TFG VIDEOJOCS – SAVE!

Sergio Blasco Ruiz
Grau d'Enginyeria Informàtica

Joel Servitja Feu

10 Juny de 2018


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

FITXA DEL TREBALL FINAL

Títol del treball:	*SAVE!*
Nom de l'autor:	*Sergio Blasco Ruiz*
Nom del consultor:	*Joel Servitja Feu*
Data de lliurament (mm/aaaa):	*06/2018*
Àrea del Treball Final:	*Videojocs*
Titulació:	*Grau d'Enginyeria Informàtica*

Resum del Treball (màxim 250 paraules):

S'ha iniciat aquest treball amb la intenció de guanyar experiència en el sector del desenvolupament de videojocs. L'objectiu era veure tot el procés de creació d'un videojoc des de 0 fins a aconseguir un producte acabat amb un cert grau de qualitat.

S'ha desenvolupat un joc Arcade 2D per plataformes Android mitjançant un motor de creació potent com és Unity, així com altres eines de suport tant per guanyar coneixement (tutorials, videos, etc..) com per afegir elements extres, com poden ser textures i Sprites mitjançant Photoshop.

En aquest document es pot trobar tota la informació sobre el procés seguit així com la informació necessària sobre com està fet i perquè.

Abstract (in English, 250 words or less):

The main objective to start this project has been to get experience in the game development sector. For that, it was important to see all the process, starting from zero and achieving a complete playable version within a certain quality degree.

The game developed is a 2D Arcade for Android platforms. Unity has been the chosen engine due to its powerfulness and easiness of use. Other tools have been used as well to support Unity, such as Photoshop and tutorials/videos.

All the information about the development process and the decisions made can be found in this document.

Paraules clau (entre 4 i 8):

Videojoc, Repte, Android, Passió, Arcade, 2D, TFG

Índex

1. Introducció..... 3
   - 1.1 Context i justificació del treball..... 3
   - 1.2 Objectius del Treball..... 4
   - 1.3 Enfocament i mètode seguit..... 4
   - 1.4 Planificació del Treball..... 4
   - 1.5 Breu sumari de productes obtinguts..... 7
   - 1.6 Breu descripció dels altres capítols de la memòria..... 7
2. Disseny..... 8
   - 2.1 Tipus d'interacció joc-jugador..... 8
   - 2.2 Plataforma destí..... 8
   - 2.3 Conceptualització..... 9
   - 2.4 Avaluació d'engines i kits de desenvolupament..... 13
3. Implementació..... 15
   - 3.1 El jugador..... 15
   - 3.2 PNJ..... 18
   - 3.3 Foc..... 19
   - 3.4 Nivells pistola..... 20
   - 3.5 Aigua..... 22
   - 3.6 Moneda..... 23
   - 3.7 Entorn..... 24
   - 3.8 Puntuació..... 28
   - 3.9 Temps..... 28
   - 3.10 Controlador de joc..... 29
   - 3.11 Controlador de menús..... 34
   - 3.12 Càmera..... 35
   - 3.13 So..... 35
   - 3.14 Persistència..... 37
   - 3.15 Tutorial..... 39

4. Interfícies..... 40
4.1 Menús..... 40
4.2 UI durant la partida..... 46
4.3 Gràfics..... 47
4.4 Animacions..... 48
5. Conclusions..... 52
6. Glossari..... 54
7. Webgrafia..... 55
8. Annexos..... 56

1.Introducció

1.1 Context i justificació del Treball

Des de ben petit, com qualsevol altre nen, sempre m'han apassionat els videojocs. Podia passar hores i hores jugant sense avorrir-me i sense adonar-me'n del temps que hi portava. Amb el pas dels anys, aquesta passió va anar transformant-se mica en mica fins el punt de no només jugar, sino de formar part d'aquest apassionant.

D'entre tots els tipus de joc que existeixen avui en dia, el gènere que més m'ha agradat sempre és el de jocs de rol multijugador massius en línia (MMORPG). Anys enrere, després de jugar a qualsevol d'aquests jocs durant un temps, el meu costat més crític em feia voler trobar i pensar en característiques que el podrien arribar a millorar. D'aquesta manera, va arribar el dia que em vaig fer la gran pregunta: Si no hi ha cap joc que trobi perfecte, si sempre trobo a faltar alguna característica interessant, si no acabo de trobar EL joc... perquè no fer-lo jo?

Així, la meva passió va donar-me la resposta a què em volia dedicar. Des de llavors, vaig decidir estudiar un Grau en Enginyeria Informàtica per tenir una base sobre la que recolzar-me a l'hora d'aprendre a fer videojocs i, sincerament, quan vaig veure que hi havia la opció de fer el TFG sobre videojocs, no m'ho vaig pensar dos cops. Soc conscient que em queda un llarg camí per recórrer, però estic desitjant veure quin serà el resultat.

Sobre el joc desenvolupat, com que és el primer joc que desenvolupo i no tinc experiència prèvia, he decidit desenvolupar un joc en 2D per plataformes mòbils de gènere Arcade. Crec que els jocs per aquests tipus de plataformes tenen molt futur i cada cop n'apareixen més. A més, permet arribar a un públic més ampli que amb consoles i inclús PC, per tant, pot arribar a un major número de persones.

Amb aquest joc espero estrenar-me com a desenvolupador en aquest món, amb l'ajuda d'un IDE molt potent com és Unity. Espero obtenir un resultat del que sentir-me orgullós, així com graduar-me com a enginyer informàtic per poder continuar la meva formació i obrir-me la porta a treballar en altres projectes amb l'experiència obtinguda.

1.2 Objectius del Treball

Hi ha varis objectius en aquest treball, tant acadèmics/professionals com personals:

- Aprendre tot el que pugui i obtenir una experiència valuosa de cara al futur.
- Resoldre els problemes que puguin sorgir provant diferents aproximacions, que també ajudarà a guanyar experiència.
- Veure les fases de desenvolupament per les que passa un videojoc, des de la idea inicial, fins al resultat final.
- Conèixer i experimentar tot el procés de planificació, veient les parts més complicades de predir i les més previsibles.
- Acabar el grau universitari satisfactòriament.
- Crear el meu primer videojoc.
- Aprendre Unity, una eina molt potent per donar vida a les nostres idees en forma de videojoc.

1.3 Enfocament i mètode seguit

Un projecte com aquest es pot enfocar de diferents maneres, es pot crear un joc basant en altres o intentar buscar un sistema de videojoc totalment innovador. No obstant, considero que no és assequible fer un videojoc innovador com a primer projecte sense una idea clara de què es vol fer i sense experiència.

Per poder assolir els objectius que demana un treball d'aquesta envergadura en el termini de temps establert, no es pot escollir un projecte de gaire dificultat, ja que s'ha de tenir en compte que es desenvolupa individualment. Per tant, un joc relativament senzill, atractiu, però que compleixi amb els requisits de mida i dificultat és el més adient.

Per tant, per al treball he decidit crear un videojoc 2D basat en una vista **top-down** per plataformes mòbils, amb una jugabilitat senzilla i divertida per crear el màxim d'entreteniment possible mentre es presenta un repte mental al jugador per esbrinar com passar els nivells. Amb aquests elements en ment va néixer SAVE!.

1.4 Planificació del Treball

Per realitzar el projecte s'han dividit les feines en 4 PACs que s'han anat entregant durant el semestre. La divisió de feines és la següent:

- PAC1, disseny:

  - ✓ Obtenir la idea del joc
  - ✓ Conceptualització
  - ✓ Planificació

- PAC2, una versió parcial:

  - ✓ Aprenentatge Unity: aprendre els conceptes bàsics del engine per poder començar a treballar en el desenvolupament del joc
  - ✓ Escenari basic: creació d'un escenari bàsic per provar el joc
  - ✓ Creació jugador: creació d'un objecte que actuarà de jugador
  - ✓ Creació PNJ: creació d'un objecte que actuarà de persona a salvar
  - ✓ Creació foc: creació d'objectes a posicions aleatòries que actuaran de foc
  - ✓ Sistema moviment: programació del sistema de moviment del jugador
  - ✓ Sistema pistola: programació del sistema de disparar per apagar el foc
  - ✓ Sistema de vida del foc: programació del sistema de vida del foc per saber quan s'extingeix
  - ✓ Menú de joc: creació d'un menú bàsic per poder iniciar partida

- PAC3, una versió jugable:

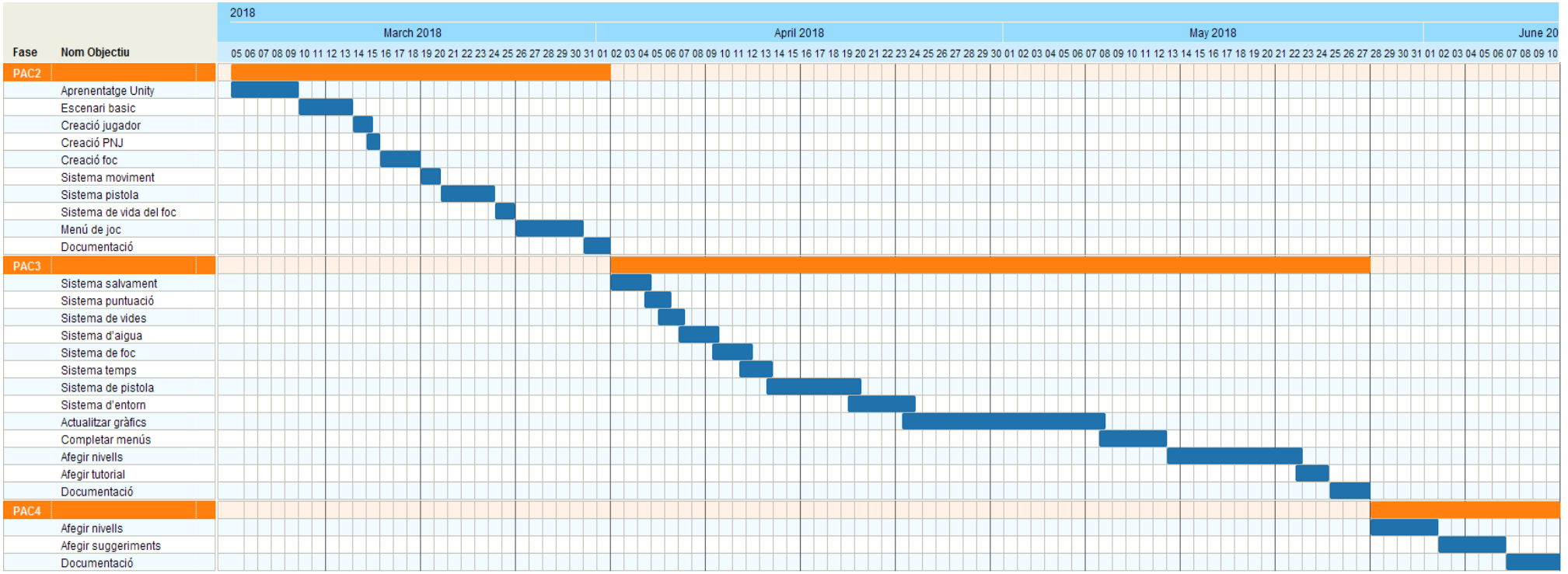
  - ✓ Sistema salvament: programar que els PNJ segueixin al jugador un cop s'interactua amb ells
  - ✓ Sistema puntuació: programar el sistema de puntuació del joc
  - ✓ Sistema de vides: programar el sistema de vides del jugador
  - ✓ Sistema d'aigua: programar el control de volum d'aigua restant
  - ✓ Sistema de foc: programar que si un foc porta massa estona sense apagar-se, no es pugui tornar a caminar per aquella zona
  - ✓ Sistema temps: programar el sistema de temps per completar cada nivell
  - ✓ Sistema de pistola: programar el sistema de millora de pistola
  - ✓ Sistema d'entorn: programar que es puguin fer accions especifiques sobre parts del nivell per moure elements d'aquest i fer accessibles algunes zones
  - ✓ Actualitzar gràfics: canviar els elements gràfics provisionals per els elements correctes/finals
  - ✓ Completar menús: implementar els menús complets del joc.
  - ✓ Afegir nivells: implementar/crear diferents nivells (número a concretar segons el temps del que es disposi en aquest punt)
  - ✓ Afegir tutorial: incloure una pantalla informant dels controls del joc

- PAC4, una versió final:

  - ✓ Afegir nivells: implementar/crear més nivells
  - ✓ Afegir suggeriments: afegir/implementar possibles millores i suggeriments per part del consultor
  - ✓ Memòria final i video de defensa del projecte

Quantificació de temps i recursos per objectiu

Per fer el gràfic es comptarà que es disposa de 98 dies per desenvolupar el projecte (5/03/18 – 10/06/18).



1.5 Breu sumari de productes obtinguts

Es tracta d'un joc 2D en el que el jugador es posa a la pell d'un bomber que intentarà salvar les persones atrapades a un gratacels en flames. Armat només amb una pistola d'aigua, haurà d'obrir-se camí entre les flames de cada pis fins on es troben les persones per ajudar-les a tornar a l'entrada i salvar-les. Un cop arriba a l'últim pis, el jugador haurà completat el joc i totes les persones hauran estat salvades.

El joc consta de 5 nivells de dificultat creixent on el jugador haurà d'utilitzar les millores que va desbloquejant a mesura que avança amb les monedes que pot recollir. A més, alguns nivells disposen de mecàniques que fan que sigui impossible passar-los sense haver desbloquejat algunes millores de la pistola.

Quan el jugador hagi completat els 5 nivells, haurà completat el joc. Esmentar que tot i que actualment no hi ha més nivells implementats, el joc està preparat per suportar més nivells en cas d'implementar-los.

1.6 Breu descripció dels altres capítols de la memòria

Els següents capítols de la memòria intentaran plasmar el millor possible el procés seguit per desenvolupar el joc.

Per una banda hi ha l'apartat de disseny, que engloba tot el treball d'anàlisi i disseny que es va desenvolupar per la PAC1 afegint totes les funcionalitats i canvis que s'han efectuat al llarg del projecte degut a problemes, canvis d'idea i els suggeriments aportats per el consultor i amics i familiars.

Seguidament hi ha l'apartat d'implementació. Aquest apartat intenta explicar com s'ha implementat el joc, detallant objecte per objecte i **script** per **Script** el millor possible per tenir una idea general de com s'ha creat aquest joc i del perquè s'ha fet així. A més, es van plantejant alternatives que s'han provat o pensat i millores que es poden aplicar en un desenvolupament futur.

Per últim, abans de les conclusions hi ha un altre apartat enfocat a mostrar tota la part gràfica del joc, així com les animacions i el disseny de menús. D'aquesta manera també es té una visió de com s'ha desenvolupat la part gràfica i no només la implementació interna.

2. Disseny

2.1 Tipus d'interacció joc-jugador

El jugador pot interaccionar amb el joc de manera totalment tàctil mitjançant els menús de navegació i la interfície d'usuari que es presenta durant la partida.

Mentre està al menú principal, el jugador por començar a jugar quan clica al botó "PLAY" i selecciona el nivell al que vol jugar. A més de poder començar partida, el jugador també pot accedir a un menú d'opcions des del que pot controlar tant el volum de la musica i dels efectes de so (per separat) com silenciar qualsevol so que provingui del joc.

El menú principal disposa també d'un botó per permetre al jugador entrar a la botiga, on pot desbloquejar millores de la pistola d'aigua amb les monedes que va recollint per els nivells. Cada millora depèn de l'anterior, és a dir, el jugador no pot desbloquejar una millora si no ha desbloquejat primer l'anterior, i les característiques que ofereix cada millora s'uneixen a la pistola d'aigua per fer-la cada cop més potent. Per últim, el menú principal també ofereix al jugador un botó per tancar el joc.

Un cop s'inicia un nivell, el jugador pot moure`s per la pantalla en qualsevol direcció, a més de disparar la pistola d'aigua per apagar els focs, accionar el botó de salvament quan s'acosti a un personatge no jugador (PNJ) o parar la partida per tornar al menú principal o reiniciar el nivell. A més, durant la partida el joc mostrarà al jugador informació sobre quanta aigua li queda a la pistola, quantes vida li queda, quant temps li queda per superar el nivell i quina és la puntuació que porta.

Per finalitzar, al completar cada nivell es mostrarà al jugador la puntuació que ha obtingut, informant de si ha superat el nivell amb un nou rècord de puntuació i amb l'opció per continuar jugant amb el següent nivell, a més de les opcions per tornar a intentar el nivell en cas de voler obtenir més puntuació i l'opció per tornar al menú principal.

2.2 Plataforma destí

Inicialment la intenció era fer un joc per desktop, ja que és una de les plataformes amb més usuaris. No obstant, en els darrers temps, els tipus de joc com el que es pretén desenvolupar tenen molt més èxit en plataformes mòbils, ja que acostumen a ser jocs per jugar de manera casual quan es té un moment lliure i no se sap què fer. Per tant, és adient que el joc estigui destinat a aquest tipus de dispositius, concretament dispositius Android.

2.3 Conceptualització

El joc es centra en un bomber destinat a apagar l'incendi que s'ha originat misteriosament a un gratacels. Abans, però, haurà de rescatar a totes les persones atrapades a l'interior. D'aquesta manera el joc es basarà en salvar persones a cada pis fins que no en quedi cap.

2.3.1 Definició dels personatges / elements

Per una banda tenim el personatge principal del joc, el bomber, caracteritzat amb la típica vestimenta vermella dels bombers, un casc amb màscara de gas i una motxilla de bombones d'aigua a l'esquena. A més, el bomber també disposa d'una pistola d'aigua per poder apagar els focs que es trobi als nivells.

Un altre element que trobem, tot i que va unit al bomber, és la pistola d'aigua. S'encarrega de generar l'aigua que es dispara i disposa de certes característiques segons les millores desbloquejades a la botiga. Com a element associat, hi ha l'aigua, que avançarà uns metres abans de desaparèixer i apagarà el foc que es trobi per el camí.

Per una altra banda tenim els PNJ que són les persones a qui el bomber ha de salvar. Generalment, estan atrapats a llocs inaccessibles inicialment, però que el jugador podrà arribar al seu costat fent certes acciones que s'explicaran més endavant.

Un altre element que mereix una menció especial és el foc. El foc és un element bàsic del joc, ja que és qui s'interposa entre el jugador i les persones a qui ha de salvar. Segons la temperatura que agafi, el foc pot tenir 2 colors, que serveix d'indicador per saber si es poden apagar o no, segons les millores de la pistola. A més, el foc també disposa de vida que, al acabar-se, provoca que s'apagui el foc.

L'últim element important que cal destacar amb més detall és la botiga del joc. Aquesta part és essencial per donar la sensació de progressió que cal en un joc, per què el jugador senti que està progressant mentre desbloqueja millores i passa els nivells.

Com a elements més petits però no per això menys importants hi ha les monedes que s'aconsegueixen a cada nivell, els obstacles del nivell per crear zones de difícil accés junt amb el foc, tota la interfície gràfica de la partida per poder dur a terme les accions, la barra de capacitat d'aigua que indica quanta aigua queda a les bombones i tots els menús del joc.

2.3.2 Interacció entre els actors del joc

Per part del bomber, interacciona amb el foc, l'entorn, les monedes i els personatges a rescatar. Per una banda pot apagar el foc amb la pistola d'aigua, no obstant, si s'hi apropa massa, rep mal i baixa la seva vida. Quan la vida del bomber arriba a zero, es finalitza la partida.

Per l'altra banda, pot interactuar amb l'entorn a través de l'aigua per descobrir accions secretes que, al fer-les, desbloquegin l'accés a noves zones del nivell on assolir altres objectius. Actualment només hi ha una acció secreta disponible, però en un futur no es descarta afegir-ne més.

Seguidament, al apropar-se a un PNJ pot interaccionar amb ell i fer que el segueixi per tot el nivell per tal d'ajudar-lo a arribar a la sortida i salvar-lo. Cal destacar que si durant el salvament, el PNJ s'allunya massa del bomber, aquest es para i no torna a caminar fins que el bomber no s'apropa.

Com ja s'ha vist al punt anterior, el bomber porta una pistola. Aquest fet és simbòlic ja que no hi ha cap element gràfic dedicat exclusivament a mostrar-la, sino que forma part del model del bomber. La interacció de la pistola amb el joc es basa en carregar les característiques de les millores que hi ha desbloquejades al joc en aquell moment i generar l'aigua quan el jugador vulgui disparar.

Al fer clic sobre el botó de disparar, apareix en escena l'aigua. L'aigua recorre una distància fixada per les característiques de la pistola i quan entra en contacte amb un foc, li baixa la vida (seguidament s'aprofundirà en aquest aspecte).

Pel que fa al foc, apareix cada cert temps i aleatòriament a qualsevol zona del nivell, sempre al terra. No pot aparèixer mai a sobre del jugador ni dels PNJ i tampoc de la decoració, però si que és possible que aparegui just a la posició on es vol moure el jugador en aquell moment. Al aparèixer el foc sempre és de tipus normal, i quan passa una certa quantitat de temps sense haver-se apagat, es torna foc blau, fent que no es pugui apagar si no s'ha desbloquejat la tercera millora de la pistola. A més, el jugador no pot traspassar cap foc.

El foc, al igual que el bomber, té vida, tot i que no es visible per al jugador. Cada cop que l'aigua entra en contacte amb el foc, aquesta vida disminueix en una certa quantitat fins a arribar a cero, moment en el qual el foc desapareix i es dona per apagat.

Les monedes interactuen tant amb el jugador com amb la botiga. Quan el jugador s'acosta i travessa la moneda, aquesta desapareix i es suma al comptador de monedes del jugador, que només s'afegirà a les monedes totals si es supera el nivell. Cal esmentar que un nivell

només pot sumar el màxim de monedes que té, és a dir, si un jugador supera dos cops el mateix nivell agafant la única moneda que hi ha, només es comptarà com una moneda aconseguida de cara al comptador total per la botiga.

Pel que fa a la botiga, cada millora té associat un preu que s'ha de pagar amb monedes per desbloquejar-les. Quan es tenen suficients monedes per desbloquejar una millora, només cal fer clic sobre el botó "UNLOCK" per desbloquejar-la, el que afegirà la millora a les característiques de la pistola i disminuirà la quantitat de monedes que disposa el jugador.

2.3.3 Objectius plantejats al jugador

L'objectiu més important de qualsevol joc és arribar al final, o el que es diu més col·loquialment, passar-se el joc. Per tant, l'objectiu principal d'aquest joc és rescatar a totes les persones que hi ha atrapades per tot l'edifici. Per fer-ho, el jugador haurà de passar nivells cada cop més difícils.

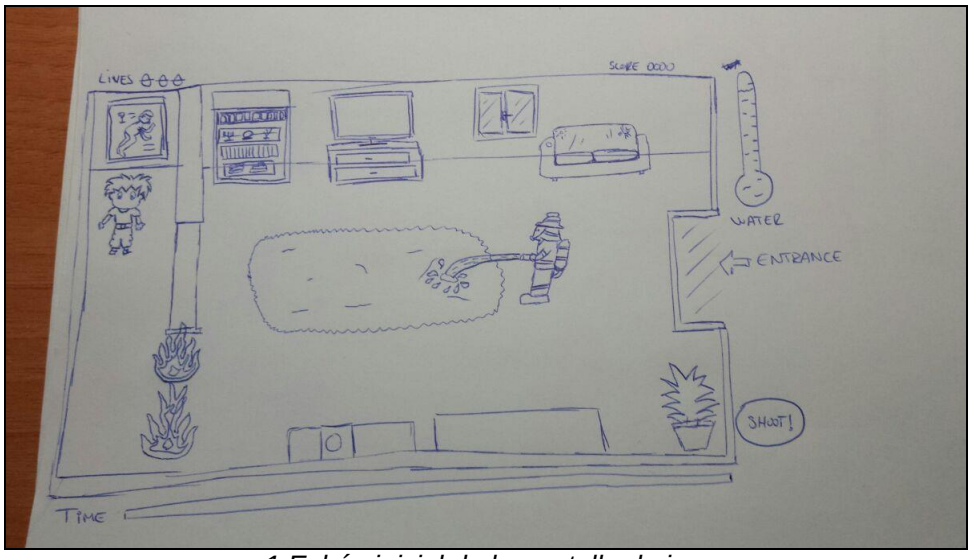
Per salvar al PNJ de cada nivell, primer s'ha d'arribar al seu costat. Això presenta un objectiu intern a cada nivell, ja que no sempre serà possible inicialment. El jugador haurà de trobar la manera d'arribar, ja sigui accionant un botó amb l'aigua o travessant foc quan és segur. Així, arribar al PNJ per salvar-lo és un objectiu.

Un altre objectiu plantejat, a més de salvar als PNJ, és el de recol·lectar monedes. Pot haver més d'una moneda a cada nivell, i el jugador haurà de buscar-les si vol tenir suficients monedes per desbloquejar les millores de la pistola i fer els següents nivells més senzills.

Per últim, cada foc apagat dóna una petita puntuació, però la quantitat d'aigua utilitzada també influeix en la puntuació obtinguda al completar el nivell, així com el temps que hagi trigat a aconseguir-ho. D'aquesta manera, es presenta l'objectiu de trobar l'equilibri entre foc apagat, aigua restant i temps necessitat per completar el nivell per tal d'aconseguir la màxima puntuació possible.

2.3.4 Concept Art: Screenshots, animació

Per aquest apartat es mostra un esbós inicial de la pantalla de joc, així com alguna imatge de l'aspecte del joc de les primeres setmanes de desenvolupament. L'aspecte gràfic del joc final i les animacions es mostren en un altre apartat més endavant.



1.Esbós inicial de la pantalla de joc.

2. Primera versió de la interfície gràfica de joc.

3. Gràfics inicials del primer nivell.

2.4 Avaluació d'engines i kits de desenvolupament

Actualment hi ha moltes opcions viables per desenvolupar jocs tant de manera professional com amateur. Com es d'esperar, cada engine té els seus avantatges i inconvenients[1] i es important tenir-ho en compte a l'hora d'escollir amb quin es vol treballar.

Per una banda tenim **Unreal Engine 4**, que és un dels més populars del món. Degut a que té un dels motors gràfics més potents, és ideal si es treballa en un joc de mida considerable i complex que necessiti un bon motor de joc 3D. Per aquesta mateixa raó, al ser tan potent la corba d'aprenentatge és una mica més alta que en altres sistemes. Actualment aquest motor es pot obtenir de manera gratuïta a la web d'Epic Games, però hi ha una taxa del 5% dels beneficis si es superen els $3.000 per quadrimestre.

Donada la seva potència, es poden desenvolupar jocs per a les millors plataformes (PC, Xbox, PlayStation), però també per altres plataformes que no necessiten d'un motor gràfic tan potent. Com que aquest projecte busca crear un joc per Android, tot i que seria totalment viable utilitzar aquest sistema, s'utilitzarà un altre més adient.

Un altre sistema per desenvolupar jocs és **GameMaker Studio 2 (GMS2)**[2]. Tot i que es poden desenvolupar jocs en 3D, està pensat per el desenvolupament de jocs en 2 dimensions. Es tracta d'un kit de desenvolupament extremadament fàcil d'utilitzar gràcies al sistema Drag&Drop (DnD) per afegir i treure elements. És recomanable per principiants que no tenen nocions de programació, ja que la quantitat de codi que s'ha de crear és mínima degut a totes les facilitats que ofereix el sistema.

GMS2 també es pot aconseguir de manera gratuïta, però es tracta d'una versió de prova amb només les funcions bàsiques (nucli). El problema trobat amb aquest sistema és que la versió gratuïta només permet crear jocs d'escriptori, motiu per el qual no es podria crear per a Android. Per tant, aquest sistema queda descartat per aquest projecte.

L'últim dels sistemes a analitzar és **Unity**. Es tracta d'un sistema molt popular per jocs Indie i plataformes mòbils, al voltant d'un 34% dels 1.000 millors jocs gratuïts d'Android estan fets amb aquest engine. Al ser fàcil d'utilitzar i permetre la creació de jocs tant 2D com 3D és molt accessible per tot tipus de desenvolupadors de jocs (tant principiants com experimentats). Té l'inconvenient que l'editor 3D que incorpora és una mica limitat en quant a les seves capacitats, fet pel qual generalment s'ha d'utilitzar software de tercers per al modelatge 3D, tot i així, al permetre tot tipus d'extensions d'arxiu no suposa cap problema greu.

[1] https://blog.instabug.com/2017/12/game-engines/ (03/03/2018)
[2] https://www.yoyogames.com/gamemaker/features (03/03/2018)

Donat que el joc a desenvolupar en aquest projecte és en 2D, no hi haurà problema en aquest aspecte.

De nou, també es pot adquirir de manera gratuïta per a us personal des de la seva web, però si els beneficis anuals dels jocs desenvolupats superen els $200.000 és obligatori contractar una versió de pagament (a variar segons els ingressos anuals). Com que no es el cas d'aquest projecte, i l'accessibilitat del sistema és adient per a desenvolupadors principiants, es creu oportú escollir aquest engine per desenvolupar el joc.

3. Implementació

3.1 El Jugador

Per implementar el jugador s'ha creat un objecte de tipus **Sprite2D** al que se li ha associat una sèrie de components per tal de que tingui el comportament esperat. A més, té també associat un component de tipus **Script** que contindrà tota la lògica de l'objecte.

Per una banda s'ha associat un component **Rigidbody2D** de tipus dinàmic. Aquest component permet aplicar forces sobre l'objecte per tal de poder simular el seu moviment. És un component realment útil, ja que controla molts aspectes i funcionalitats automàticament que. Si no ho fes, s'haurien de fer a mà. Per exemple, en cas d'estar desenvolupant un joc on intervingui la força de la gravetat, aquest component s'encarrega d'aplicar-la sobre l'objecte amb la força definida al propi editor. Com que no és el cas, per aquest projecte s'ha igualat a cero.

Gràcies a aquest component, el moviment del jugador es torna molt senzill i només cal, com ja s'ha esmentat, aplicar una força en forma de velocitat, que no és més que un vector de tipus (x,y). Per tant, per moure el personatge, només cal capturar els valors que retorna el joystick (que s'explicarà més endavant en aquest apartat), aplicar-hi una certa velocitat per fer que el personatge es mogui més ràpid, i assignar el vector amb aquests valors com a nova velocitat del component. Això provocarà el moviment del personatge en la direcció del vector.

```
void FixedUpdate () {

    Vector2 move = new Vector2(joystick.Horizontal, joystick.Vertical);

    if (move == Vector2.zero) playerMoving = false;
    else playerMoving = true;
    SetLookingDirection(move);
    ControlAnimators(move);

    Vector2 targetVelocity = move * speed;
    rb2d.velocity = targetVelocity;
```

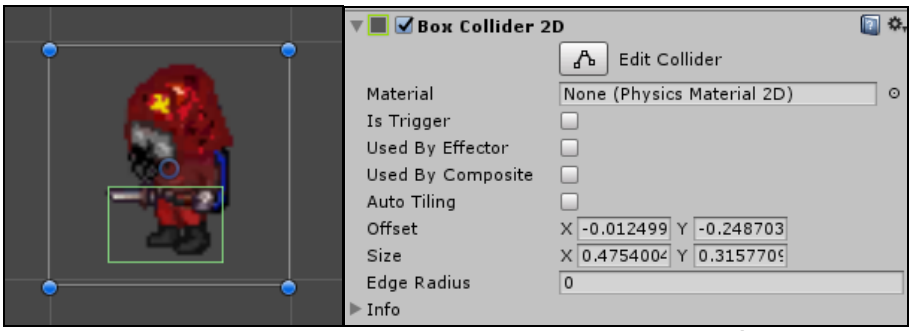
4. Codi per al moviment del jugador

Cal esmentar que aquest fragment de codi s'executa a la funció **FixedUpdate** en comptes de la funció **Update** perquè aquesta última s'executa un cop per frame i pot ser que entre execucions el moviment sigui massa gran i provoqui salts del personatge. La funció "**FixedUpdate**" s'executa més freqüentment que l'altra, per tant, és adient codificar tot allò on intervinguin físiques del joc per assegurar una actualització dels objectes contínua i evitar salts.

Pel que fa al joystick que permet el moviment del personatge, s'ha utilitzat un asset gratuit[3] de la **Asset Store d'Unity**[4] que portava un objecte prefabricat (prefab) que ja s'encarrega automàticament de detectar quan el jugador l'està movent i actualitza les variables de posició a les que accedim al executar el codi de la imatge anterior.

La part de codi que es veu entre la captura de les dades del joystick i l'assignació de la velocitat al component **Rigidbody2D** es fan servir per controlar les animacions del personatge. Aquesta part es veurà en l'apartat 4.

Un altre component que s'ha assignat a aquest objecte és el **Collider**. Aquest component permet al motor d'**Unity** detectar col·lisions entre objectes i gestionar-les automàticament. Hi ha diferents tipus de **Colliders**, en aquest cas s'utilitza un de tipus **Box**, que no és més que un rectangle. Aquest component, juntament amb els **Colliders** de la resta d'elements del joc, alliberen al programador d'haver de programar cap control respecte a les col·lisions del jugador i no el permetran travessar objectes (sempre que no es configurin de manera que ho permetin, per exemple, amb la opció **Trigger**).



*5. En verd el **Collider** del jugador, a la dreta la configuració del **Collider***

A més del control de moviment i del control d'animacions, el script del jugador també disposa de funcions per intentar salvar al PNJ i per controlar el mal que rep.

Per salvar al PNJ disposa d'una funció que es crida cada cop que es prem el botó de l'escut que hi ha a la part inferior dreta de la pantalla durant la partida. Per evitar "falsos" positius a la crida d'aquesta funció (quan hi ha lliscaments tàctils, per exemple, o si es premen 2 posicions diferents de la pantalla al mateix temps) el primer que es fa és verificar si s'ha premut el botó de salvament.

Si és tracta d'una crida vàlida, es comprova si el PNJ està a una distància prou petita com per poder iniciar el salvament. Per fer-ho, s'executa un **Raycast** al davant del jugador (en la direcció on està

[3] https://assetstore.unity.com/packages/tools/input-management/joystick-pack-107631 (03/06/2018)
[4] https://assetstore.unity.com/ (03/06/2018)

enfocat) que retorna tots els elements amb el que "xoca" al comprovar els seus **Colliders**. Si algun d'aquests elements és el PNJ, llavors s'informa al controlador de joc que s'inicia el salvament.

```
public void SaveNPC()
{
    if (Input.touchCount > 0 && EventSystem.current.IsPointerOverGameObject(Input.GetTouch(0).fingerId))
    {
        //Debug.Log(EventSystem.current.currentSelectedGameObject.tag);
        if (EventSystem.current.currentSelectedGameObject != null && EventSystem.current.currentSelectedGameObject.CompareTag("SaveButton"))
        {

            // Saving logic
            //Ray2D ray = new Ray2D(transform.position, lastMove);
            //ContactFilter2D contactFilter = new ContactFilter2D();
            RaycastHit2D[] hit = Physics2D.RaycastAll(transform.position, lastMove,1.3f);
            if ( hit.Length > 0)
            {
                for (int i = 0; i < hit.Length; i++)
                {
                    if (hit[i].collider != null && hit[i].collider.tag.Equals("NPC"))
                    {
                        gameControl.SaveTheNpc(gameObject);
                    }
                }
            }
        }
    }
}
```

6. Funció de salvament, part del jugador

Les últimes funcions codificades al jugador són per gestionar quan rep mal. Per fer-ho, disposa d'una funció que es crida quan el jugador s'apropa massa a un foc. Utilitzant una variable per donar uns segons d'invencibilitat quan el jugador rep mal s'evita que es quedi sense vida massa ràpid, ja que la funció es cridarà moltes vegades abans el jugador no s'allunyi del foc.

El jugador comença amb 100 punts de vida. Si no està en estat d'invencibilitat, es resta els punts de mal que ha de rebre de la vida restant actual i, si aquesta arriba a 0 o disminueix encara més, s'executa la funció que indica el **GameOver** de la partida. En cas de que encara quedi vida suficient per continuar, s'executa la co-rutina[5] que controla la invencibilitat i la intermitència gràfica que indica quan el jugador ha rebut ma. S'ha utilitzat una co-rutina perquè s'executa en segon pla i permet al jugador continuar movent-se mentre s'executa.

```
public void DamageMe(int dmgPoints)
{
    if (!invincible) {
        AudioManager.Instance.Play("PlayerDamaged");
        hp = hp - dmgPoints;

        if (hp <= 0)
            GameControl.instance.GameOver();
        else
            // First value is flashing time, second value is time interval
            StartCoroutine(Damaged(0.1f, 0.05f));
    }
}
```

7. Funció per gestionar el mal que rep el jugador

[5] La consulta de Stackoverflow d'on es va treure la major part del codi d'aquesta co-rutina ja no està disponible, raó per la qual s'adjunta una altra de semblant que també es va consultar: https://stackoverflow.com/questions/16114349/make-player-flash-when-hit (última resposta) (08/06/2018)

```
IEnumerator Damaged(float time, float intervalTime)
{
    invincible = true;
    //this counts up time until the float set in FlashingTime
    float elapsedTime = 0f;
    //This repeats our coroutine until the FlashingTime is elapsed
    while (elapsedTime < time)
    {
        //This gets an array with all the renderers in our gameobject's children
        Renderer[] RendererArray = GetComponentsInChildren<Renderer>();
        //this turns off all the Renderers
        foreach (Renderer r in RendererArray)
            r.enabled = false;
        //then add time to elapsedtime
        elapsedTime += Time.deltaTime;
        //then wait for the Timeinterval set
        yield return new WaitForSeconds(intervalTime);
        //then turn them all back on
        foreach (Renderer r in RendererArray)
            r.enabled = true;
        elapsedTime += Time.deltaTime;
        //then wait for another interval of time
        yield return new WaitForSeconds(intervalTime);
    }
    invincible = false;
}
```

8. Funció cridada com a co-rutina al rebre mal

3.2 PNJ

El PNJ és un element força més simple que el jugador, ja que no disposa de vida , i per tant no rep mal, i tampoc fa cap acció extra a banda del moviment. No obstant, pel que fa a l'editor, sí que disposa dels mateixos components que el jugador per tal de poder gestionar el moviment, les col·lisions i la lògica. Així doncs, també disposa d'un component **Rigidbody2D**, un **Collider** i un **Script**. La única diferència es troba en el codi del script, per tant, serà l'únic que es mostrarà en aquest apartat.

Quan el jugador ha premut el botó de salvament i estava a rang correcte del PNJ per salvar-lo, el controlador de joc cridarà la funció de salvament del PNJ per què aquest s'adoni que el volen salvar i inicïi el seguiment. Per fer-ho, s'assigna el jugador a una variable i s'activa la variable d'estat que indica quan l'estan salvant.

```
public void SaveMe(GameObject player)
{
    savior = player;
    beingSaved = true;
}
```

9. Funció per activar salvament

Al tenir activada la variable d'estat de salvament, s'executarà la part de codi de la funció **FixedUpdate**. Aquest codi captura la distància que hi ha entre el PNJ i el jugador per avaluar si es troba massa a prop o massa lluny per moure's i, si es permet el moviment, es mou en la direcció del jugador amb una velocitat determinada per la

variable **speed**. En aquest cas no s'ha utilitzat la velocitat del **Rigidbody2D** ja que al utilitzar-la, el PNJ podia intentar anar a una posició ocupada per el jugador i empentar-lo contínuament, ja que al canviar aquest de posició, la velocitat no es reduïa mai.

```
void FixedUpdate () {
    if (beingSaved)
    {
        //transform.position = lastPosition;
        Vector3 movement = transform.position;
        Vector3 offset = transform.position - savior.transform.position;
        float sqrLen = offset.sqrMagnitude;

        if ((sqrLen < 5f * 5f) && (sqrLen > 0.75f * 0.75f))
            movement = Vector3.MoveTowards(transform.position, savior.transform.position, speed);

        ControlAnimators(transform.position - movement);
        transform.position = movement;
    }
}
```

10. Part del script del PNJ que controla el seguiment al jugador

3.3 Foc

El foc és un altre dels elements principals del joc. Pot haver dos tipus de foc, normal o blau. Inicialment el foc sempre és normal, però passat un temps variable comprès entre 1-4 minuts, es tornarà blau. Mentre el foc normal es pot apagar amb la pistola d'aigua sense cap tipus de millora, per apagar el foc blau és necessari disposar de la tercera millora disponible desbloquejada. També fa més mal al jugador que el foc normal i dóna la mateixa puntuació al apagar-se.

A més, tant el foc normal com el foc blau disposen de vida, igual que el jugador. Això és degut a que sense vida, només cal que el toqui una instància d'aigua per apagar-lo, i queda poc realista, ja que generalment, el foc no s'apaga al instant al tirar-hi aigua. Així, amb la vida s'aconsegueix que no s'apagui tan ràpid i dona un plus de realisme. Cada cop que una instància d'aigua entra en contacte amb el foc, aquest perd un punt de vida i, quan perd 10 punts, s'apaga. Tant el foc normal com el foc blau tenen la mateixa quantitat de vida.

Com que no té cap moviment, sino que apareix a una certa posició i no es mou, no necessita tenir cap component **Rigidbody2D**. No obstant, en comptes de tenir un sol **Collider**, en té dos. Un és exactament igual que els del jugador i el PNJ i serveix per evitar que el jugador o el PNJ el travessin i l'altre, està configurat com a **Trigger**. Això provoca que no es tingui en compte a l'hora d'analitzar col·lisions per veure si es pot travessar o no, però al entrar en contacte amb un altre **Collider** genera un event que es pot capturar al script de l'objecte.

Amb aquest event es poden codificar dues funcions. La primera serveix per identificar els objectes que han col·lisionat amb ell, i la segona serveix per identificar els objectes que han col·lisionat amb

ell i que encara continuen “dins” de la zona delimitada per el **Collider**. D’aquesta manera, amb la primera es pot identificar quan l’aigua arriba a tocar el foc, amb el que se li resta vida. Amb la segona, en canvi, es pot identificar quan és el jugador qui està en contacte amb el foc, i procedir a fer-li mal mentre continuï dins.

```
private void OnTriggerEnter2D(Collider2D collision)
{
    if (fireType == NORMAL_FIRE)
        if (collision.gameObject.tag == "Water")
            hp--;

    if (fireType == BLUE_FIRE)
        if (collision.gameObject.tag == "Water")
            if (collision.gameObject.GetComponent<Water>().canExtinguishBlueFire)
                hp--;


}

private void OnTriggerStay2D(Collider2D collision)
{
    if (collision.gameObject.tag == "Player")
        GameControl.instance.DamagePlayer(fireDamage);
}
```

11. Funcions control col·lisions amb el foc

Cal mencionar que, quan el foc s’apaga, a més de desaparèixer informa al controlador de joc que la seva posició torna a estar disponible. Així, quan sigui el moment de generar un altre foc, podrà aparèixer en aquesta mateixa posició.

3.4 Nivells pistola

La pistola d’aigua és un objecte fill de l’objecte que fa de jugador. Aquest element no té imatge pròpia ja que està inclosa a la imatge del jugador. Com que no ha de controlar-se ni el seu moviment ni les col·lisions que pugui tenir, no té cap dels components que ho gestionen. Sí que té un component **Script** per permetre guardar les característiques de la pistola i generar l’aigua.

Realment aquest element podria haver-se unificat amb el jugador i afegir tota la lògica del seu **Script** al del jugador. Tot i així, per modularització i coherència de conceptes, ja que una pistola d’aigua és un objecte en si mateix independent de la persona que el porta, s’ha creat com a objecte de joc buit (**Empty Object**).

Així doncs, la pistola d’aigua s’encarrega de carregar les característiques de totes les millores desbloquejades al principi del nivell, amb la funció **Start**. Segons el nivell de pistola que tingui desbloquejat, adapta el rang, la capacitat i si l’aigua que dispara pot apagar el foc blau.

```
void Start()
{
    // This will do stuff based on the level of the gun?
    GunLevel = SettingsAndProgress.Instance.lastGunUnlocked;
    switch (GunLevel)
    {
        case 0:   // BASIC GUN {
            WaterBody.GetComponent<Water>().lifes = 7f;
            totalCapacity = 500;
            WaterBody.GetComponent<Water>().canExtinguishBlueFire = false;
            WaterHead.GetComponent<Water>().lifes = 7f;
            WaterHead.GetComponent<Water>().canExtinguishBlueFire = false;
            break;
        case 1: // Higher Range GUN {
            WaterBody.GetComponent<Water>().lifes = 14f;
            totalCapacity = 500;
            WaterBody.GetComponent<Water>().canExtinguishBlueFire = false;
            WaterHead.GetComponent<Water>().lifes = 14f;
            WaterHead.GetComponent<Water>().canExtinguishBlueFire = false;
            break;
        case 2: // Higher Capacity GUN {
            WaterBody.GetComponent<Water>().lifes = 14f;
            totalCapacity = 1000;
            WaterBody.GetComponent<Water>().canExtinguishBlueFire = false;
            WaterHead.GetComponent<Water>().lifes = 14f;
            WaterHead.GetComponent<Water>().canExtinguishBlueFire = false;
            break;
        case 3: // Cooler GUN {
            WaterBody.GetComponent<Water>().lifes = 14f;
            totalCapacity = 1000;
            WaterBody.GetComponent<Water>().canExtinguishBlueFire = true;
            WaterHead.GetComponent<Water>().lifes = 14f;
            WaterHead.GetComponent<Water>().canExtinguishBlueFire = true;
            break;
    }
    //*************************************
    shooting = false;
    currentCapacity = totalCapacity;
}
```

12. Inicialització característiques pistola

Indicar que s'utilitzen dos línies de codi per indicar que l'aigua pot apagar foc blau ja que existeixen dos objectes de joc d'aigua, un amb la imatge que indica el principi del raig d'aigua i el que té la imatge del "cos" del raig. La solució òptima hauria sigut guardar en un fitxer els valors que pot agafar la pistola i carregar-los directament sense fer-ho a mà aquí, però per manca de temps no ha sigut possible.

Per disparar l'aigua podria seguir una solució similar al botó de salvament, i de fet al principi funcionava així. No obstant, es va haver de canviar la que l'aigua no es disparava al mantenir el botó premut, sino que es disparava només un cop. Aquest no era el comportament desitjat i, per tant, es va canviar per la versió actual.

Així doncs, a la funció **FixedUpdate** analitza cada cop si hi ha algun element de la interfície gràfica premut i, si n'hi ha algun, mira si és el botó de disparar. Si ho és i queda capacitat a la pistola, crea instàncies d'aigua a una posició calculada per encaixar bé amb la posició i la direcció on està mirant el jugador i li assigna una velocitat per què es mogui cap a la direcció on mira el jugador. A més, per cada instància d'aigua que genera, també disminueix la capacitat

restant i actualitza la barra de capacitat de la part dreta de la interfície gràfica.

```
private void FixedUpdate()
{
    if (Input.touchCount > 0 && EventSystem.current.IsPointerOverGameObject(Input.GetTouch(0).fingerId))
    {
    //Debug.Log(EventSystem.current.currentSelectedGameObject.tag);
        if (EventSystem.current.currentSelectedGameObject != null && EventSystem.current.currentSelectedGameObject.CompareTag("ShotButton"))
        {


            if (currentCapacity > 0)
            {
                if (!shooting) AudioManager.Instance.Play("ShotWater");
                bool down = false;
                Quaternion rotation;// = Quaternion.Euler(0f, 0f, 0f);
                if (Player.lastMove.x == 0) rotation = Quaternion.Euler(0, 0, ((Player.lastMove.y < 0) ? -90f : 90f));
                else rotation = Quaternion.Euler(0, 0, ((Player.lastMove.x < 0) ? 180f : 0f));
                /*      Movement Offset     */
                float x = 0f;
                float y = 0f;
                if (Player.lastMove.x > 0) { x = -0.15f; y = 0.15f; } else if (Player.lastMove.x < 0) { x = 0.15f; y = 0.15f; }
                if (Player.lastMove.y > 0) { y = -0.15f; x = -0.15f; } else if (Player.lastMove.y < 0) { y = 0.15f; x = -0.15f; down = true; }
                Vector3 offset = new Vector3(x, y, 0);
                Rigidbody2D water = null;

                if (shooting) water = Instantiate(WaterBody, Player.transform.position - offset, rotation) as Rigidbody2D;
                else water = Instantiate(WaterHead, Player.transform.position - offset, rotation) as Rigidbody2D;

                water.velocity = transform.TransformDirection(Player.GetLastLookingDirection() * 10f);
                if (down) water.GetComponent<SpriteRenderer>().sortingOrder = 7;
                currentCapacity--;
                float fillBar = currentCapacity / totalCapacity;
                waterBar.value = fillBar;
                shooting = true;
            }
            else
            {
                AudioManager.Instance.StopPlaying("ShotWater");
                shooting = false;
            }
```

13. Codi per generar l'aigua

3.5 Aigua

Hi ha dos objectes de joc que pertanyen a l'aigua que es poden generar. Com s'ha vist al punt anterior, un s'utilitza per simular el cap del raig i l'altre per simular-ne el cos. Com que la única diferència entre els dos objectes de joc és la seva imatge, tot el que s'explica en aquest apartat inclou els dos.

Com que s'ha de moure per el nivell, l'aigua sí que té un component **Rigidbody2D**. A més, també un **Collider** i està configurat com a **Trigger**. D'aquesta manera s'aconsegueix que l'aigua travessi tots els elements del joc, inclosos el foc i el PNJ (de manera intencionada), però al entrar en contacte amb el **Collider** del botó vermell que hi ha a la paret d'alguns nivells, que és un element d'entorn que es veurà més endavant, provocarà una acció. Tot i que aquesta acció s'explicarà junt amb el botó, es mostra el codi que la genera en aquest apartat ja que forma part de l'element d'aigua.

```
private void OnTriggerEnter2D(Collider2D collision)
{

    if (collision.gameObject.tag == "FireButton")
    {

        GameControl gc = FindObjectOfType<GameControl>();
        if (!gc.fireButtonPlugged)
        {
            gc.MakeFireDisapear();
        }
    }
}
```

14. Event que executa l'acció d'entorn del botó

Per dotar l'aigua d'un cert rang, s'utilitza una variable per indicar el número de vides que li queden. Com que la pistola d'aigua ja genera l'aigua amb una velocitat determinada, no cal gestionar el seu moviment i, per tant, només cal disminuir la quantitat de vides que té cada cop que passa per la funció **Update** (que en aquest cas s'utilitza perquè el moviment el gestiona el propi motor d'**Unity**).

```
void Update () {
    if (lifes < 0)
    {
        Destroy(gameObject);
    }
    if (lifes < 2)
    {
        SpriteRenderer spriteRenderer = GetComponent<SpriteRenderer>();
        spriteRenderer.sprite = headSprite;
        spriteRenderer.sortingOrder = 4;
        transform.localScale += new Vector3(0f, 0.1f, 0f);

    }

    lifes--;
}
```

*15. Actualització vides restants i gestió de **Sprites***

Com es pot veure a la imatge, quan les vides disminueixen a 1, es canvia la imatge de l'objecte de cos a cap per tal de mantenir la coherència gràfica del raig i aconseguir l'animació d'aigua en moviment. Quan les vides arriben a 0, l'objecte es destrueix.

3.6 Moneda

Aquest element només té una missió: sumar puntuació. Per fer-ho té assignat un **Collider** configurat com a **Trigger** que al activar-se informarà al controlador de joc que s'ha recollit una moneda i que s'ha d'incrementar el comptador de monedes del nivell.

A més, també té assignat un component **Rigidbody2D** per poder iniciar l'animació un cop l'han agafat ja que, per aquest element, en comptes d'utilitzar el component **Animator** que ofereix **Unity** s'ha optat per crear l'animació amb el propi codi per explorar diferents maneres de fer la mateixa acció.

D’aquesta manera, quan el jugador travessa la moneda, aquesta es mourà verticalment cap amunt uns segons fins desaparèixer.

```
void CoinCollected()
{
    CoinCollider2D.enabled = false;
    taken = true;
    gameControl.AddCoin();
}

void OnTriggerEnter2D(Collider2D theCollider)
{
    if (theCollider.CompareTag("Player"))
    {
        CoinCollected();
    }
}
```

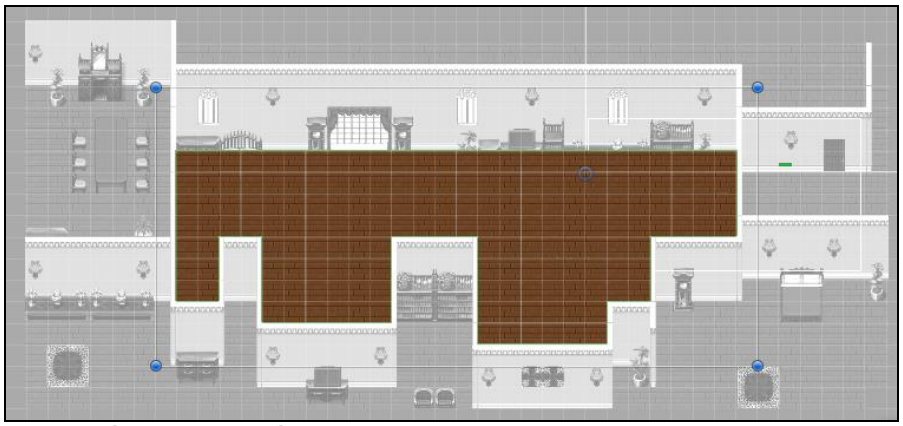
16. Codi per recollir moneda

3.7 Entorn

Per desenvolupar l’entorn del joc, el que seria l’escenari de cada nivell, s’han utilitzat una sèrie de **Tilemaps**, que són un tipus d’objecte estil **Grid** que ofereix **Unity** per “dibuixar” mapes a partir de **Tiles**. Amb aquesta eina s’ha simplificat moltíssim la creació dels escenaris i ha permès dedicar més temps i esforç en altres aspectes del joc.

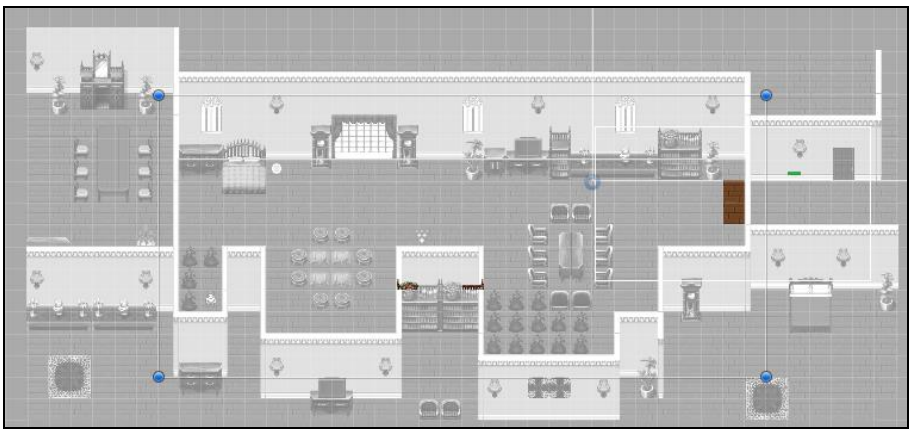
En el primer nivell hi ha 4 **Tilemaps**, en el segon nivell hi ha 5, i a partir del tercer nivell hi ha 6. Això és degut a que al incrementar la dificultat de cada nivell, s’han hagut d’afegir elements que no calia afegir als nivells més inicials. Seguidament s’aprofundirà més en aquest aspecte.

A tots els nivells hi ha un **Tilemap** dedicat exclusivament al terra, que és tota la zona per la que el jugador es pot moure i on apareixen tots els focs. Està separat de la resta d’elements d’entorn ja que així simplifica molt més el control de les posicions on pot generar-se el foc.



*17. Visió de la superfície del **Tilemap** del terra en el conjunt del nivell 1*

El següent **Tilemap** també es troba a tots els nivells, és un dels més petits que hi ha en tot el joc i té una única missió. Es tracta de l'entrada i sortida del nivell. Al començar el nivell, el jugador comença la partida situat a les seves caselles. Només serveix per actuar com a punt de salvament, és a dir, serveix per detectar quan el PNJ arriba a l'entrada/sortida del nivell i quan ho fa, significa que s'ha salvat.



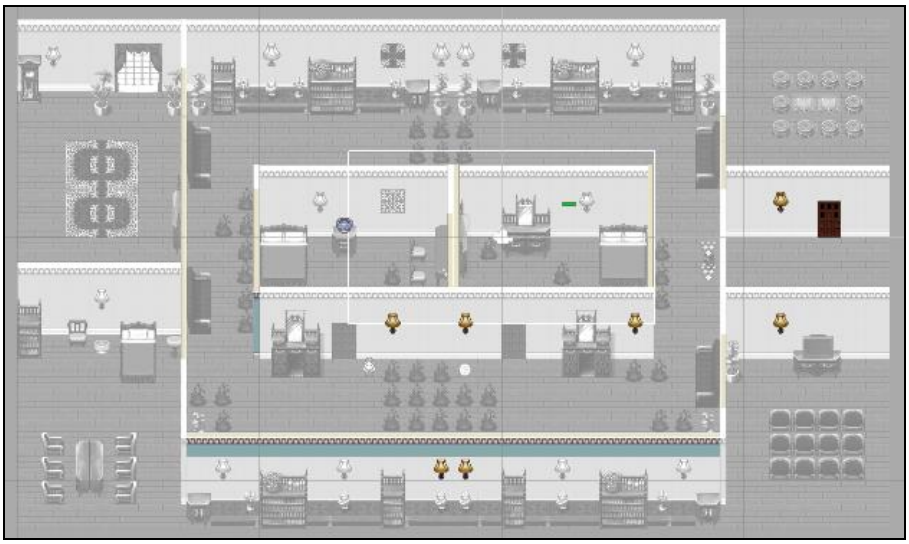
*18. Visió del **Tilemap** que actua d'entrada/sortida del nivell*

Seguidament, tenim els **Tilemaps** que simulen les parets i la decoració. Com el propi nom indica, un s'encarrega de "dibuixar" totes les parets del nivell, i l'altre de tota la decoració, respectivament. Cal destacar que per definir el "sostre" de les parets en algunes zones s'ha hagut d'utilitzar el **Tilemap** de la decoració i, en aquelles zones, el del terra per la decoració. Això és degut a problemes de superposició de **Tiles** que provocaven errors gràfics.

*19. Visió del **Tilemap** de decoració amb zones de sostre de parets*

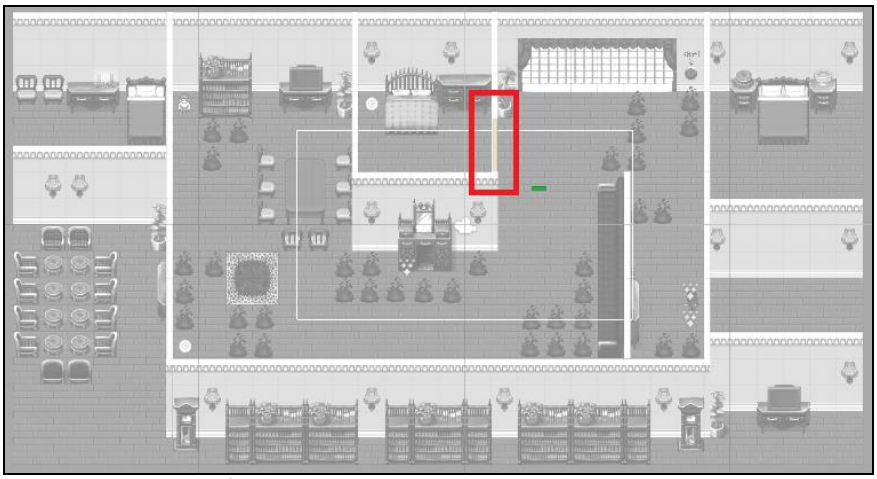
Per la nivell 2 s'ha afegit un **Tilemap** extra que actua de segona paret. Això ha facilitat més el fet de "dibuixar" les parets del nivell sense haver de recórrer al **Tilemap** de la decoració, permetent més combinacions de decoració que el nivell 1 no permetia. Es podria haver aplicat retroactivament al primer nivell però, de nou, al no

afectar directament a l'experiència de joc s'ha decidit enfocar els esforços a altres detalls més importants.



*20. Visió del **Tilemap** de paret 2 per "dibuixar" el sostre d'algunes parets*

Per últim, a partir del nivell 3 s'ha afegit un altre **Tilemap** que permet crear zones que a priori semblen inaccessibles però que en realitat sí que ho son. S'utilitzen **Tiles** amb l'aspecte de sostre de les parets per tal de simular una paret en l'eix vertical que a priori no s'hauria de poder travessar. No obstant, a banda del terra, no hi ha cap **Tile** de cap altre **Tilemap** en aquella posició i, per tant, com que aquest **Tilemap** no té **Colliders** associats, permet al jugador travessar-lo, deixant-lo accedir a habitacions que semblen tancades.



*21. Visió del **Tilemap** de paret invisible del nivell 2*

Un cop vists tots els **Tilemaps** que formen part de l'entorn dels nivells cal esmentar que tots, excepte el d'entrada i el de sostres invisibles tenen **Colliders** associats. Cal veure que no són del mateix tipus que els utilitzats als altres objectes del joc.

Els **Colliders** utilitzats s'anomenen **Tilemap Collider 2D** i, per fer-los més eficients i evitar que es creï un **Collider** per cada cel·la de cada **Tilemap**, s'ha afegit un component anomenat **Composite Collider 2D** que al indicar que s'utilitza, crea un únic **Collider** al perímetre de cada agrupació de cel·les del **Tilemap**. Gràcies a això, el jugador no pot abandonar mai la zona del nivell corresponent al terra i tampoc pot travessar la decoració que hi ha dins de la zona jugable. Això és més fàcil d'apreciar en la imatge que es mostra seguidament.

*22. Visió del **Collider** (línia verda) de la decoració del nivell 1*

A partir del nivell 3 es pot trobar l'últim element d'entorn implementat al joc. Es tracta del botó vermell del que s'ha parlat a l'apartat de la implementació de l'aigua. Aquest objecte de joc només consta d'un **Collider** circular per detectar quan l'aigua hi col·lisiona i generalment està col·locat a zones on el raig no arriba si no s'ha desbloquejat la millora de la pistola que proporciona més rang.

23. Botó vermell

Al activar aquest botó amb l'aigua, una sèrie de focs blaus del nivell que bloquegen l'accés a alguna zona desapareixeran, permetent així passar a l'altre banda i completar els objectius d'aquella zona, ja

sigui obtenir una moneda, arribar al PNJ o disparar a algun altre botó per obrir alguna altra zona.

3.8 Puntuació

La puntuació és una part important de l'experiència de joc, ja que és vital que els jugadors sentin la progressió i la recompensa de completar els nivells amb certa destresa. Per aquest motiu s'ha afegit un sistema de puntuació molt senzill que es calcula al completar un nivell i es guarda de manera permanent.

Ja s'ha esmentat a punts anteriors que el foc fa guanyar punts quan s'apaga, concretament 5, i al salvar un PNJ s'atorguen 100 punts. A més, al completar el nivell, es calcula una puntuació final en base a la puntuació total de focs apagats més el PNJ salvat, el temps que ha passat des del principi del nivell i l'aigua que s'ha gastat. Concretament, la fórmula que s'utilitza per calcular la puntuació és la següent.

```
float timeScore = (timeText.GetComponent<TimeManager>().timeLeft / timeText.GetComponent<TimeManager>().startingTime)*100;
float waterScore = waterGun.GetComponent<WaterGun>().GetWaterLeft();
int finalScoreInt = (int)(score * timeScore * waterScore);
```

24. Fórmula per calcular la puntuació final d'un nivell

Després de fer moltes proves, s'ha vist que actualment la variable **waterScore** té un pes potser massa important en la puntuació final, i si es gasta massa aigua no hi ha mai manera d'aconseguir una puntuació alta. Per tant, queda pendent per futures modificacions aplicar un sistema de pesos/modificadors a cada variable per tal d'equilibrar el seu pes a la fórmula.

Un cop es finalitza el nivell, es guarda un registre de la puntuació que s'ha aconseguit. Per cada nivell, només es guarda la puntuació màxima aconseguida. D'aquesta manera, quan es completa un nivell amb una puntuació més alta que la que hi havia enregistrada per aquell nivell, es mostra un missatge de nou rècord i es guarda la nova puntuació.

S'ha decidit guardar aquest registre per tal de facilitar en un futur la implementació d'un sistema de rànquings d'usuaris, ja sigui en local o a través d'internet en una actualització del joc.

3.9 Temps

El temps afegeix certa dificultat al joc, ja que exerceix pressió al jugador per tal de completar el nivell abans que s'esgoti. Si el temps arriba a 0, apareix la pantalla de **GameOver** i s'ha de reiniciar el nivell o tornar al menú principal.

Per controlar el temps s'ha utilitzat una classe exclusiva associada a la pròpia etiqueta que mostra el temps restant, d'aquesta manera s'allibera una petita part de la feina del controlador de joc. L'únic que fa aquesta classe és iniciar amb els segons que s'indica a l'editor d'**Unity** i va restant 1 al valor cada segon que passa. Quan el comptador arriba a 0 avisa al controlador de joc que s'ha acabat el temps.

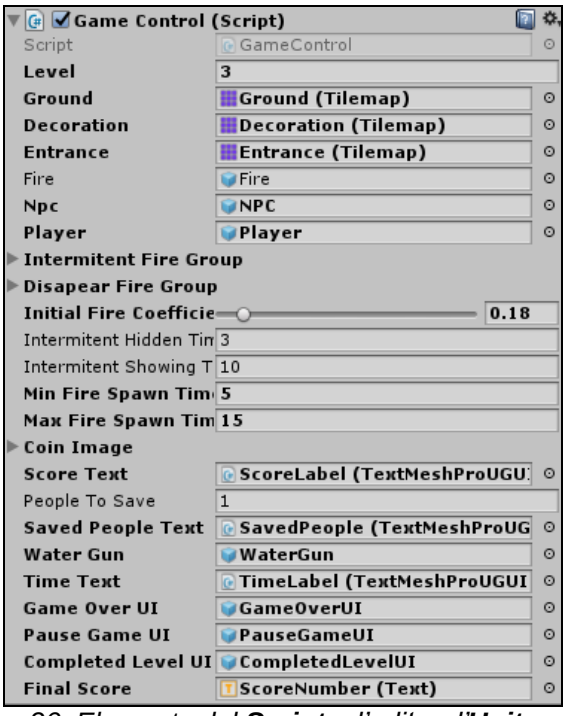
```
void Update () {
    timeLeft -= Time.deltaTime;
    theText.SetText("Time Left: "+Mathf.Round(timeLeft) +" seconds");

    if (timeLeft <= 0.0)
        FindObjectOfType<GameControl>().GameOver();
}
```

25. Script que controla el temps restant

3.10 Controlador de joc

El controlador de joc és l'element més complex de tot el projecte i el que té el **Script** més extens associat, ja que ha de controlar tot el que passa durant el nivell. Té molts objectes del joc associats per poder dur a terme aquest control tan exhaustiu.



*26. Elements del **Script** a l'editor d'**Unity***

La primera particularitat que té aquest element és que és de tipus **Singleton**. Això significa que durant la partida, nomes hi haurà un objecte de joc d'aquest tipus. Aquesta característica s'aconsegueix igualant la instància a ella mateixa si no n'hi ha cap o destruint

l'objecte quan es crea si ja hi ha un objecte d'aquest tipus instanciat, on la variable que guarda la instància és estàtica.

```
public static GameControl instance;

void Awake () {
    // To make it singleton
    if (instance == null)
    {
        instance = this;
    }
    else if (instance != this)
    {
        Destroy(gameObject);
    }
```

*27. Exemple de classe **Singleton***

Aquesta comprovació es realitza a la funció **Awake** i no a **Start** ja que la primera s'executa abans que la segona, i interessa que el control d'instància sigui el primer que es faci.

Seguidament es calcula i s'omple el vector de posicions disponibles per al foc a partir de totes les posicions del **Tilemap** del terra. Per fer aquest càlcul, es mira cada posició del terra i es comprova que en aquella posició no hi ha cap element de decoració ni de l'entrada. Així, s'evita que puguin aparèixer focs en aquelles posicions ja que no apareixeran al vector de posicions disponibles.

Abans d'inserir la posició al vector de posicions disponibles es pot apreciar una última comprovació. Això és degut a que quan hi havia foc intermitent com a element, que s'explicarà què és exactament més endavant, es donaven casos en que apareixia foc normal a posicions de foc intermitent mentre durava la intermitència. D'aquesta manera, es descarten inicialment totes aquelles posicions que tinguin foc blau mitjançant un control de col·lisions que podem obtenir de la funció **OverlapBoxNonAlloc** de la classe **Physics2D.**

```
availablePlaces = new List<Vector3>();
for (int n = Ground.cellBounds.xMin; n < Ground.cellBounds.xMax; n++)
{
    for (int p = Ground.cellBounds.yMin; p < Ground.cellBounds.yMax; p++)
    {
        Vector3Int localPlace = (new Vector3Int(n, p, (int)Ground.transform.position.y));
        //Vector3 place = Ground.CellToWorld(localPlace);
        Vector3 place = Ground.GetCellCenterLocal(localPlace);
       // Vector3 finalPlace = Ground.LocalToWorld(place);
        if (Ground.HasTile(localPlace) && !Decoration.HasTile(localPlace) && !Entrance.HasTile(localPlace))
        {
            int collisions = Physics2D.OverlapBoxNonAlloc(place, Fire.GetComponent<BoxCollider2D>().size, Fire.transform.eulerAngles.z, results);
            //Tile at "place"
            if (collisions == 0)
                availablePlaces.Add(place);
        }

    }
}
```

28. Codi que captura i guarda les posicions possibles on pot aparèixer el foc

Un cop s'obté aquest vector, es pot procedir a omplir el nivell amb una certa quantitat de focs, que vindrà determinada per el valor de coeficient inicial de focs. Aquest coeficient actua de manera que s'omplin un cert percentatge del total de posicions disponibles.

Per fer aparèixer cada foc, es crida la funció **SpawnFire**, que agafarà una posició aleatòria del vector de posicions, controlarà que en aquella posició no hi hagi cap objecte que pugui col·lisionar amb el foc, crea una instància de l'objecte foc en aquella posició i elimina la posició del vector de posicions disponibles. D'aquesta manera s'evita que es generi foc a la posició del jugador o del PNJ.

```
public void SpawnFire()
{
    // Create a gameobject fire at a random position
    //bool placed = false;

    int pos = (int)Random.Range(0f, availablePlaces.Count);
    int collisions = Physics2D.OverlapBoxNonAlloc(availablePlaces[pos], Fire.GetComponent<BoxCollider2D>().size, Fire.transform.eulerAngles.z, results);
    //Debug.Log(availablePlaces[pos]);
    //Debug.Log("collisions= " + collisions);
    if (collisions == 0)
    {
        //GameObject fire = (GameObject)Instantiate(Fire, availablePlaces[pos], transform.rotation);
        Instantiate(Fire, availablePlaces[pos], transform.rotation);
        availablePlaces.RemoveAt(pos);
    }
}
```

29. Codi de generació dels focs

Cal esmentar que, de manera intencionada, en cas de que la posició obtinguda no estigui disponible perquè hi ha algun element que bloquegi la posició, es descarta la generació del foc i es tornarà a intentar amb una posició nova quan hagi passat l'interval de generació de foc un altre cop.

Un cop generat tot el foc inicial del nivell, només cal inicialitzar algunes variables i ja està preparat per començar la partida. Com que hi ha varies funcions que només s'utilitzen com a canal de comunicació entre objectes, no cal explicar-les. Excepte una que sembla molt simple però que és bàsica per fer funcionar una de les mecàniques del joc. La funció **MakeFireDisapear** és la que s'executa quan es dispara al botó vermell dels nivells que el tenen. Aquesta funció elimina del nivell tots els focs blaus assignats al grup de focs que desapareix al prémer el botó vermell.

```
public void MakeFireDisapear()
{
    fireButtonPlugged = true;
    foreach (GameObject go in DisapearFireGroup)
    {
        //Debug.Log(go);
        Destroy(go);
    }
}
```

30. Funció per fer desaparèixer els focs assignats

És moment de mostrar el funcionament del controlador de joc a cada execució de la funció **Update**. El primer que s'executa en aquesta funció és el control de generació de foc, sempre que hagi passat l'interval establert entre focs. Si és moment de generar foc i hi ha posicions disponibles al vector, s'executa la funció vista anteriorment per generar un foc i es calcula el nou temps per la pròxima generació, que és variable entre un rang de segons establert

inicialment a l'editor d'**Unity**, com es pot apreciar a la imatge número 26 d'aquest mateix document.

```
// Spawning fire
if (Time.time > nextActionTime)
{
    // There are still free places
    if (availablePlaces.Count > 0)
    {
        nextActionTime = Time.time + Random.Range(minFireSpawnTime, maxFireSpawnTime);
        SpawnFire();
    }
}
```

31. Codi que genera foc cada cert interval de temps

Com a millora, en comptes de només generar un foc, es podria augmentar la dificultat dels nivells fent possible que es generi més d'un foc al mateix temps. Queda com a futures millores / actualitzacions.

Seguidament, es comprova si és moment de fer intermitent el foc assignat a ser-ho. En alguns nivells, pot haver una sèrie de focs blaus assignats per a actuar com a focs intermitents. Això vol dir que, mentre duri la intermitència dels focs, els seus **Colliders** estaran desactivats i, per tant, es podran travessar i no es rebrà mal. Això permet crear situacions on el jugador ha de controlar bé els temps per poder accedir a zones particulars dels nivells temporalment. Per fer-ho, s'utilitza la mateixa co-rutina que per l'efecte intermitent del jugador al rebre mal, però adaptada als focs.

```
// Intermitent Fire
if (IntermitentFireGroup != null && IntermitentFireGroup.Length > 0)
{
    if (Time.time > nextIntermitentSwap)
    {
        // If fire is active
        if (active)
        {
            active = false;
            nextIntermitentSwap = Time.time + intermitentHiddenTime;
            //IntermitentFire.SetActive(false);
            for (int i = 0; i < IntermitentFireGroup.Length; i++)
                StartCoroutine(Flash(0.6f, 0.1f, i));
        }
        else
        {
            active = true;
            nextIntermitentSwap = Time.time + intermitentShowingTime;
        }

    }
}
```

32. Porció de codi que crida i provoca la intermitència dels focs

```
IEnumerator Flash(float time, float intervalTime, int index)
{
    foreach (BoxCollider2D c in IntermitentFireGroup[index].GetComponents<BoxCollider2D>())
    {
        c.enabled = false;
    }
    //this counts up time until the float set in FlashingTime
    float elapsedTime = 0f;
    //This repeats our coroutine until the FlashingTime is elapsed
    while (elapsedTime < time)
    {
        //This gets an array with all the renderers in our gameobject's children
        Renderer[] RendererArray = IntermitentFireGroup[index].GetComponentsInChildren<Renderer>();
        //this turns off all the Renderers
        foreach (Renderer r in RendererArray)
            r.enabled = false;
        //then add time to elapsedtime
        elapsedTime += Time.deltaTime;
        //then wait for the Timeinterval set
        yield return new WaitForSeconds(intervalTime);
        //then turn them all back on
        foreach (Renderer r in RendererArray)
            r.enabled = true;
        elapsedTime += Time.deltaTime;
        //then wait for another interval of time
        yield return new WaitForSeconds(intervalTime);
    }
    foreach (BoxCollider2D c in IntermitentFireGroup[index].GetComponents<BoxCollider2D>())
    {
        c.enabled = true;
    }
}
```

*33. Funció que desactiva els **Colliders**, genera la intermitència i els activa de nou*

Per últim, la funció **Update** també controla quan el PNJ arriba a la zona de sortida per tal de detectar-lo com a salvat. Per fer-ho comprova si la posició del PNJ es troba dins d'alguna de les **Tiles** del **Tilemap** de l'entrada i, si hi és, el detecta com a salvat i gestiona la puntuació i la funcionalitat de completar nivell. Esmentar que aquesta part de codi està preparada per tenir més d'un PNJ a salvar, però per problemes de temps no s'ha implementat a la resta del projecte.

```
// If the npc still exists
if (Npc != null) {
    // Check npc and character order layer
    if (Player.transform.position.y < Npc.transform.position.y)
    {
        Player.GetComponent<SpriteRenderer>().sortingOrder = 6;
        Npc.GetComponent<SpriteRenderer>().sortingOrder = 5;
    }
    else
    {
        Player.GetComponent<SpriteRenderer>().sortingOrder = 5;
        Npc.GetComponent<SpriteRenderer>().sortingOrder = 6;
    }

    // Check if NPC reached the exit
    if (Entrance.HasTile(Entrance.WorldToCell(Npc.transform.position)))
    {
        //Debug.Log("NPC Saved!");
        AddPoints(100);
        Destroy(Npc);
        savedPeople++;
        savedPeopleText.SetText("PEOPLE: "+savedPeople + "/" + peopleToSave);

        if (savedPeople == peopleToSave)
        {
            CompletedLevel();
        }
    }
}
```

34. Porció de codi que controla quan es salva al PNJ i l'ordre de capes entre jugador i PNJ

A més de controlar això, també porta un senzill control entre la posició del PNJ i la posició del jugador per tal de canviar l'ordre dels dos objectes a les capes. D'aquesta manera s'aconsegueix l'efecte de profunditat, ja que quan el jugador està per sobre el PNJ, aquest últim estarà superposat al jugador, i quan estigui a sota, serà el jugador qui estigui superposat al PNJ.

Per acabar amb el controlador de joc, falta veure el control que té sobre els menús de la partida. Hi ha 3 menús que es poden obrir durant la partida. El primer és el menú de **GameOver**, que apareix quan la vida del jugador o el temps arriben a 0. La segona és el menú de pausa, que apareix quan el jugador prem el botó de pausa que es pot veure a la part superior esquerra de la pantalla. Aquests dos menús no tenen gaire dificultat i per tant no cal aprofundir en ells.

El tercer i últim és el menú que apareix al completar nivell, amb la puntuació que s'ha aconseguit durant la partida i la possibilitat de continuar al pròxim nivell, tornar a intentar el nivell que s'acaba de completar o tornar al menú principal. Al completar el nivell, la funció que es crida s'encarrega de calcular la puntuació total del nivell, comparar-la amb la puntuació rècord que tenia aquell nivell, si és que en tenia cap i gestionar tot el que envolta la finalització d'un nivell, com és guardar la puntuació i les monedes obtingudes, i desbloquejar el següent nivell, si és que ho estava, sempre que no superi el total de nivells implementats al joc.

```
public void CompletedLevel ()
{
    AudioManager.Instance.StopAllSound();
    AudioManager.Instance.Play("LevelCompleted");
    Time.timeScale = 0f;

    // calculate score and such
    // Formula: finalScore = score + timeScore + waterScore
    //int timeScore = timeText;
    float timeScore = (timeText.GetComponent<TimeManager>().timeLeft / timeText.GetComponent<TimeManager>().startingTime)*100;
    float waterScore = waterGun.GetComponent<WaterGun>().GetWaterLeft();
    int finalScoreInt = (int)(score * timeScore * waterScore);

    finalScore.text = finalScoreInt + "";

    if(SettingsAndProgress.Instance.lastLevelUnlocked == level) SettingsAndProgress.Instance.lastLevelUnlocked++;
    if (SettingsAndProgress.Instance.levelProgress[level - 1].bestScore < finalScoreInt)
    {
        // New record
        finalScore.text += " New Record!";
        SettingsAndProgress.Instance.levelProgress[level - 1].bestScore = finalScoreInt;
        // show label new record
    }
    if (SettingsAndProgress.Instance.levelProgress[level - 1].achievedCoins < coins && SettingsAndProgress.Instance.levelProgress[level - 1].level.totalCoins >= coins)
    {
        // got one more coin
        SettingsAndProgress.Instance.coins += coins - SettingsAndProgress.Instance.levelProgress[level - 1].achievedCoins;
        SettingsAndProgress.Instance.levelProgress[level - 1].achievedCoins = coins;
    }

    SettingsAndProgress.Instance.SaveLevelProgress();

    completedLevelUI.SetActive(true);
}
```

35. Funció que s'encarrega de la gestió de finalització d'un nivell

3.11 Controlador de menús

El controlador de menús es tracta d'un objecte buit amb un **Script** associat que recull totes les funcions necessàries per als botons dels menús del joc. Per tant, recull les funcions per tornar al menú principal, reiniciar el nivell, continuar la partida quan s'està al menú de pausa, carregar el següent nivell, carregar un nivell concret, obrir

el menú de selecció de nivells, obrir el menú d'opcions, obrir la botiga i, per últim, tancar el joc.

No es considera necessari aprofundir més en aquest objecte ja que les seves funcions es basen en reproduir el so de click de botó, executar l'acció pertinent (ja sigui desactivar el menu principal o parar/iniciar el temps del nivell, etc...) i activar o desactivar els menús pertinents.

3.12 Càmera

La càmera és un altre dels elements del joc més simples però alhora més importants ja que, sense càmera, seria impossible veure l'escenari de joc i per tant jugar. En realitat hi ha dos objecte càmera al joc, una al menú principal, que no té cap funcionalitat especial i és fixa, ja que només serveix per veure els menús, i l'altre una mica més complexa que es troba a cada nivell.

Aquesta darrera càmera l'únic que fa és posicionar-se a l'escenari agafant la posició del jugador més un valor d'offset com a centre i es va actualitzant a mesura que el jugador es mou per mantenir-lo al centre. No cal mostrar parts de codi ja que només es tracta d'una sola instrucció.

3.13 So

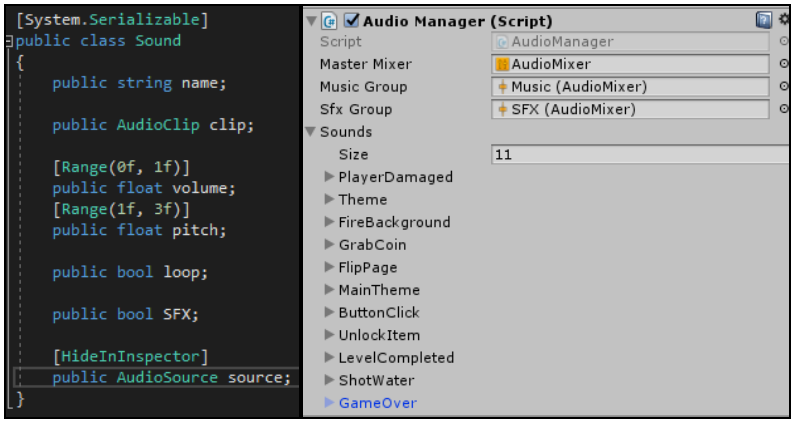
Per controlar el so del joc s'ha creat un objecte que actua de controlador de so seguint un tutorial de **Youtube**[6] molt útil i adaptant-lo a les necessitats del projecte, afegint AudioMixers per poder controlar el so mitjançant el menú d'opcions.

El controlador de so té una particularitat no vista fins ara al projecte, i és que no es destrueix mai entre escenes. És a dir, és de tipus **Singleton** i, a més, gràcies a la instrucció **DontDestroyOnLoad(gameObject)** no es destrueix mai quan es canvia d'escena. Això s'ha fet així per optimitzar la càrrega de cada nivell. Al principi, cada cop que es carregava un nivell es destruïa la instància anterior i es creava una instància nova, amb tota la càrrega de pistes d'audio que suposa. Evidentment, trigava massa a carregar cada nivell, per tant, es va decidir fer aquesta càrrega només un cop al iniciar el joc i mantenir-lo sempre instanciat per disminuir el temps de càrrega dels nivells. Després de provar-ho, el resultat va ser molt positiu.

Aquest objecte s'encarrega de guardar un vector de pistes d'audio que s'assignen de manera molt senzilla des de l'editor d'**Unity**. Per

[6] https://www.youtube.com/watch?v=6OT43pvUyfY (20/05/2018)

fer-ho, s'ha creat una classe de **Sound** que guarda la pista d'audio, un nom per identificar-la, informació sobre el volum base, si s'ha de reproduir en bucle o si és un efecte de so (per separar els efectes de la música de fons) i, el més important, un **AudioSource**, que és qui generarà el so a partir de la pista assignada.



*36. Classe **Sound** i mostra de l'audio manager a l'editor d'**Unity***

Per al correcte funcionament del controlador de so, aquesta classe s'ha de definir com a **Serializable**. El controlador disposa de funcions per iniciar una sèrie de musica/sons predefinits al iniciar el menú principal o una partida del joc, una funció per parar tot el so per quan hi ha transicions entre escenes (ja que aquesta classe no es destrueix mai), una funció per parar un so determinat i una funció per iniciar un so determinat. Aquesta última funció és la funció que s'executa a totes les parts del codi on es vol reproduir un so, ja sigui d'efectes de so o de música de fons.

És important mencionar que dins de l'**AudioMixer** general s'han creat dos **AudioMixerGroup** per separar el control de volum de la música del joc i dels efectes de so per separat. D'aquesta manera, al inicialitzar el controlador de so, s'assigna a cada instància de **Sound** l'**AudioMixerGroup** corresponent.

```
foreach (Sound s in sounds)
{
    s.source = gameObject.AddComponent<AudioSource>();
    s.source.clip = s.clip;

    s.source.volume = s.volume;
    s.source.pitch = s.pitch;
    s.source.loop = s.loop;

    if (s.SFX)
    {
        s.source.outputAudioMixerGroup = sfxGroup;
    }
    else
    {
        s.source.outputAudioMixerGroup = musicGroup;
    }
}
```

*37. Assignació de paràmetres al **AudioSource** de cada instància de **Sound***

3.14 Persistència

En un dels últims apartats de la implementació, s'ha de parlar de la persistència. El joc disposa d'un sistema de persistència que manté dos arxius que es carreguen per definir opcions de so i progrés del joc. Tota la informació referent d'aquest apartat s'ha extret del tutorial oficial d'**Unity** "**Persistence – Saving and Loading Data**[7]". Aquest objecte també és de tipus **Singleton** i **DontDestroyOnLoad** per tal que sempre sigui el mateix durant tota l'execució del joc. D'aquesta manera es pot guardar informació entre nivells.

Per poder guardar i mantenir informació sobre les opcions escollides i sobre el progrés del joc durant la partida, l'objecte defineix variables que guarden aquesta informació. A l'hora de carregar les dades guardades des del fitxer, primer es llegeix el fitxer i després s'omplen les variables amb les dades carregades. A l'hora de guardar el progrés i les opcions, el procés és el contrari, es guarden els valors de les variables en instàncies de les dues classes internes i es converteixen en binaris que es guarden en dos fitxers.

Per la persistència referent a les opcions de so, es guarda un arxiu amb el nivell de volum que l'usuari ha definit tant per la música de fons com per els efectes de so, així com informació sobre els **Toggle Buttons** que defineixen si es vol tenir silenciada la música o els efectes.

Per fer-ho, es defineix una classe interna al **Script** anomenada **SettingsInfo**, de tipus **Serializable** (que indica que es pot serialitzar en un fitxer binari) on es guarda els nivells de so desitjats i si es volen silenciar o no.

```
[Serializable]
class SettingsInfo
{
    /* Music */
    public float musicLevel;
    public bool music;

    /* SFX */
    public float sfxLevel;
    public bool effects;
}
```

*38. Classe **SettingsInfo***

[7] https://unity3d.com/es/learn/tutorials/topics/scripting/persistence-saving-and-loading-data (23/05/2018)

```
public void LoadSettings()
{
    if (File.Exists(Application.persistentDataPath + "/settingsInfo.dat"))
    {
        BinaryFormatter bf = new BinaryFormatter();
        FileStream file = File.Open(Application.persistentDataPath + "/settingsInfo.dat", FileMode.Open);
        SettingsInfo settings = (SettingsInfo)bf.Deserialize(file);
        file.Close();

        musicLevel = settings.musicLevel;
        music = settings.music;
        sfxLevel = settings.sfxLevel;
        effects = settings.effects;
    }
    else
    {
        musicLevel = 0;
        music = true;
        sfxLevel = 0;
        effects = true;
    }
}
```

39. Procés de càrrega de les opcions de so

```
public void SaveSettings()
{
    BinaryFormatter bf = new BinaryFormatter();
    FileStream file = File.Create(Application.persistentDataPath + "/settingsInfo.dat");

    SettingsInfo settings = new SettingsInfo();
    settings.musicLevel = musicLevel;
    settings.music = music;
    settings.sfxLevel = sfxLevel;
    settings.effects = effects;

    bf.Serialize(file, settings);
    file.Close();
}
```

40. Procés de guardat de les opcions de so

Per poder guardar el progrés del joc s'ha creat una estructura amb més subclasses que per les opcions de so. Primer s'ha creat una classe bàsica per definir els nivells de joc anomenada **Level**. Aquesta classe guarda el número de nivell, el número de persones a salvar per aquell nivell i el número total de monedes que hi conté. Seguidament, s'ha creat una segona classe anomenada **LevelProgress** que conté una variable de la classe **Level**, que conté tota la informació del nivell, una variable per guardar la puntuació màxima obtinguda i una variable per guardar el número de monedes aconseguides en aquell nivell. Cal destacar que aquest número de monedes, tal como està programat el joc, mai pot superar el número de monedes totals del nivell. D'aquesta manera s'evita que un jugador pugui obtenir dues monedes per desbloquejar millores repetint un nivell fàcil que en tingui només una.

```
[System.Serializable]
public class Level {

    public int level;

    public int peopleToSave;

    public int totalCoins;
}
```

```
[System.Serializable]
public class LevelProgress {

    public Level level;

    public float bestScore;

    public int achievedCoins;
}
```

41. Classes per definir el progrés de cada nivell

Igual que per les opcions de so, a l'objecte **SettingsAndProgress** s'ha definit una classe **Serializable** que guarda el codi de l'últim nivell desbloquejat, un vector d'objectes de tipus **LevelProgress** amb la informació de progrés de tots els nivells, una variable per guardar el total de monedes que té el jugador i una altra per guardar quina és la última millora desbloquejada de la pistola. D'aquesta manera, mitjançant el mateix procés exacte que amb les opcions de so, es poden carregar i guardar les dades referents al progrés del joc.

Ja que es tracta d'una classe **Singleton** i que no es destrueix mai, tots els objectes del joc poden consultar en qualsevol moment les dades que necessitin per al correcte funcionament del joc, ja sigui saber els nivells o les millores de la pistola desbloquejats/des, les monedes totals disponibles, les opcions de so definides per l'usuari, etc...

3.15 Tutorial

L'últim apartat que forma part de la implementació mostra el tutorial del joc. Per manca de temps s'ha fet un tutorial molt senzill que es mostra al iniciar el primer nivell a cada execució del joc. És a dir, cada cop que es comença a jugar, el tutorial apareix quan es juga al primer nivell, però un cop vist, ja no apareixerà més encara que es reinicïi el nivell. Quan es tanca el joc i es torna a obrir, el tutorial tornarà aparèixer al començar a jugar amb el primer nivell.

Per aconseguir aquest comportament, s'ha creat una variable a la classe **SettingsAndProgress** que indica si cal mostrar el tutorial o no. Al iniciar un nivell, el controlador de joc comprova si és el nivell 1 i, si la variable esmentada indica que s'ha de mostrar el tutorial, es mostra. Si no, no es mostra.

NOTA: Degut a que aquesta funcionalitat ha estat la última en implementar-se, és possible que alguna de les imatges mostrades per al controlador de joc tingui algun canvi respecte als scripts que es poden trobar als fitxers del codi font del joc (concretament la imatge que mostra l'execució inicial del **Script**).

4. Interfícies

4.1 Menús

Per crear els menús s'ha utilitzat un **Asset** gratuït de la **Asset Store** d'**Unity** anomenat **Casual Game GUI Skin**[8].

4.1.1 Fora de la partida

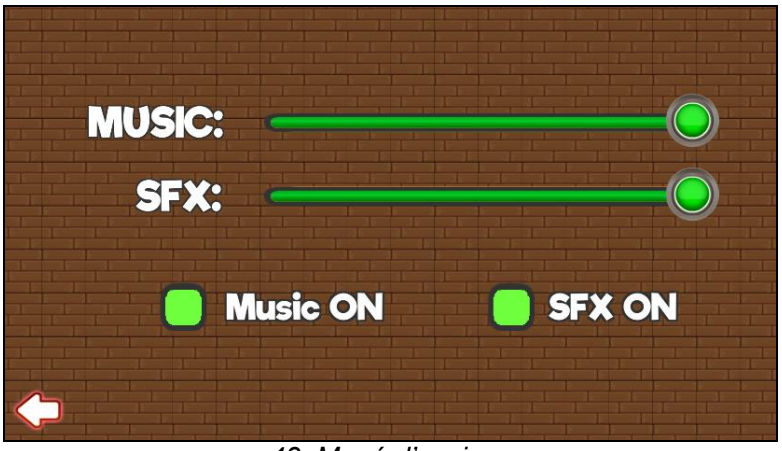
Fora de la partida es poden apreciar 4 menús en el joc: la pantalla principal, el menú d'opcions, la botiga i el menú de selecció de nivells.

El menú principal consta del logo del joc, que s'ha creat exclusivament per aquest projecte, un botó principal que porta al menú de selecció de nivell i 3 botons petits a cada cantonada de la pantalla excepte la superior esquerra que serveixen per entrar a la botiga, entrar al menú d'opcions i tancar el joc.

42.Menú principal

Per al menú d'opcions s'ha fet servir una interfície lleugerament diferent ja que no s'ha emmarcat el contingut dins el requadre groc que hi ha a la resta de menús. A més, disposa d'un botó a la cantonada inferior esquerra per tornar al menú principal.

[8] https://assetstore.unity.com/packages/2d/gui/casual-game-gui-skin-67196 (25/04/2018)



43. Menú d'opcions

Cal esmentar que els elements d'aquest menú estan connectats a un **Script** anomenat **SettingsMenu** que s'encarrega comunicar els valors dels elements amb les variables de **SettingsAndProgress** que guarden les preferències de les opcions, d'aquesta manera, quan el jugador modifica els valors en aquest menú, es guarden a la persistència.

```
public void SetMusicVolume(float volume)
{
    if (SettingsAndProgress.Instance.music) {
        masterMixer.SetFloat("musicVol", volume);
    } else
    {
        masterMixer.SetFloat("musicVol", -80f);
    }
    SettingsAndProgress.Instance.musicLevel = volume;
}

public void SetSFXVolume(float volume)
{
    if (SettingsAndProgress.Instance.effects)
    {
        masterMixer.SetFloat("sfxVol", volume);
    }
    else
    {
        masterMixer.SetFloat("sfxVol", -80f);
    }
    SettingsAndProgress.Instance.sfxLevel = volume;
}
```

44. Funcions que controlen el volum de so i connecta el so de les opcions amb el controlador de persistència

Pel que fa a la botiga, es tracta del menú on es poden desbloquejar les millores de la pistola. Les imatges que representen cada millora s'han extret de Google Images, però no es disposa de les URL exactes. Aquest menú té mostra la quantitat de monedes totals a la part superior dreta de la pantalla, així com el botó per tornar al menú principal amb el botó de la cantonada inferior esquerra.

A l'interior de la "finestra" groga es troben els elements a desbloquejar. Com que hi ha varis i l'espai és limitat, s'ha decidit simular la llista d'elements com si fossin pàgines que es van passant. D'aquesta manera, el menú disposa de fletxes laterals, que apareixen quan és necessari, per anar navegant entre les millores.

45. Botiga de millores

Cada millora s'encapsula en un panel que conté el nom de la millora, una imatge, el preu que costa desbloquejar-la (si no ho està ja), una descripció curta i un botó per desbloquejar-la. Si es disposa de suficients monedes per desbloquejar-la, el preu apareix en verd. Si la millora ja està desbloquejada, en comptes d'un botó apareixerà un text en verd indicant-ho i, en cas de no estar disponible per desbloquejar, un text que ho indica en color vermell.

Per aconseguir l'efecte de llista i d'anar passant pàgines s'ha creat una llista d'objectes de joc, que són els panells corresponents a cada millora. Un cop creada la llista, els botons que fan de fletxa s'encarreguen d'activar o desactivar els panells segons quin toqui mostrar en aquell moment.

```
public void Right()
{
    FindObjectOfType<AudioManager>().Play("FlipPage");
    if (currentItem == (totalItems - 1)) return;
    if (currentItem == 0)
    {
        leftButton.gameObject.SetActive(true);
    }
    items[currentItem].SetActive(false);
    currentItem++;
    items[currentItem].SetActive(true);

    if (currentItem == (totalItems-1))
    {
        rightButton.gameObject.SetActive(false);
    }
}
```

```
public void Left()
{
    FindObjectOfType<AudioManager>().Play("FlipPage");
    if (currentItem == 0) return;
    if (currentItem == (totalItems - 1))
    {
        rightButton.gameObject.SetActive(true);
    }
    items[currentItem].SetActive(false);
    currentItem--;
    items[currentItem].SetActive(true);

    if (currentItem == 0)
    {
        leftButton.gameObject.SetActive(false);
    }
}
```

46. Funcions assignades als botons dret i esquerra, respectivament

A més de la lògica que controla la visibilitat dels panells, cada panell té una lògica interna que serveix per controlar els seus elements. En essència, controla el color del cost de la millora, si s'ha de mostrar o no, si s'ha de mostrar el botó de desbloquejar, si s'ha de mostrar el text que indica que ja s'ha desbloquejat o si s'ha de mostrar el text que indica que no es pot desbloquejar.

L'últim menú d'aquest apartat és el que mostra els diferents nivells per poder començar a jugar. Els nivells bloquejats es mostren amb un cadenat i no es poden prémer. Els nivells desbloquejats apareixen sense el cadenat i al clicar-los, s'executa la funció del controlador de menús que inicia el nivell corresponent. Com la resta, aquest menú també disposa del botó inferior esquerra per tornar al menú principal.

47. Menú per seleccionar nivell

Cal mencionar que al iniciar-se aquest menú, s'executa un procés que comprova quins nivells han d'estar disponibles per al jugador en funció del seu progrés en aquesta execució i en totes les anteriors. Així, recorre tots els botons, activant-los o desactivant-los segons hagin d'estar disponibles o no. Com es pot observar, el recorregut de botons està fixat a 10 ja que hi ha 10 botons, però per bones pràctiques de programació s'hauria d'utilitzar el número total de nivells, assegurant-se que hi ha tants botons com nivells.

```
void OnEnable()
{
    for (int i = 1; i <= 10; i++)
    {
        if (SettingsAndProgress.Instance.lastLevelUnlocked >= i && i <= SettingsAndProgress.Instance.totalLevels) {
            GameObject.FindGameObjectWithTag("LevelBtn" + i).GetComponent<Button>().interactable = true;
            GameObject.FindGameObjectWithTag("ImageLock" + i).SetActive(false);
        } else
        {
            GameObject.FindGameObjectWithTag("LevelBtn" + i).GetComponent<Button>().interactable = false;
        }
    }
}
```

48. Procés de tractament dels botons de seleccionar nivell

4.1.2 Dins de la partida

Com a menús disponibles durant la partida es troben el menú de pausa, el menú de game over, el menú de nivell completat i, si es tracta del primer nivell i cal mostrar-lo, el tutorial.

El menú de pausa es mostra quan el jugador prem sobre el botó de les barres paral·leles que hi ha a la part superior esquerra de la pantalla. Quan s'obre aquest menú, el temps de la partida es paralitza, fent que no apareguin més focs, que no passi el temps i que no es permeti el moviment ni disparar. Entre les accions disponibles d'aquest menú es troben continuar la partida, reiniciar el nivell o tornar al menú principal.

49. Menú de pausa

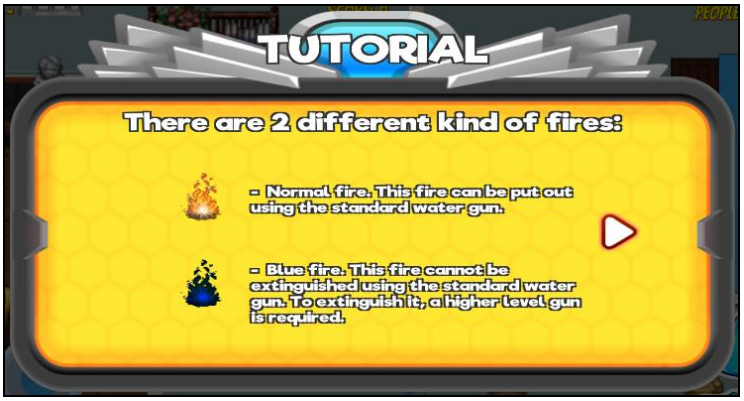
El menú de game over es mostra quan el jugador ha perdut tota la vida o quan s'esgota el temps. En aquest menú només hi ha dos opcions disponibles: reiniciar el nivell o tornar al menú principal.

50. Menú de Game Over

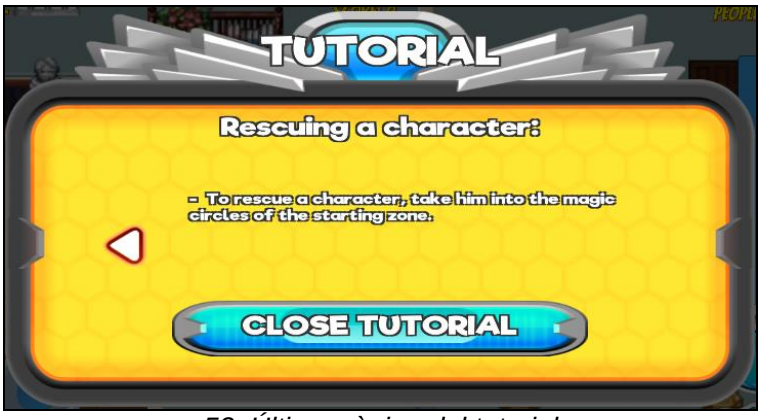
Al aconseguir salvar al PNJ de cada nivell, es mostrarà el menú de nivell completat, felicitant al jugador i mostrant la puntuació que ha aconseguit. A més, si ha superat el nivell amb un rècord, també ho mostrarà. Aquest menú ofereix les opcions de jugar al següent nivell, reiniciar el nivell (per si vol provar d'aconseguir millor puntuació) o tornar al menú principal.

51. Menú de nivell completat

Per últim, si cal mostrar tutorial es mostrarà un menú semblant a la botiga amb la informació més rellevant de les mecàniques del joc. D'aquesta manera, el jugador podrà passar per vàries pàgines d'explicacions curtes abans de començar el nivell 1. És obligatori passar per totes les pàgines per tal de poder tancar el tutorial i començar el nivell.



52. Primera pàgina del tutorial



53. Última pàgina del tutorial

4.2 UI durant la partida

Durant la partida, hi ha elements gràfics que acompanyen al jugador i sempre es mostren per pantalla. El primer és el panel d'estadístiques que es mostra a tota la part superior. Per la text d'aquest panel s'ha fet servir **TextMeshPro**[9], que és un altre asset gratuït de l'**Asset Store** d'**Unity**.

Aquest panel serveix per mostrar la informació estadística del nivell referent al temps restant, la puntuació actual i el número de persones salvades, respectivament d'esquerra a dreta. A més, també apareixen imatges de les monedes que es van recollint durant el nivell. Seria bona idea mostrar les imatges de les monedes amb color gris com si estiguessin desactivades mentre el jugador no les reculli, d'aquesta manera el jugador podria saber exactament quantes monedes té aquell nivell.

El següent element gràfic que es pot trobar és el text que apareix al mig al començar la partida, informant de quantes persones s'han de salvar per aquell nivell. Per crear l'efecte d'aquest text s'ha consultat el video de **Youtube** "**Text Mesh Pro – Text Reveal Effect**[10]**"** i s'ha invertit per aconseguir l'efecte contrari.

A la part dreta de la pantalla hi ha el panell relacionat amb les mecàniques del joc. Els elements que engloba són la barra que indica la capacitat restant de l'aigua (que no és més que un **Slider** adaptat sense el "botó" per moure'l) i els dos botons que serveixen per disparar i iniciar el salvament. Les imatges d'aquests botons s'han extret de l'asset gratuït **Mobile Aqua GUI**[11] de la **Asset Store** d'**Unity**.

[9] https://assetstore.unity.com/packages/essentials/beta-projects/textmesh-pro-84126 (15/03/2018)
[10] https://www.youtube.com/watch?v=U85gbZY6oo8 (06/03/2018)
[11] https://assetstore.unity.com/packages/2d/gui/icons/mobile-aqua-gui-28635 (18/04/2018)

El penúltim element de la interfície gràfica de la partida és el Joystick. Com ja s'ha indicat prèviament (pàgina 13), per aquest element s'ha utilitzat un altre asset gratuït que ja venia totalment funcional i preparat per inserir-lo a l'escenari de joc i utilitzar-lo sense cap altre configuració apart de consultar els valors per al moviment.

L'últim element de la interfície gràfica és la barra de vida del jugador. Es tracta d'un altre **Slider** sense el botó d'ajustar que es va actualitzant segons la vida que li queda al jugador. Per tal de mostrar que és la vida del jugador, s'ha ancorat a aquest mitjançant un **Script** que s'ha consultat a la resposta de l'usuari **DMGregory**[12] a una consulta d'una web d'ajuda similar a **StackOverflow** anomenada **StackExchange**.

Posteriorment, i ja que el jugador sempre està al centre de la càmera, va sorgir l'alternativa d'ancorar la barra de vida a una posició adient en comptes de fer que actualitzi la seva posició cada cop que canvia la posició del jugador. No obstant, com que tal com estava funcionava correctament, es va decidir deixar-ho com a canvi futur en cas que ajudés a optimitzar el rendiment del joc.

54. Interfície gràfica durant la partida

4.3 Gràfics

Pel que fa als gràfics dels nivells, s'han utilitzat una sèrie de sprites transformats en **Tilesets** per poder treballar amb els **Tilemaps**. A la Bibliografia s'adjuntaran les URL d'on s'han descarregat. A més, també s'adjuntaran totes aquelles URL d'on s'han descarregar tota la resta de textures i tots els sons del joc que no s'han adjuntat com a referències dins de les mateixes pàgines quan ha sigut necessari.

[12] https://gamedev.stackexchange.com/questions/131666/anchor-a-health-bar-to-a-character-enemy (14/04/2018)

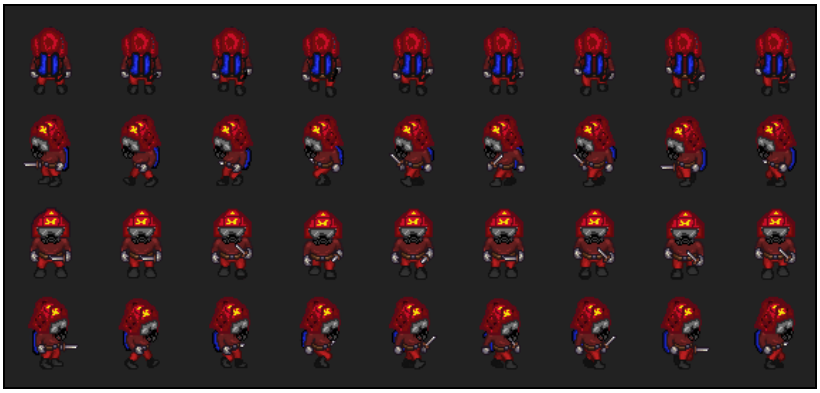
*55. Exemple de **Tileset** utilitzat per la decoració*

4.4 Animacions

4.4.1 Jugador / PNJ

Per els gràfics del jugador i del PNJ s'ha fet servir una web[13] amb una eina extremadament útil que permet crear personatges amb certa personalització i generar els **Sprites** d'animació corresponents.

Cal destacar que el PNJ a salvar sempre és el mateix model directament extret d'aquesta web. El jugador, en canvi, té elements afegits creats manualment amb **Photoshop** única i exclusivament per aquest projecte. Aquests elements són el casc i la motxilla de bombones d'aigua que porta a l'esquena.

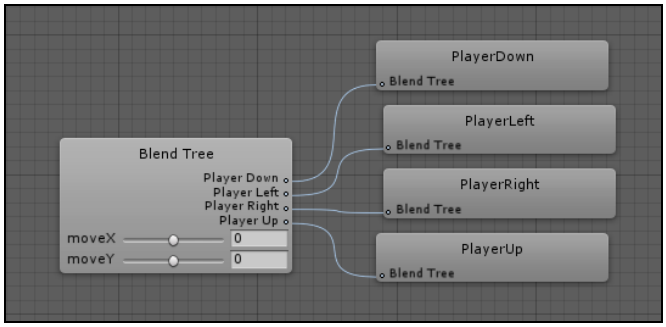


*56. **Sprite** de moviment del jugador*

Per crear l'animació s'ha fet servir el component **Animator** que ofereix **Unity**. A més amb l'ajuda del video "**Unity RPG Tutorial #4 – Blend Tree Animation**[14]" s'ha creat tota l'animació de

[13] http://gaurav.munjal.us/Universal-LPC-Spritesheet-Character-Generator/ (21/05/2018)
[14] https://www.youtube.com/watch?v=M1bEtHOE8z8 (06/03/2018)

moviment del jugador i el PNJ, tant l'animació de caminar com les transicions entre moviment i estat **"idle**", que és el nom col·loquial que se li dóna a un personatge quan està quiet sense fer res.



*57. **Blent Tree** per al moviment del jugador*

4.4.2 Foc

Per l'animació tant del foc normal com del foc blau s'ha fet servir el mateix **Sprite** en bucle infinit, descarregat de la web **OpenGameArt.org**[15]. De fet, el foc blau utilitza el mateix **Sprite** però amb canvi de color.

*58. **Sprite** utilitzat per al foc*

4.4.3 Aigua

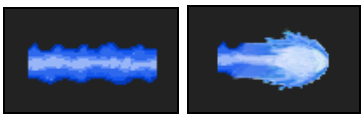
L'aigua va ser un dels elements que més difícil semblava que seria de crear, ja que després de buscar molts recursos per internet per veure si hi havia algun que es pogués utilitzar no es va trobar res.

Al final, es va decidir intentar crear l'aigua píxel a píxel amb l'eina **Photoshop**. El resultat va ser molt millor de l'esperat inicialment. Es van crear 2 **Sprites**, el primer simula el cos del raig i, el segon, el cap. D'aquesta manera no hi ha problema al crear molts elements d'aigua amb el mateix **Sprite**, ja que una de les majors preocupacions era que no quedés bé aquest efecte.

Quan una instància d'aigua està apunt de desaparèixer, es canvia el seu **Sprite** de cos a cap per crear l'efecte efecte de raig. En

[15] https://opengameart.org/content/camp-fire-animation-for-rpgs-finished (13/03/2018)

aquest aspecte, realment no hi intervenen animacions, ja que es juga amb la creació i el moviment dels objectes a través dels **Scripts** del propi joc. Per a que encaixi amb la situació del jugador quan es crea, es juga amb la seva rotació.



59. Sprites creats per l'aigua

4.4.4 Moneda

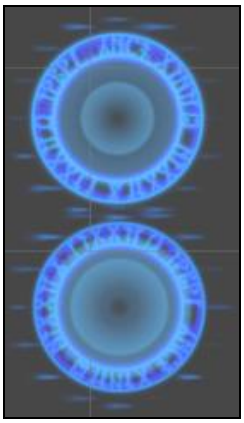
Pel que fa a la moneda, al igual que el foc, s'ha fet servir un **Sprite** descarregada de la mateixa web[16]. Aquesta animació només consta de la rotació de la moneda sobre ella mateixa. Com s'ha esmentat anteriorment, el moviment vertical es controla via **Script**.

*60. **Sprite** utilitzat per la moneda*

4.4.5 Partícules

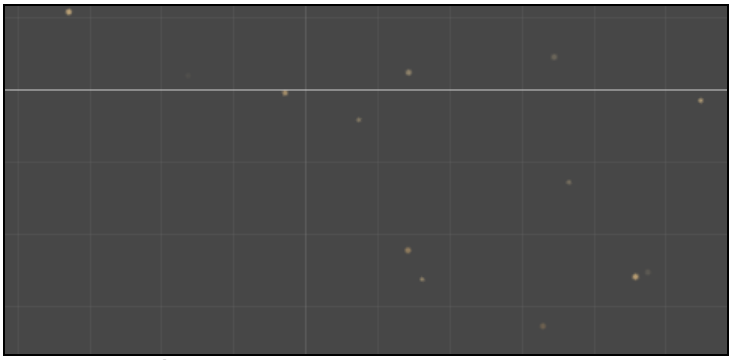
Les partícules s'han utilitzat per indicar l'entrada del nivell i, al mateix temps, la zona de salvament del PNJ. Aquestes partícules s'han extret de l'asset gratuït **Cartoon FX Free**[17] de la **Asset Store** d'**Unity**. Com que és un paquet que té moltes partícules, esmentar que s'ha utilitzat l'anomenada "**CFX3_MagicAura_D_Runic**" que es pot trobar a la ruta "**JMO Assets\Cartoon Fx\CFX3 Prefabs\ Magic Misc**". A més, s'ha ajustat la mida i s'ha canviat el color per blau per a que encaixi millor amb la temàtica del joc.

[16] https://opengameart.org/content/rotating-coin (13/03/2018)
[17] https://assetstore.unity.com/packages/vfx/particles/cartoon-fx-free-109565 (22/05/2018)



61. Partícules que marquen l'entrada/sortida

Per últim, mentre es cercava alguna partícula que es pogués utilitzar per marcar la zona de salvament, es va trobar dins del mateix asset gratuït una altra partícula que encaixava perfectament amb el joc. Es tracta de les partícules que simulen espurnes de foc que es van elevant lentament fins desaparèixer. Es va considerar que al afegir aquestes partícules s'aconseguiria una millor immersió dins de cada nivell.



62. Partícules utilitzades per simular les espurnes

Indicar que aquesta partícula es pot trobar a la ruta "**JMO Assets\Cartoon Fx\CFX3 Prefabs\Fire**" i s'anomena "**CFX3_Flying_Ember_Upward**".

5. Conclusions

Estic molt content amb el resultat aconseguit en trets generals. Soc conscient que hi ha parts que es poden millorar i inclús ja tinc moltes idees per aplicar si decideixo continuar el desenvolupament en un futur, però crec que per no tenir experiència prèvia en desenvolupament de videojocs el resultat és prou bo.

Al llarg d'aquest projecte m'he iniciat amb Unity i he après a treballar amb la seva metodologia a nivell principiant. Al ser un motor tan extens ha sigut possible trobar molta informació al respecte per fer qualsevol cosa, i això m'ha permès aprendre vàries maneres de fer la mateixa cosa.

Hi ha hagut parts del projecte més difícils que altres, com per exemple el moviment del jugador o el sistema de seguiment del PNJ. Per poder implementar-les, he hagut de pensar en diverses aproximacions al problema que tenia fins aconseguir un resultat satisfactori, no tot el que provava funcionava a la primera, però això també és útil per guanyar experiència, ja que a vegades s'aprèn més dels errors que dels èxits.

Ja ho havia vist al llarg de la tota la carrera, però he tornat a aprendre que una bona planificació és clau alhora d'assolir els objectius del projecte. Gràcies als requeriments de la PAC1 que en certa manera "obligaven" a fer el plantejament i la planificació, ha sigut més senzill poder fer el seguiment del projecte, especialment amb el diagrama de Gantzz.

Els objectius plantejats inicialment per al projecte s'han aconseguit tots. Hi ha hagut algun petit canvi al llarg del projecte per temes de funcionalitat, com el moviment del jugador que inicialment era només en 4 direccions i s'ha acabat implementant per 8, però a trets generals s'ha pogut implementar tot el que s'havia plantejat al principi i, a més , s'han pogut afegir gairebé tots els suggeriments del consultor i familiars/amics.

És cert, però, que no tot ha anat perfecte. Hi ha hagut coses que m'hagués agradat fer millor però que per manca de temps no he pogut. Exemples d'això són el tutorial que, tot i que queda ben encaixat al que és tota l'estructura de joc, m'hagués agradat poder implementar alguna cosa més complexa. Tampoc he pogut fer més nivells dels 5 que estan al resultat final, i he de reconèixer que inicialment pensava que en podria fer algun més.

M'he quedat amb les ganes també de poder implementar un suggeriment del consultor sobre el número de persones a salvar per cada nivell. He intentat afegir-ho fins a l'últim moment, però tot i que ja hi havia parts de la implementació preparades per aquest fet, hi havia altres que no ho estaven i no he tingut temps de posar-me a adaptar-les.

Això sí, queda pendent per una futura implementació com a primer objectiu de la llista.

Una altra millora que m'he plantejat desenvolupar en un futur és afegir la funcionalitat d'internet al joc per permetre als jugadors crear usuaris i enregistrar els seus rècords en algun servidor. D'aquesta manera es podrien mostrar rànquings i crearia un ambient de competitivitat sana que empentaria als jugadors a continua jugant els nivells que ja han completat per intentar aconseguir més puntuació.

Per últim, una reflexió sobre el desenvolupament de videojocs amb un equip d'una única persona. Després de treballar en aquest projecte, m'he adonat que és molt difícil obtenir un producte d'alta qualitat quan es treballa sol. Això és degut a que en un videojoc intervenen moltes branques diferents de diverses especialitats, com pot ser el so, els dissenys gràfics, la implementació. Inclús dins de la implementació pot haver grans variacions. Per exemple, per aplicar la millora d'internet seria recomanable comptar amb un expert en xarxes per tal de tenir la millor infraestructura possible per als servidors, però al mateix temps l'expert en xarxes no té perquè ser també expert en programació dels processos interns de les mecàniques del joc.

Així, com a conclusió puc extreure que per desenvolupar un videojoc d'alta qualitat és recomanable comptar amb com a mínim un equip de 3-4 persones: un programador, un dissenyador gràfic, un compositor de so i un guionista/dissenyador de nivells que domini els aspectes que ha de tenir un joc.

Sense més dilació, m'acomiado dient que no me'n penedeixo d'haver escollit fer el TFG sobre videojocs i estic molt agraït per poder haver tingut aquesta experiència.

6. Glossari

PNJ – Personatge no jugador.
Singleton – Classe que només pot instanciada un cop en un moment donat.
Script – Arxiu de codi associat a un objecte de joc.
Collider – Delimitació de col·lisió d'un objecte.
Tilemap – Mapa de Tiles.
Tile – Cel·la d'un Tilemap que s'omple amb una textura gràfica.
Tileset – Conjunt de Tiles formats a partir d'un Sprite.
Sprite – Una imatge, generalment en format PNG.
Rigidbody2D – Component d'Unity que assigna propietats físiques a un objecte.
Raycast – Funció per detectar col·lisions en una àrea determinada

7. Web grafia

En aquest apartat s'afegiran les URL que no s'han afegit al llarg del document.

Textures dels Tilemaps:

- Decoració: https://nicnubill.deviantart.com/art/Gothic-Furniture-404100352 (13/03/2018)
- Terra, parets i altra decoració: https://sharmclucas.deviantart.com/art/LPC-Interior-Mockup-336632539 (20/04/2018)

Pistes d'audio: (16/05/2018)

- https://assetstore.unity.com/packages/audio/ts-sounds-34453
- https://assetstore.unity.com/packages/audio/sound-fx/free-casual-game-sfx-pack-54116
- https://assetstore.unity.com/packages/audio/carefree-sound-bundle-65562
- https://www.youtube.com/watch?v=qfx6yf8pux4
- https://www.youtube.com/watch?v=XEm-anELm10
- http://soundbible.com/2032-Water.html
- https://www.youtube.com/watch?v=mz9ftphTWTM

Alguns dels efectes de so s'han retallat amb l'eina Audacity per crear una millor transició del so en bucle.

8. Annexos

Tutorials de Scripting: https://unity3d.com/es/learn/tutorials/s/scripting
Tutorials 2D: https://unity3d.com/es/learn/tutorials/s/2d-game-creation
Documentació d'Unity: https://docs.unity3d.com/Manual/index.html
Scripting API: https://docs.unity3d.com/ScriptReference/index.html
Canal de youtube dedicat a tutorials de creació de jocs principalment utilitzant Unity: https://www.youtube.com/channel/UCYbK_tjZ2OrIZFBvU6CCMiA

Com a últim Annex, per si fos d'interès, adjunto una URL a una carpeta oberta de mega.nz on es poden trobar moltes de les versions que s'han anat creant del joc disponibles per descarregar: https://mega.nz/#F!9TAClAjA!fdx8CEZuYtknf3rz6VRjbg