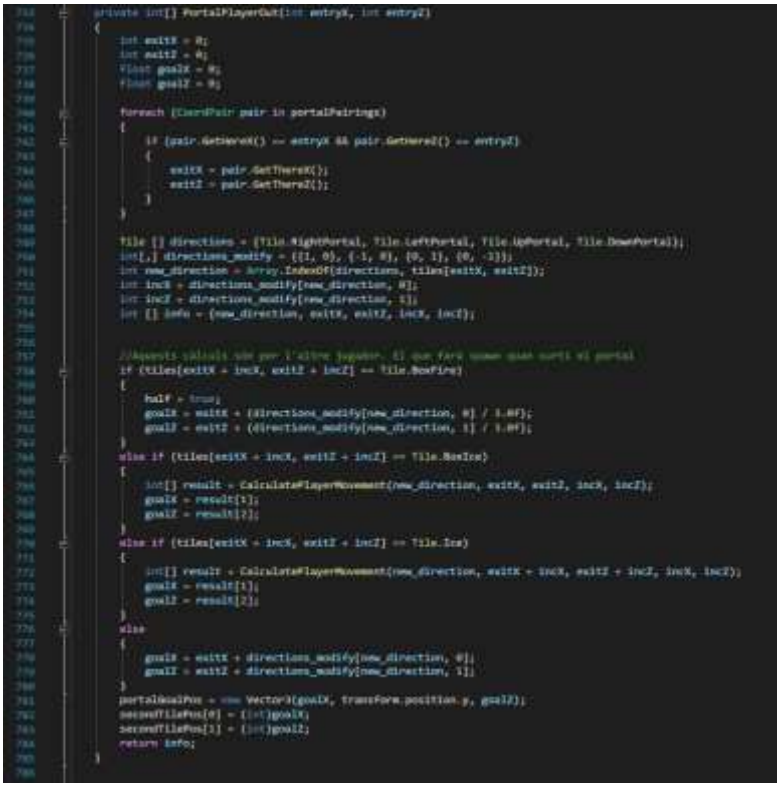
crea el nivell també es fa una llista amb totes les CoordPair del joc. Així, quan el jugador entra a un portal s'eliminen tots els colliders que té, s'instancia un jugador nou a la sortida del portal assignat segons la CoordPair, es passa tota la informació necessària al jugador nou, i quan el jugador original ja ha quedat amagat pel portal al que ha entrat, se'l destrueix. Per a passar tota la informació necessària s'utilitzen els events esmentats a l'apartat anterior:

```
1039    IEnumerator SendInfoPlayer()
1040    {
1041        yield return new WaitForSeconds(0.05f);
1042        onEnterPortal.Raise(this, tiles);
1043        yield return new WaitForSeconds(0.05f);
1044        onEnterPortal2.Raise(this, portalGoalPos);
1045        onEnterPortal3.Raise(this, portalPairings);
1046        onEnterPortal5.Raise(this, half);
1047    }
```

Imatge 24: Mètode que envia la informació del jugador quan passa per un portal.

Aquest mètode "SendInfoPlayer()" s'activa quan el jugador colisiona amb un portal. Primer es desactiven els colliders, i després generem la informació que cal enviar amb l'ajuda de PortalPlayerOut.

Imatge 25: Funció que desactiva els col·lisionadors dels portals quan el jugador hi xoca i calcula per on sortirà.



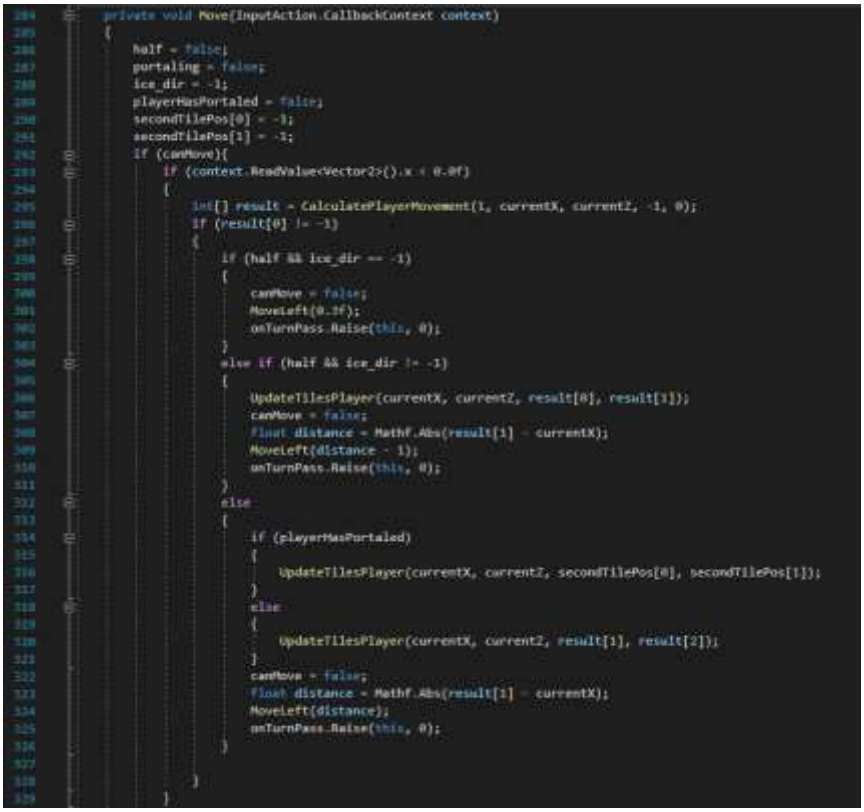
Imatge 26: Funció que calcula tota la informació necessària per passar al jugador que es crea a la sortida del portal.

Quan una caixa passa per un portal es segueix una lògica similar, però la informació que s'envia és una mica diferent, ja que la caixa no necessita tenir control del mapa de caselles ni de les parelles de portals.

Finalment, tota la informació calculada per CalculatePlayerMovement i CalculateBoxMovement es veu reflectida a Move(InputAction.CallbackContext context), que és el mètode que detecta l'input dels controls i crida els mètodes que fan les comprovacions. Si aquests mètodes permeten al jugador moure's, Move(InputAction.CallbackContext context) indica als mètodes MoveLeft, MoveRight, MoveUp i MoveDown si el jugador s'ha de moure i quina quantitat de caselles. Aquest mètode s'ajuda molt de variables auxiliars, que l'ajuden a saber si el jugador s'està movent sobre gel, intentant moure's sobre foc, etc.

També cal esmentar que, mentre el jugador està en moviment, aquesta funció queda bloquejada a través d'un booleà anomenat canMove, per tal d'evitar que el personatge es

quedi entre dues caselles i movent-se en una direcció nova. canMove es desbloqueja de nou quan la funció Update() detecta que el jugador ja ha arribat a la seva posició destí.



```
private void Move(InputAction.CallbackContext context)
{
    half = false;
    portaling = false;
    ice_dir = -1;
    playerHasPortaled = false;
    secondTilePos[0] = -1;
    secondTilePos[1] = -1;
    if (canMove){
        if (context.ReadValue<Vector2>().x < 0.0f)
        {
            int[] result = CalculatePlayerMovement(1, currentX, currentZ, -1, 0);
            if (result[0] != -1)
            {
                if (half && ice_dir == -1)
                {
                    canMove = false;
                    MoveLeft(0.5f);
                    onTurnPass.Raise(this, 0);
                }
                else if (half && ice_dir != -1)
                {
                    UpdateTilesPlayer(currentX, currentZ, result[0], result[1]);
                    canMove = false;
                    float distance = Mathf.Abs(result[1] - currentX);
                    MoveLeft(distance - 1);
                    onTurnPass.Raise(this, 0);
                }
                else
                {
                    if (playerHasPortaled)
                    {
                        UpdateTilesPlayer(currentX, currentZ, secondTilePos[0], secondTilePos[1]);
                    }
                    else
                    {
                        UpdateTilesPlayer(currentX, currentZ, result[1], result[2]);
                    }
                    canMove = false;
                    float distance = Mathf.Abs(result[1] - currentX);
                    MoveLeft(distance);
                    onTurnPass.Raise(this, 0);
                }

            }
        }
```

Imatge 27: Mètode que rep l'entrada del jugador i fa les crides necessàries per a processar el moviment.

La funció Update() s'ajuda també de moltes variables auxiliars, que li indiquen quina és la situació del jugador. Gràcies a aquestes variables pot saber si ha passat per un portal, si acaba de sortir d'un portal, si només s'ha de moure una mica i tornar enrere (passa quan està empenyent una caixa damunt de la lava), si ha passat per gel, etc.

```
void Update()
{
    transform.position = Vector3.MoveTowards(transform.position, goalPos, 2.5f * Time.deltaTime);
    if (transform.position == goalPos)
    {
        boxGoalPos[0] = -1;
        boxGoalPos[1] = -1;
        if (!portaling && !half)
        {
            currentPos = goalPos;
            currentX = (int)currentPos.x;
            currentZ = (int)currentPos.z;
            canMove = true;
        }
        if (ice_dir != -1 && half)
        {
            currentPos = goalPos;
            currentX = (int)currentPos.x;
            currentZ = (int)currentPos.z;
            if (ice_dir == 0)
            {
                canMove = false;
                MoveRight(0.3f);
            }
            else if (ice_dir == 1)
            {
                canMove = false;
                MoveLeft(0.3f);
            }
            else if (ice_dir == 2)
            {
                canMove = false;
                MoveUp(0.3f);
            }
            else if (ice_dir == 3)
            {
                canMove = false;
                MoveDown(0.3f);
            }
            ice_dir = -1;
        }
        else if (portaling && !half)
        {
            Destroy(gameObject);
        }
        else if (!portaling && half)
        {
            canMove = true;
            goalPos = new Vector3 (currentX, 1.0f, currentZ);
```

Imatge 28: A Update calculem si el jugador ha arribat a la posició final, i s'utilitzen variables auxiliars per saber què hi ha de fer un cop arriba.

5.4. Les mecàniques noves

Com ja he esmentat abans, la idea d'aquest projecte era expandir el joc base mitjançant la creació d'algunes mecàniques noves que donessin més profunditat al seu gameplay. La meva idea era la d'implementar més mecàniques que les que han resultat al final, però la implementació dels portals va ser més llarga del que pensava. La interacció general de les mecàniques noves amb el jugador és l'explicada a l'apartat 5.3, però considero important esmentar-ho aquí per separat. Al final, el joc consta de:

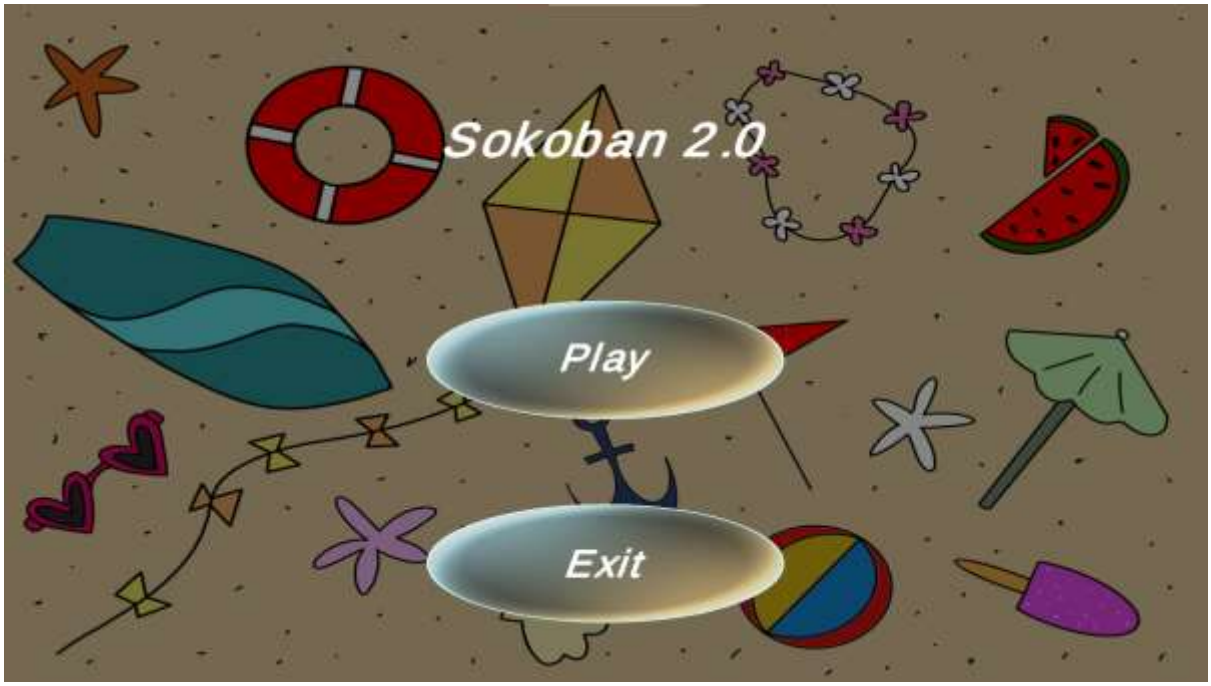
- **Blocs de foc:** Si CalculateBoxMovement() veu que la caixa s'està movent en direcció d'un bloc de foc, la deixa passar, ja que la caixa pot passar per damunt del foc. Si CalculatePlayerMovement() veu que el jugador està intentant dirigir-se cap a un bloc de foc, retorna un {-1, X actual, Z actual}, indicant-li que no pot moure's.
- **Blocs de gel:** Si el jugador camina per damunt del gel relliscarà fins a trobar un bloc que l'aturi (ja sigui una paret, terra sense gel, un bloc de foc) o una caixa (en aquest cas ocuparà la posició de la caixa i li passarà la inèrcia a aquesta). Les caixes també rellisquen per damunt del gel.
- **Portals:** Quan un jugador ha d'entrar un portal, es comprova què hi ha a l'altra banda del portal com si no hi hagués els portals entre mig. És a dir, s'ignora la presència del portal fins a cert punt i només mira si podrà desplaçar-se quan surti. Si no hi ha res bloquejant-lo, el jugador entra al portal, on serà destruït. Simultàniament, s'instancia un jugador nou a l'altre portal.
- **Doble jugador:** És una mecànica de la que no he parlat gaire, però perquè no vaig tenir temps d'implementar-la fins a última hora, i conseqüentment només hi ha un nivell que la provi de manera molt bàsica, però és una mecànica que es pot explotar per a donar lloc a situacions molt interessants. Aquesta mecànica consisteix en l'existència de dos jugadors dins del nivell. Tots dos es mouen alhora però han de seguir camins diferents, així que cal trobar la manera que ambdós aconsegueixin empènyer les caixes fins les metes pel seu compte, però movent-se sempre sincronitzats. Si hagués tingut més temps hagués explotat aquesta mecànica per veure si funcionava amb la resta.

Aquestes mecàniques no només funcionen de manera individual, sinó que també funcionen quan es combinen les unes amb les altres, donant lloc a moltes possibilitats.

5.5. Les interfícies gràfiques

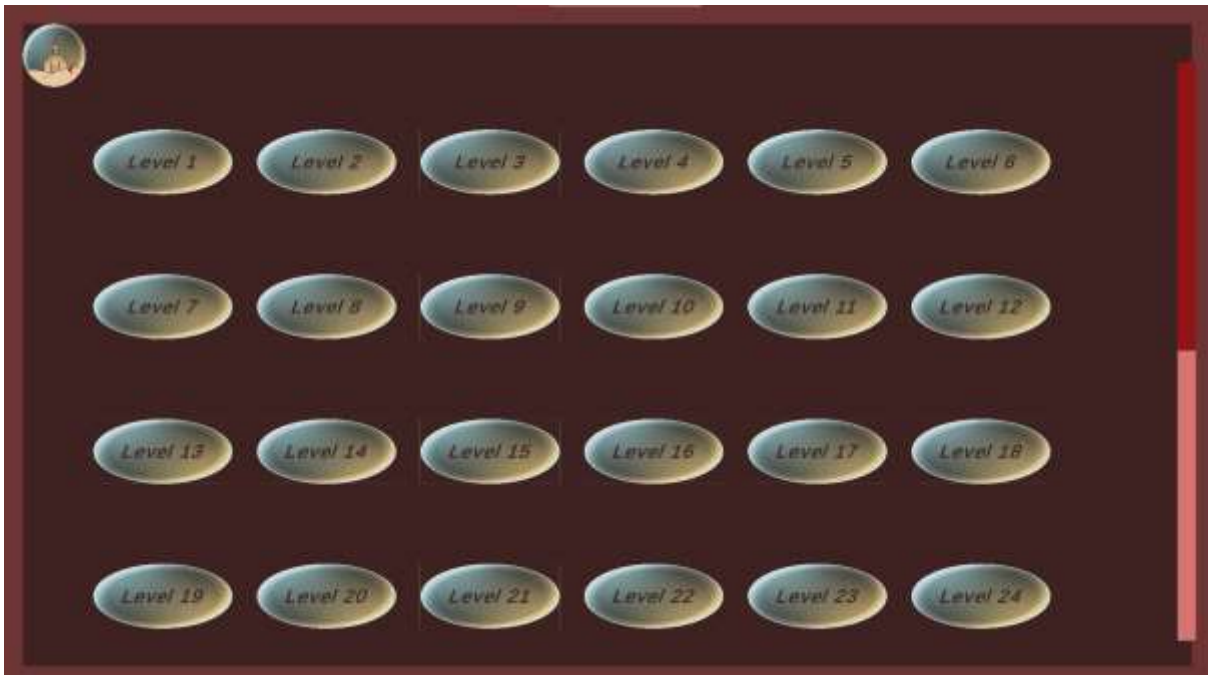
Tot i que les interfícies gràfiques són trivials de programar, he volgut esmentar-les perquè penso que són importants. Així doncs, tenim 4 escenes, 3 de les quals són menús d'alguna mena:

- **Menú principal:** Aquest menú conté el títol i dos botons, un per jugar i un per abandonar la partida. També té un fons que segueix l'estil de l'estètica de platja escollida:



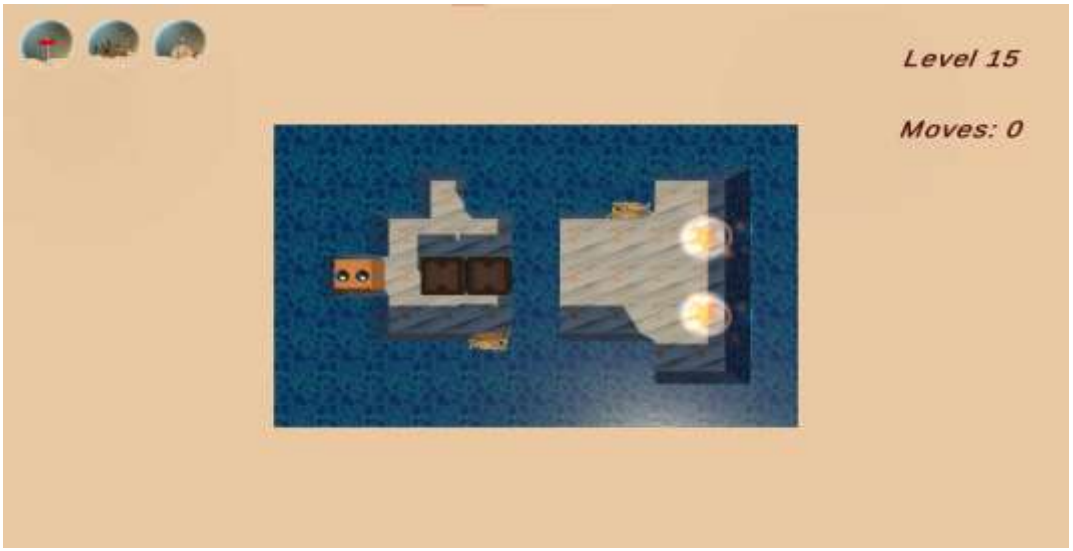
Imatge 29: Escena del menú principal

- **Selecció de nivell:** Aquesta escena té 33 botons, un per cada nivell del joc, distribuïts en 6 files i columnes de manera proporcional, utilitzant els constraints i anchors que vénen amb els elements que formen part del Canvas. També té una scrollbar i un botó que permet tornar al menú principal.



Imatge 30: Escena de selecció de nivell

- **Nivell:** La interfície del nivell consisteix de 3 botons que permeten activar i desactivar la música, reiniciar el nivell i tornar al menú principal. A més, hi ha un text a la part dreta del canvas que indica a quin nivell es troba el jugador i quants moviments ha realitzat.



Imatge 31:Exemple de nivell

- **Pantalla de victòria:** Aquesta pantalla felicita al jugador per haver batut el nivell que estava jugant i el presenta amb 3 opcions: jugar el següent nivell, anar a la pantalla de selecció de nivell per escollir quin vol jugar, i sortir de l'aplicació.

Imatge 32: Pantalla de victòria

5.6. Altres aspectes

Altres aspectes a considerar del treball són:

- **Banda sonora:** mitjançant l'ús del programa "Bosca Ceoil", s'ha creat una petita banda sonora intentant imitar el so d'una màquina arcade. S'han creat dues cançons; una per al menú principal i una altra pels nivells. Sonen a través d'un AudioSource. El volum del nivell es pot desactivar mitjançant el botó de sound on/off.

Imatge 33: Pantalla inicial de Bosca Ceoil.

Imatge 34: Vista de la font d'àudio a l'inspector de Unity.

- **Texturització:** Per a realitzar els texture maps dels blocs s'ha utilitzat Adobe Substance, tant el Painter com el Designer. Un cop creats els texture maps, només ha calgut importar-los dins de Unity i crear un material per a cada tipus de bloc. Dins del material he anat assignant cada mapa a la part que li corresponia, per finalment posar cada material al bloc pertinent.

Imatge 35: Vista d'un material a l'inspector de Unity.

- **Sistema de partícules i il·luminació:** El prefab de "Goal" té un sistema de partícules que emet petites esferes de color taronja i un torus pla, també de color taronja, cap amunt. També té un focus de llum propi que l'enfoca des del cel directament a ell. Això serveix per fer-ho més visible per al jugador.

Imatge 36: Vista del sistema de partícules des de l'inspector de Unity.

Imatge 37: Vista de la llum de les metes des de l'inspector de Unity.

6. Demostració

6.1 Instruccions d'ús

Per a l'execució de l'aplicació només és necessari accedir al següent repositori de github: https://github.com/MSBeatles/TFM_SOKOBAN/tree/main

i descarregar-lo. Un cop descarregat, es pot descomprimir o bé el fitxer .zip o bé el .7z (són diferents comprimits de la mateixa build). Un cop estigui la carpeta descomprimida, només cal prémer doble click damunt del fitxer executable (el .exe) i començarà el joc.

6.2 Tests

Tot i que no he realitzat tests unitaris a l'hora de desenvolupar el projecte, sí que m'ha anat molt bé aprendre a debugar amb Unity. No sabia que el programa permetia fer-ho (tot i que assumia que alguna manera hi hauria) i la veritat és que, quan funciona, és una manera molt útil de trobar errors. A la recta final, quan les persones que van provar el meu joc voluntàriament per buscar-hi errors, m'informaven d'alguna cosa, no corregia res sense debugar. És una eina que s'hauria de normalitzar a l'hora de desenvolupar perquè ha resultat un autèntic salva vides.

6.3 Exemples d'ús del producte (o guia d'usuari)

Tot i que el funcionament del joc és molt senzill, incloc aquí una petita guia d'usuari:

- Prémer tecles AWSD o de direcció per moure's. També es pot utilitzar un comandament.
- Dels blocs vistos a la secció 5.1, l'usuari és el jugador. L'objectiu del jugador és empènyer caixes (bloc Box) perquè arribin a la meta final (bloc Goal). El nivell es passa quan totes les metes tenen una caixa damunt.
- Les parets (bloc Wall) no permeten el pas al jugador.
- El jugador només pot empènyer una caixa de cop, si n'hi ha dues actuaran com una paret.
- El foc (bloc Fire) no permet el pas del jugador, però si de les caixes.
- El gel (bloc Ice) fa que rellisquin les caixes que passin per damunt. El jugador també rellisca, fins que un bloc l'obliga a aturar-se.

- Si el jugador entra per un portal (bloc RightPortal, LeftPortal, UpPortal o DownPortal), sortirà pel portal que estigui aparellat a aquest. El jugador pot empènyer caixes a través de portals. No tots els portals són bidireccionals; no està garantit que entrar al portal pel qual el jugador acaba de sortir el porti on era abans.

7 Conclusions i línies de futur

7.2 Conclusions

Les conclusion que puc treure després de fer aquest treball són les següents:

- Desenvolupar un joc des de zero porta molta feina i requereix molta organització i planificació, i és important mantenir horaris i eines que ajudin a portar un registre de la feina feta i per fer, així com del calendari de treball.
- La complexitat del codi pot créixer de manera exponencial si no s'implementa de manera adequada d'un bon principi. És important començar els projectes bé i tenir una base sòlida abans de créixer, ja que si es comencen a desenvolupar funcionalitats noves amb una base dèbil els errors s'arrosseguen i costen molt més d'arreglar.
- Val més tenir un projecte petit però concís i ben fet que intentar arribar a tenir un projecte molt ambiciós i quedar-se curt. Cal ser realista i no passar-se amb els objectius, ja que pot portar a una quantitat de treball desmesurada que acabi en fatiga i frustració, cosa que porta a una baixada important del rendiment. Aquest punt va lligat al primer.
- Aquestes tres conclusions em duen a reflexionar sobre els objectius que em vaig proposar al principi del treball. Els resultats aconseguits no són de la qualitat que jo volia i que havia esperat, però això va de la mà amb l'optimisme a l'hora de planejar i l'ambició del projecte. Després de treballar-hi, m'he adonat que intentar tenir un producte preparat per sortir a Steam i Google Play en quatre mesos no era realista, i menys encara quan estava sent desenvolupat només per una persona (jo) que no es dedica a l'estudi del màster a jornada completa i que a més no té experiència prèvia desenvolupant un videojoc des de 0. Considero, però, que amb més temps podria aconseguir els objectius que em vaig plantejar al principi del projecte, i prefereixo centrar-me en la part positiva, que és tot el que he après durant aquest semestre.

Aquesta experiència m'ha ajudat a complir un dels meus objectius personals per al projecte, així que no considero que hagi estat un fracàs absolut.

7.3 Línies de futur

Pel que fa a les línies de futur, m'agradaria (amb temps i paciència) fer créixer aquest joc. Tinc la sensació que les mecàniques implementades per a aquest treball final només són la superfície de tot el que es podria fer partint de la base del Sokoban, i penso que amb uns quants mesos més de desenvolupament es podria crear un joc de puzzles realment complex i interessant, amb interaccions i trencaclosques que duessin al jugador al límit. També m'agradaria treballar en l'aspecte gràfic i sonor del joc, crear diverses temàtiques que el jugador pogués desbloquejar i aprendre a fer servir un programa d'edició de so professional per a poder fer bandes sonores que sonessin més modernes. Ja com a detall m'agradaria, d'alguna manera, lligar l'estètica visual del joc amb la seva música.

Bibliografia

Elite Home Theater Seating. (n.d.). The History of the Home Theater. Retrieved April 16, 2023, from https://elitehts.com/the-history-of-the-home-theater/

Game Preservation Society. (n.d.). Retrieved April 16, 2023, from https://www.gamepres.org/en/

U.S. Government Publishing Office. (2014, March 10). The History of eBooks from 1930's «Readies» to Today's GPO eBook Services. Retrieved April 16, 2023, from https://govbooktalk.gpo.gov/2014/03/10/the-history-of-ebooks-from-1930s-readies-to-todays-gpo-ebook-services/

History. (n.d.). Video Game History. Retrieved April 16, 2023, from https://www.history.com/topics/inventions/history-of-video-games

Internet Archive. (n.d.). Retrieved April 16, 2023, from https://archive.org

Metacritic. (n.d.). Game Releases by User Score (2023). Retrieved April 16, 2023, from https://www.metacritic.com/browse/games/score/metascore/year/all/filtered

Naughty Dog. (n.d.). The Last Of Us (Looking ahead to a milestone year). Retrieved April 16, 2023, from https://www.naughtydog.com/blog/the_last_of_us_10th_anniversary_kickoff?sf174194040=1

Newzoo. (n.d.). The Games Market and Beyond in 2021: The Year in Numbers. Retrieved April 16, 2023, from https://newzoo.com/resources/blog/the-games-market-in-2021-the-year-in-numbers-esports-cloud-gaming

Newzoo. (n.d.). Top Countries/Markets by Game Revenues. Retrieved April 16, 2023, from https://newzoo.com/resources/rankings/top-10-countries-by-game-revenues

The NPD Group. (n.d.). Top 10 Selling Video Games (Retail and Digital), February 2023. Retrieved April 16, 2023, from https://www.npd.com/news/entertainment-top-10/2023/top-10-video-games/

PBS NewsHour. (n.d.). A short history of the audiobook, 20 years after the first portable digital audio device. Retrieved April 16, 2023, from https://www.pbs.org/newshour/arts/a-short-history-of-the-audiobook-20-years-after-the-first-portable-digital-audio-device

National Science and Media Museum. (n.d.). A very short history of cinema. Retrieved April 16, 2023, from https://www.scienceandmediamuseum.org.uk/objects-and-stories/very-short-history-of-cinema

Video Game History Foundation. (n.d.). Retrieved April 16, 2023, from https://gamehistory.org/

Wikipedia. (n.d.). Home video game console. In Wikipedia. Retrieved April 16, 2023, from https://en.wikipedia.org/wiki/Home_video_game_console

Wikipedia. (n.d.). Retrogaming. In Wikipedia. Retrieved April 16, 2023, from https://en.wikipedia.org/wiki/Retrogaming

Wikipedia. (n.d.). Sokoban. In Wikipedia. Retrieved April 16, 2023, from https://en.wikipedia.org/wiki/Sokoban

Wikipedia. (n.d.). Video Game Preservation. In Wikipedia. Retrieved April 16, 2023, from https://en.wikipedia.org/wiki/Video_game_preservation

Annexos

Llistat d'apartats complementaris addicionals o que són massa extensos per incloure dins de la memòria i tenen un caràcter auto-contingut. Depenent del tipus de treball, és possible que no calgui afegir cap annex.

Annex A: Lliurables del projecte

El projecte entregat consisteix en un informe en format pdf. També s'ha entregat la presentació PowerPoint utiltizada en la defensa. A més, als enllaços següents hi ha la resta d'informació:

- **Repositori GitHub:** https://github.com/MSBeatles/TFM_SOKOBAN
  - Conté una carpeta anomenada "Sokoban" amb el projecte de Unity, i dues carpetes comprimides anomenades "Executable.7z" i "Executable.zip", que contenen l'executable del joc. Qualsevol de les dues serveix. Finalment conté un README.md on hi ha les instruccions per a executar el joc.
- **Tràiler del joc:** https://youtu.be/z3GmAneha-Q
- **Presentació del treball:** https://youtu.be/rAXvwbBpYG0