Universitat de Girona
Escola Politècnica Superior

Treball final de grau

Estudi: Grau en Disseny i Desenvolupament de Videojocs

Títol: Disseny i desenvolupament d'un joc 2D per mòbil

Document: Memòria

Alumne: Antoni Druguet Solà

Tutor: Antonio Rodríguez Benítez

Departament: Informàtica, Matemàtica Aplicada i Estadística

Àrea: Llenguatges i sistemes informàtics

Convocatòria (mes/any) 09/23

Índex

Índex de Figures

1. Introducció i objectius

1.1 Introducció

Actualment, vivim en un món on tothom porta un dispositiu mòbil a sobre. Això fa que el nostre medi tant per comunicar-nos, estar al dia de les notícies i entretenir-nos sigui el nostre telèfon mòbil. Però en l'actualitat el món està hiperconnectat, i això també ha afectat en la indústria de l'entreteniment i els videojocs. Fent que avui dia la major part dels videojocs necessitin connexió a internet per tal de poder jugar. Encara que aquesta no aporti res a la jugabilitat ni es necessiti en cap aspecte per l'usuari final. A part de la necessitat de connexió, però, els jocs de mòbil també han evolucionat. On abans un simple joc de tocar la pantalla o fer certs moviments amb el dispositiu era tot el necessari per tenir a l'usuari entretingut durant hores, ara ja no funciona. Ara els usuaris de mòbil també volen gaudir d'un joc amb certa profunditat en la història, les mecàniques o el contingut en general que pot proporcionar un videojoc.
Per aquest motiu l'objectiu és crear un joc amb mecàniques interessant i que es necessiti certa estratègia i presa de decisions per tal d'avançar. I que no necessiti connexió a internet per tal d'assegurar que l'experiència no és interrompuda o poc satisfactòria a causa d'una mala connexió a la xarxa.

1.2 Motivacions

Des que sóc petit que jugo a videojocs de molts gèneres diferents. I un dels moments on més jugava era quan estava de viatge cap a un lloc, o tenia o alguna estona que no sabia molt bé què fer. Tenir jocs senzills i entretinguts per poder jugar en aquestes ocasions sempre m'ha ajudat que el temps passés molt més ràpid i no m'avorrís o estrasses per no saber què fer. Per això mateix em va agradar la idea de fer un joc casual amb el qual passar l'estona sense res més que necessitat de tenir el teu dispositiu mòbil i sense tampoc dependre de la connexió a internet que moltes vegades en viatjar de petit jo no en tenia.

He triat un gènere 2D "top-down" perquè el 2D em dóna més llibertat per fer i trobar recursos, perquè no tinc un perfil artístic i això fa que el tema del modelatge en 3D se'm compliqui molt, i és un "top-down" perquè pel joc que volia fer era la perspectiva que més encaixava pel que fa a la jugabilitat.

El projecte es tracta d'un joc on has de defensar l'edifici principal del poblat mentre et van arribant onades d'enemics que cada vegada són més nombrosos i més forts. Es podria definir com una variant de tower defense, on en comptes de col·locar torres, és el propi jugador amb el seu personatge qui ha d'anar defensant la base utilitzant certes defenses com les muralles que té o acompanyants soldats que pot reclutar.

La motivació professional en aquest projecte és aprofundir en certs aspectes dels quals sentim a parlar a la carrera. Com per exemples el "pathfinding" dels enemics en buscar camins utilitzant algorismes de cerca. I també aprofundir en certs aspectes com la generació d'onades d'enemics o el sistema d'animacions 2D. Juntament amb l'estratègia i presa de decisions de com anar pujant de nivell el nostre personatge, reclutar ajudants o millorar les defenses de la ciutat i com implementar correctament totes aquestes mecàniques.

1.3 Propòsit i objectius del projecte

El nostre propòsit consisteix en construir un joc que compleixi les següents característiques:

1. Joc per dispositiu mòbil amb controls senzills.
2. Un sistema d'onades d'enemics que vagi fent el joc cada vegada més complicat amb més enemics i més forts.
3. Enemics de diferent tipus que donin jugabilitat i que intentin atacar diferents objectius.
4. Una economia interna de nivells de millores pel personatge jugable i l'entorn que faci plantejar diferents estratègies al jugador.

1.4 Distribució de tasques

Normalment, un videojoc sol ser desenvolupat per un equip amb diverses persones per a cada especialitat i sector, com poden ser el disseny del joc, l'apartat artístic, la programació, el so, etc. En aquest projecte tots aquests camps seran portats per una sola persona, qui haurà de planificar, desenvolupar i fer proves en tots els camps. Així doncs, la distribució percentual de quant valor i temps es dedicarà a cada apartat són:

Estètica	20%
Narrativa	5%
Mecàniques	30%
Tecnologia	45%

Pel que fa a l'apartat estètic, tot i que sol ser un dels apartats més importants des de fa molt de temps, ja que els jugadors cada vegada exigeixen millors gràfics, menús més cuidats i més interactius, etc. No és un dels camps en els quals tingui molta experiència i habilitat. Per aquest motiu, farem usos de molts tutorials i també agafarem recursos ja fets per tal de poder garantir el millor resultat possible, sense haver de dependre exclusivament de recursos de tercers i fent tot el possible dins la meva pròpia habilitat.

Per aquest projecte la part narrativa no ha de ser molt enrevessada, ja que és un joc casual per poder passar l'estona i el que ens interessa és tenir una jugabilitat interessant i que ens animi a millorar la nostra anterior puntuació. Per això tindrà una simple introducció de la història per situar al jugador.

L'apartat de mecàniques és un dels més importants per aquest projecte. Ja que és l'apartat que ha d'aportar més interès pel jugador per tal que vulgui recórrer a aquest joc quan vulgui relaxar-se una estona i superar-se. Per tant, hem de fer unes mecàniques interessants perquè sigui un repte i una satisfacció per igual.

Finalment, l'apartat tecnològic és el més important, ja que la programació és l'aspecte protagonista de la interactivitat i el funcionament. També és l'aspecte que més m'agrada i al que sempre intento que sigui la millor versió possible dins les meves capacitats, fent que pugui repetir diverses vegades com fa el joc un aspecte per tal de garantir que sigui el millor possible. Ja sigui perquè sigui el més òptim, el que millor encaixa o el que millor resultat dóna.

2. Estudi de viabilitat

Per realitzar aquest projecte s'utilitzaran eines i recursos que no suposessin cap cost, sempre respectant els drets d'autor i de propietat intel·lectual. Usant material d'ús lliure i/o lliure distribució llegint, entenent i assegurant cada llicència quan s'hagi d'utilitzar algun producte que no sigui de creació pròpia.

2.1 Recursos necessaris i viabilitat

2.1.1 Recursos tècnics

Aquest projecte es portarà a terme amb el hardware següent:
Ordinador portàtil model **MSI Katana GF66**:

- Targeta gràfica (GPU): NVIDIA GeForce RTX 3060 6GB GDDR6
- Processador (CPU): Intel® Core™ i7-12650H 4.70GHz
- Mèmoria: 16GB DDR4 3200MHz

2.1.2 Recursos humans

Cada videojoc desenvolupat necessita un equip format per individus amb diferents rols i especialitats que solen definir-se amb diversos perfils:

- **Dissenyador de joc:** l'encarregat de definir el joc, els objectius, narrativa i qualsevol element que ha d'aparèixer.
- **Programador:** encarregat de la implementació del joc, desenvolupament del codi i dels aspectes més tècnics del projecte.
- **Artista:** l'encarregat de dissenyar i dibuixar els components visuals del joc.
- **Dissenyador de so:** encarregat de dissenyar la música i els efectes de so del projecte.

Depenent del joc poden haver-hi molts més especialistes i rols, segons el producte i la capacitat de l'empresa.

En aquest cas, molts d'aquests rols els farà una única persona, o s'utilitzaran recursos d'internet.

2.1.3 Viabilitat econòmica

En aquest cas no s'haurà d'invertir en recursos tecnològics ni econòmics extres per la realització d'aquest projecte perquè tot el programari utilitzat és gratuït, s'ha utilitzat la seva versió gratuïta o ja es disposa d'ell. Tot i això, a continuació farem un pressupost hipotètic en cas que es necessités tot el material de zero.

RECURS	COST
Maquinària amb sistema operatiu	~1200 €
Software artístic	0 €
Motor de videojoc	0 €
TOTAL	~1200 €

Respecte a recursos humans, si tenim en compte el preu per hora dels diferents especialistes públics a webs com (www.payscale.com) veurem que els preus són:

- **Dissenyador de joc:** ~16 €/h
- **Programador:** ~15-20 €/h
- **Artista:** ~13-15 €/h
- **Dissenyador de so:** ~12 €/h

Per obtenir una xifra aproximada s'hauria de multiplicar el preu per hora de cada especialitat segons les hores necessàries per a cada tasca del seu camp. I si nosaltres no som experts com a Project Manager potser hauríem de contractar a algú que ens portés el projecte per tal d'assignar quantes hores es necessita per cada aspecte necessari.

Com ja s'ha comentat abans, com que nosaltres no necessitarem res d'això. El cost és de 0€.

2.2 Estudi de mercat

Un cop s'ha analitzat la viabilitat tècnica i econòmica del projecte, el següent pas és fer un estudi del mercat actual i saber quins productes hi ha actualment semblants al nostre projecte, quines característiques tenen i quins són els seus punts forts.

2.2.1 Estat de l’art

S’ha utilitzat el motor de recerca de Google per cercar diferents jocs que tinguin similituds amb el nostre projecte, tot i que en el gènere dels Towers Defense hi ha molts productes, farem una cerca dels que tinguin mecàniques similars al nostre projecte per tal de veure quines idees i mecàniques aporten i quins són els seus aspectes més rellevants.

- **Orcs Must Die! (2011):** És una saga de jocs de tower defense en 3D on el jugador controla un heroi que pot construir trampes i utilitzar els seus atacs per derrotar els enemics. El jugador ha de mantenir els orcs fora del seu castell i impedir que trenquin la porta d'entrada. Al llarg de la saga han anat millorant gràficament i hi han afegit coses. Com la varietat en els herois, més trampes, etc. Tot i que els enemics només tenen un camí a seguir, el mapa té un aspecte molt obert i no dóna la sensació que no hi hagi llibertat de moviment.

Figura 1: Portada Orcs Must Die!

Figura 2: Captura Orcs Must Die!

- **Kingdom Rush (2011):** És una saga de jocs de tower defense en 2D on el jugador controla un rei que ha de defensar el seu regne dels orcs. El jugador ha de construir torres i utilitzar els seus recursos per derrotar les onades d'enemics. Els enemics segueixen un camí molt marcat i no tenen altres estratègies o camins diferents per anar. Té molta varietat varietat d'enemics i moltes torres diferents que el jugador pot fer servir per la seva defensa.

Figura 3: Portada Kingdom Rush Frontiers

Figura 4: Captura Kingdom Rush Frontiers

- **They are Billions (2017)** Quan va sortir en primícia, era un joc on havies de construir una base i aguantar diferents onades de zombis cada cop més grans. Més endavant hi van afegir nous zombis, més mapes i el mode història/campanya. Els enemics, tot i que van en grups grans durant les onades i solen fixar-se un camí i un objectiu, tenen molta llibertat de moviment i s'adapten a les defenses del jugador. El jugador té un control més estil RTS (Estratègia en Temps Real) i, per tant, controla tota la base i no un sol personatge.

Figura 5: Portada They Are Billions

Figura 6: Captura They Are Billions

2.3 Públic objectiu

El jugador ideal per aquest joc és la persona aficionada als jocs estil Tower Defense, ja que té el component de defensar un objectiu de diferents onades. Però alhora, és un joc més frenètic que els típics d'aquest gènere, per tant, han de ser jugadors que vulguin estar constantment atents per tal que cap enemic s'escapoleixi. Tanmateix, aquells qui els hi agradin les temàtiques medievals de fantasia es veuran atrets per l'ambient del joc. Per acabar, tot i que els Towers Defense poden estar orientats per totes les edats, pels controls, els objectius i reptes que tindrà el jugador, l'edat objectiu seria per a majors de 7 anys perquè no se'ls hi faci massa complicat i es frustrin.

3. Planificació

3.1. Metodologia de treball

Des d'octubre del 2022 s'ha iniciat la planificació del projecte.
S'ha organitzat el projecte de tal manera que cada aspecte passes pels següents processos: Disseny, Recerca i estudi, Desenvolupament, Proves i Fet.

- Disseny: Abans de començar a desenvolupar qualsevol aspecte del joc es farà un disseny previ de com hauria de ser, tant a nivell artístic com quines accions, sons i efectes haurà de tenir.
- Recerca i estudi: Un cop decidit com s'ha de ser, es realitzarà una recerca per tal de veure què hi ha fet, quines solucions ha trobat la gent i possibles recomanacions de com fer certes coses.
- Desenvolupament: Un cop s'ha fet la recerca i estudi previ sobre l'aspecte a implementar, es començarà el seu desenvolupament pas per pas fins a aconseguir el resultat que es busca o, en cas que requereixi d'altres elements, un resultat aproximat que ha de fer aquella part fins a interaccionar amb una altra. Paral·lelament a això o en algunes ocasions abans de la programació, s'anirà desenvolupament la part artística de l'element, i buscant els efectes de so que siguin necessaris.
- Proves: Mentre es faci el desenvolupament de cada part, s'aniran fent proves periòdiques per tal de garantir que es va avançant correctament. I un cop acabat cada aspecte important, es farà una prova general tant d'aquell aspecte com de tots

el que hi hagin en aquell moment per tal de comprovar que el funcionament global sigui el correcte i esperat.

Cada una d'aquestes grans tasques estarà dividida en tasques o apartats més petits segons el seu aspecte: art, programació, sons, etc.
Per l'organització d'aquestes tasques s'utilitzarà l'eina web de *Trello* de la pàgina web Trello.com (Figura X).

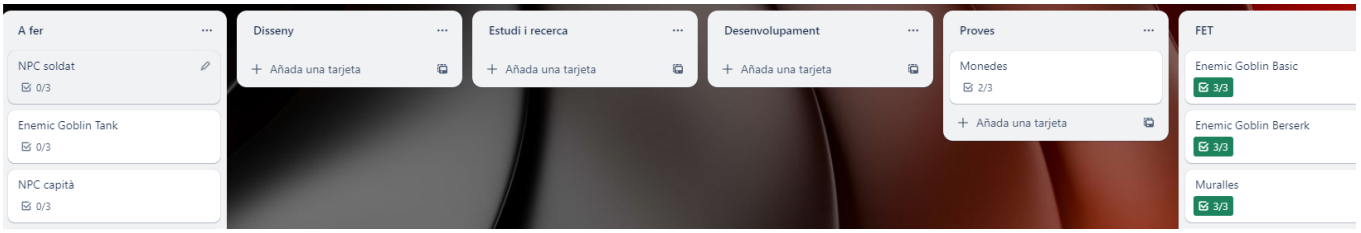


Figura 7: Captura de Trello

Cada setmana s'establiran uns objectius a aconseguir per la següent i s'anirà avançant a mesura que les tasques es vagin complint.

3.2 Diagrama de Gantt

	10/22				11/22				12/22				01/23			
Mecàniques jugador																
Art jugador																
Creació de mapes amb TileMap																
Controls jugador																
Murs destruïbles																
Algorisme A*																
Art enemics																
Mecàniques enemics																
Animacions 2D																
Menú principal																
Menú Game Over																
Mecànica final de partida																
Edificis de millores																
Edifici central (objectiu enemics)																
Millores																
Soldats aliats																
Re-jugabilitat																
Traducció																
Música i efectes de so																
Documentació																

Figura 8: Diagrama de Gantt

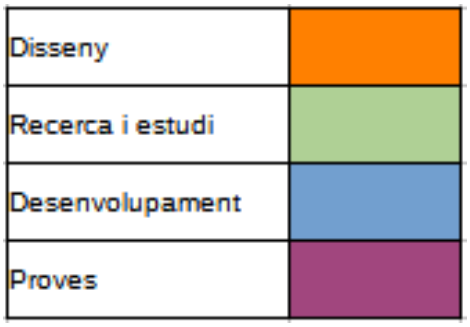


Figura 9: Llegenda diagrama de Gantt

3.3 Llista de tasques a desenvolupar

A continuació es presenta la llista de tasques totals a desenvolupar segons modalitat i prioritat:

🟥 Prioritat Alta 🟪Prioritat Mitja 🟨Prioritat Baixa

- **Disseny:**
    1. Disseny del jugador 🟥
    2. Disseny de mecàniques de jugador 🟥
    3. Disseny d'enemics 🟥
    4. Disseny del mapa 🟥
    5. Disseny del HUD i menús 🟥
    6. Disseny d'aliats 🟨
    7. Disseny de millores 🟥

- **Programació:**
    8. Desenvolupament mecàniques jugador 🟥
    9. Implementació animacions jugador 🟪
    10. Implementació sons de jugador 🟨
    11. Creació del mapa amb TileMap de Unity 🟥
    12. Controls del jugador 🟥
    13. Implementació de murs destruibles 🟪
    14. Implementació dels sons de murs destruibles 🟨
    15. Creació Algorisme A* 🟪
    16. Desenvolupament mecàniques enemics 🟥
    17. Implementació d'animacions enemics 🟪

18. Implementació sons enemics 🟨
19. Implementació menu principal 🟥
20. Implementació menú Game Over 🟥
21. Implementació final de partida 🟥
22. Implementació d'edificis interaccionales 🟪
23. Implementació de recursos 🟥
24. Implementació de millores de característiques 🟪
25. Implementació soldats 🟨
26. Implementació re-jugabilitat 🟨
27. Implementació traducció 🟨
28. Implementació músiques i altres efectes de so 🟨

- **Art:**
    29. Creació de sprites jugador 🟥
    30. Creació de Tilesets pel mapa 🟪
    31. Creació art de murs destruibles 🟪
    32. Creació sprites enemics 🟥
    33. Creació botons per menús 🟪
    34. Creació botons per millores 🟪
    35. Creació elements HUD 🟪
    36. Creació art edificis interaccionales 🟪
    37. Creació art de recursos (monedes) 🟪
    38. Creació sprites soldats 🟨

4. Marc de treball i conceptes previs

4.1 Eines de treball i programari

En aquest apartat s'analitzarà els programes que s'han utilitzat per al desenvolupament del projecte, explicant els motius de per què s'han utilitzat les escollides.

4.1.1 Motor de joc

El motor de joc és un dels elements més importants a l'hora de desenvolupar un videojoc. Ja que és l'arquitectura bàsica que garanteix certs aspectes com les físiques, renderitzat, animacions, sons, etc. Actualment, hi ha diversos motors al mercat que ens permeten

centrar-nos en el desenvolupament del joc proporcionant ells l'entorn i arquitectura necessària.

Figura 10: Logotip de Unity

S'ha escollit d'entre tots els possibles, Unity com a motor per desenvolupar el projecte.
Els motius principals són que ja tinc experiència amb aquest motor gràcies a la carrera i al grau superior que vaig fer abans d'aquesta. Un altre motiu de pes és el fet que facilita molt la creació de jocs per plataforma mòbil, i sobretot en 2D. Ja que té una interfície amigable en tot el desenvolupament, però destacant la facilitat de programar i veure com va quedant la interfície d'usuari.

4.1.2 Editor de codi

Com a editor de codi principal per escriure el codi del joc, s'utilitzarà Visual Studio, ja que és el que ve per defecte amb Unity i a més ens permet *"debugar"* el projecte en temps real i veure en tot moment per on passa el codi, parar el codi quan arriba a cert punt, parar el codi quan certa condició es compleixi, etc.

Figura 11: Logotip Visual Studio

4.1.3 Programari artístic

Per la creació i edició de l'apartat artístic es faran servir 2 programaris:

Pixelorama és un editor de píxel art lliure i de codi obert, creat amb el Godot Engine, per Orama Interactive. Permet la creació d'animacions píxel art, gràfics de joc, rajoles i qualsevol tipus d'art de píxels. El farem servir sobretot per la creació o modificació dels recursos necessaris per fer el joc.

Figura 12: Logotip Pixelorama

Krita és un programa professional de pintura digital, gratuït i fet amb codi lliure, ha estat creat per artistes. L'utilitzarem per a la creació de certs efectes que necessitin una quantitat de píxels molt més gran com per exemple els efectes de pantalla.

Figura 13: Logotip Krita

4.2 Conceptes previs

4.2.1 Conceptes de Unity

En aquest apartat explicarem alguns conceptes de Unity que es mencionaran més endavant quan parlem de la implementació del projecte.

4.2.1.1 Layers

Els layers són una eina que permet separar GameObjects en les vostres escenes. Podeu utilitzar capes a través de la interfície d'usuari i amb scripts per a editar com els GameObjects dins de la vostra escena interactuen entre si. Fent que 2 objectes amb col·lisionadors amb layers diferents s'ignorin i no col·lisionin entre ells.

4.2.1.2 Tags

Un Tag o etiqueta és una paraula de referència que es poden assignar a un o més GameObjects. Per exemple, es pot definir un Tag "Player" per el personatge controlat pel jugador i un Tag "Enemy" per els personatges no controlats pel jugador. O es podria definir els elements que el jugador pot recollir en una Escena amb un Tag "Recursos".

Els Tags ajuden a identificar els GameObjects per exemple quan hi ha col·lisions. I poder identificar amb què s'ha col·lisionat i tenir un comportament en concret si això passa.

4.2.1.3 Prefabs

Els prefabs actuen com a plantilla per tal de poder instanciar GameObjects amb certs valors i que si es fa un canvi en aquesta plantilla, tots els objectes creats a l'escena a partir d'aquest Prefab tinguin aquest canvi guardat.

4.2.1.4 ScriptableObjects

Un ScriptableObject és un contenidor de dades que es pot utilitzar per guardar grans quantitats de dades, independentment de les instàncies d'una classe. Un dels principals casos d'ús dels ScriptableObjects és reduir l'ús de memòria del projecte evitant còpies de valors. Això és útil si el vostre projecte té un Prefab que emmagatzema dades que no canvien. Així doncs, ajuden a guardar dades dels objectes de manera més eficient que els prefabs poden tenir moltes variacions d'aquests sense ocupar memòria.

4.2.1.5 FixedUpdate / Update

La diferència entre l'Update i FixedUpdate és que l'Update s'executa cada fotograma, mentre que el FixedUpdate s'executa a una velocitat específica definida a l'editor. De manera que si volem que un pas passi de manera constant sense dependre dels

fotogrames per segon, el FixedUpdate és el més adequat. Mentre que si volem que una cosa passi cada fotograma, s'ha de recórrer a l'Update.

4.2.1.6 Escenes

Les escenes de Unity són assets que contenen tots els assets que es facin servir a l'hora de fer un joc. Per exemple un joc simple podria estar tots dins una mateixa escena. Mentre que un projecte més complex podria estar separat per diferents escenes per exemple per separar el menú principal del joc, un nivell d'un altre, etc.

4.2.1.7 GameObjects

Un GameObject a Unity és la unitat més bàsica. Tots els objectes i components són GameObjects, però per si sols no poden fer res. Necessiten d'altres components com per exemple el componen Transform per col·locar-se en una posició. Així doncs, es fa servir la terminologia GameObject com a objecte dins del joc.

4.2.2 Algorismes de cerca de camins

Perquè els enemics tinguin una certa intel·ligència i busquin per on han d'atacar al seu objectiu és farà ús d'un algoritme de cerca de camins. Els Algorismes de cerca de camins ens permeten trobar el camí més curt entre 2 o més punts.
Alguns exemples són:

- ***A :** Aquest és un algorisme de cerca de camins heurístic que troba el camí més curt entre dos vèrtexs d'un graf. Es basa en la idea de calcular el cost estimat del camí fins a un vèrtex des del punt de partida, i seleccionar el vèrtex amb el cost estimat més baix en cada iteració.
- **Dijkstra:** Aquest és un algorisme de cerca de camins heurístic que troba el camí més curt entre dos vèrtexs d'un graf en un graf ponderat. Es basa en la idea de construir una llista de vèrtexs no explorats, i seleccionar el vèrtex amb el pes més baix en cada iteració.
- **Bellman-Ford:** Aquest és un algorisme de cerca de camins no heurístic que troba el camí més curt entre dos vèrtexs d'un graf en un graf ponderat. Es basa en la idea de calcular el cost del camí més curt des del punt de partida fins a cada vèrtex del graf.

- **Floyd-Warshall:** Aquest és un algorisme de cerca de camins no heurístic que troba el camí més curt entre qualsevol parell de vèrtexs d'un graf ponderat. Es basa en la idea de construir una matriu que mostra el cost del camí més curt entre qualsevol parell de vèrtexs del graf.

Després d'analitzar els diferents algorismes que hi ha i veure quins es solen utilitzar en la industria pel nostre cas, he decidit que la millor solució seria un algorisme de cerca de camins A*.

4.2.3 Terminologies

Durant aquest projecte també es faran servir certes terminologies que és important explicar.

- **Berserker:** Es fa servir popularment en el món dels videojocs per descriure a personatges que com més mal rebent més fort, ràpid o amb ràbia ataquen. El seu nom d'uns guerrers viking que lluitaven sense pràcticament armadura i sense protegir-se massa, atacant violentament sense miraments. També es diu que escumejaven per la boca.
- **Tank:** Terminologia que es fa servir en els videojocs per definir personatges amb molta defensa o especialitzats en aquesta, que costa molt fer-los mal i, per tant, debilitar-los.

5. Disseny del videojoc

En aquest apartat s'explicarà pas per pas el disseny de *Defenders of the valley o Defensors de la vall*. Que és el nom que li s'ha posat al joc.

5.1 Mecàniques

5.1.1 Espai de joc

El joc transcorre sempre en el mateix escenari, i el que va canviant són les onades d'enemics que arriben. S'ha decidit aquest plantejament perquè el jugador pugui memoritzar ràpidament on estan els punts claus a defensar, els punts on poder pujar de característiques i l'objectiu a defensar. I es pugui centrar en la defensa i l'estratègia a seguir per tal d'aconseguir-ho.

5.1.2 Característiques de mecàniques dels personatges

Tots els personatges i alguns objectes tenen unes característiques comunes que defineixen les mecàniques d'atac i defensa d'aquests. Aquestes característiques són:

- **Vida:** La vida que té un personatge o element. Quan arriba a 0 es mor o destrueix.
- **Atac:** És la característica que es fa servir per atacar. El jugador la podrà augmentar intercanviant monedes a la forja. Els enemics i soldats pujaran el seu atac segons el nivell de dificultat actual.
- **Resistència:** És la característica que fa resistir els atacs. El personatge i les barreres la podran millorar a la forja amb monedes. Els enemics i soldats pujaran la seva resistència segons avancen el nivell de dificultat del joc. Quan un personatge rebi un atac, es restarà la resistència a l'atac per saber la quantitat de mal rebut, que pot ser mitigat totalment si es té una resistència molt alta.
- **Velocitat:** És la velocitat en la que es mou un personatge.
- **Velocitat d'atac:** És la capacitat que té un personatge per atacar més ràpid.
- **Rang d'atac:** És el rang al que arriba l'atac del personatge.
- **Rang de detecció:** És el rang en el qual els enemics detecten els soldats o el jugador i deixen estar les estructures per anar a per ells.
- **Rang de prioritat:** És el rang en el qual els enemics determinen si anar a per un objectiu o un altre segons la diferencia entre distàncies.

5.1.3 Mecàniques del joc, reptes a afrontar i accions possibles

Al ser una variació d'un tower defense, l'objectiu principal del joc és **defensar l'edifici central** del poble dels enemics que intenent destrossar-lo. Però al controlar tu a un personatge, l'altre objectiu és **no quedar-se sense vida i morir**.

Per aconseguir això, el personatge del jugador tindrà a la seva disposició diverses accions com:

- **Moure's:** Moure's per tot l'espai per tal d'atacar i fugir dels enemics.
- **Atacar:** Atacar els enemics que intenten entrar a la base i destruir l'edifici central.

- **Comprar:** Pot comprar millores pel seu personatge, per estructures o reclutar soldats.

5.1.4 Objectes, recursos i interaccions que pot fer el jugador

El jugador podrà interaccionar amb els enemics, l'entorn i els recursos que els enemics deixen anar. A part, també podrà interaccionar amb diferents estructures que li permetran:

- **Forja:**
    - **Augmentar l'atac del personatge:** Augmenta l'atac del personatge per fer més mal als enemics.
    - **Augmentar la resistència del personatge:** Augmenta la resistència del personatge perquè els enemics li facin menys mal.
    - **Augmentar la resistència de les barreres:** Augmenta la resistència de les barreres que els enemics han d'eliminar per poder arribar fins l'edifici central.
    - **Restaurar barreres:** Restaura les barreres destruïdes o danyades.
- **Ajuntament:**
    - **Restaurar l'edifici central:** Restaura la vida de l'edifici central si ha estat danyat.
- **Caserna:**
    - **Curar-se:** Cura al personatge principal i recuperar vida.
    - **Reclutar ajudants soldats:** Recluta fins a 2 ajudants soldats que atacaran als enemics quan s'acostin a l'edifici central.
    - **Reclutar ajudat Capità:** Recluta a 1 capità molt poderós que atacara als enemics que s'acostin a l'edifici central.

5.1.5 Economia interna del joc

L'economia del joc es basarà en 2 aspectes principals, les monedes que aconsegueix el jugador al derrotar enemics i l'economia de les característiques dels personatges i com amb aquestes perden vida.

Monedes: Cada vegada que un enemic és derrotat deixa una quantitat de monedes que depèn del tipus d'enemic i de l'onada a la qual s'estigui actualment.

Vida: És la característica principal dels personatges i estructures. El jugador i les estructures la poden perdre i recuperar, els enemics i soldats només en poden perdre.

Atac: La característica dels personatges que els hi dóna la capacitat d'atacar.

Resistència: La característica dels personatges que els dóna la capacitat de defensar-se d'un atac.

Les diferents fonts de l'economia són:

Sorcerers: Els enemics deixen anar monedes quan els elimines.

Drains: Quan un personatge rep un atac superior a la seva resistència, li treu vida.

Traders: Els edificis de forja, edifici central i caserna, intercanvien monedes per diferents característiques com: atac, resistència i vida.

Converters: Amb les monedes es poden reclutar soldats i capitans per ajudar-nos a defensar.

5.1.5 Estudiar els diferents nivells del joc

En el nostre joc tenim un únic nivell, aquest està compost per un poblat al centre d'una prada verda. El poble està mig destrossat, ja que la primera gran onada d'enemics l'ha deixat a les últimes i ara el nostre personatge ha d'intentar aguantar amb les 4 barreres de fusta que han aconseguit construir de manera ràpida.

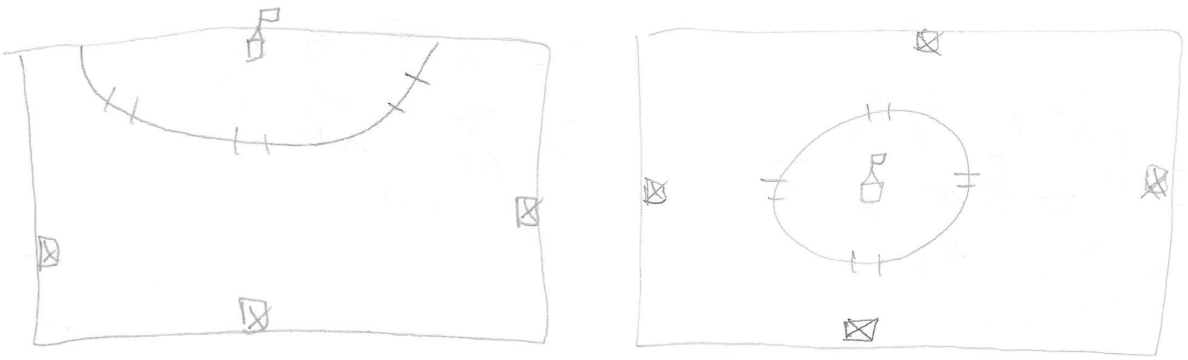


Figura 14: Esbossos del mapa

Aquest joc no estarà compost de diferents nivells, però si d'onades d'enemics.
Aquestes onades cada vegada seran amb enemics més nombrosos i amb millors característiques.
Per controlar aquestes onades tindrem un objecte encarregat de generar l'onada. A aquest, se li assigna un "pressupost" per cada onada i segon aquests pressupost crearà una onada de diferents enemics. Per exemple, si té un pressupost de 5, i el goblin bàsic val 1, el goblin berserker val 2 i el goblin tank val 3. Podria fer diferents composicions com:

- 5 goblins basics.
- 1 goblin tank i 2 goblins basics
- 2 goblins berserker i 1 goblins basics
- 1 goblin d'atac i 1 goblin tank

Cada 2 onades el pressupost puja i per tant pot treure més enemics i més quantitat dels perillosos.

5.2 Estudi i disseny de personatges

5.2.1 Pes narratiu dels personatges

Tot i que en el nostre joc els personatges no tenen un pes narratiu gaire important, si que cal destacar la diferència entre els enemics:

- **Goblin basic:** Un ser maligne i poc intel·ligent que li encanta fer el mal. Són el tipus d'enemic més bàsic, no tenen gran atac ni defensa, però són els més comuns i poden arribar a se un problema quan són molts.

- **Goblin berserker:** Són un tipus de goblin centrat en l'atac. No es preocupen per protegir-se perquè sempre pensen en atacar primer. Són molt ràpids i perillosos.

- **Goblin tank:** Són un tipus de goblin molt estrany de veure. Van molt ben armats i protegits, es centren en tenir un atac decent i molta resistència tot i que són molt lents. prefereixen destruir estructures que no anar per les persones ja que la seva baixa velocitat poques vegades els deixa arribar a objectius mòbils.

També podem destacar els 2 tipus de soldats que ens ajuden:

- **Soldat:** Un soldat valent que intenta protegir la ciutat. Són bons en el combat i tenen una bona defensa, però es poden veure fàcilment superats si els ataquen en número.

- **Capità:** Un soldat veterà entrenat. Tenen molt atac i resistència i poden donar guerra inclús quan es veuen superats en número.

5.2.2 Els personatges com a base de la jugabilitat

Dels personatges mencionats anteriorment molts tenen el mateix disseny amb petits detalls que els diferencien, tant a nivell artístic com de caracterìstiques. Per la creació d'aquests i que tinguessin un colors uniformes es fa seguir com exemple aquest recurs trobat a internet per agafar idees sobre els colors a utilitzar:

Figura 15: Referència de colors per enemics

Goblin Basic: Enemic basic que només va equipat amb una espasa i un tapa-robes.

Figura 16: Enemic bàsic goblin

Goblin Berserker: Enemic centrat en l'atac. Porta unes marques de guerra a la cara per mostrar la seva ferocitat.

Figura 17: Enemic berserker goblin

Goblin tank: Un enemic centrat en la defensa. Molt ben equipat amb armadura completa i casc.

Figura 18: Enemic tank goblin

Soldat: Soldat ben equipat amb armadura, espasa i casc.

Figura 19: Soldat aliat

Capità: Soldat veterà. Porta una distintiva pluma al cap per mostrar el seu alt rang militar.

Figura 20: Capità aliat

Personatge principal: Un habitant del poble que s'omple de valentia per protegir el que queda de casa seva.

Figura 21: Personatge principal

5.2.3 Estil artístic

L'estil artístic es defineix de diversos aspectes:

- La vista que ha de tenir el jugador.
- Tipus de gràfics 2D/3D
- Estil gràfic

La vista triada ha estat Top-Down, ja que deixa que el jugador tingui molt més control de l'espai i el moviment al poder anar en 8 direccions.
El tipus de gràfic s'ha triat el 2D perquè com ja he mencionat anteriorment, l'apartat gràfic no és el meu fort i vull poder aconseguir cert resultat que amb el 3D no serà possible.

Per acabar he triat l'estil gràfic de Pixel Art perquè em sembla més senzill a l'hora de crear els diferents recursos. Tot i que al principi em costarà degut a la poca quantitat de píxels que faré servir, ja que intentaré que gairebé tot sigui 16x16. Un cop m'hi hagi acostumat, tot i barallar-me amb els colors se'm farà més fàcil l'adaptació i creació de certes coses.

5.2.4 Objectes que caracteritzen els personatges

Tot i que no tenim molts objectes que facin una gran caracterització, si que els enemics tenen armes de colors diferent segons el tipus d'enemic:

Els goblins bàsics tenen unes espases porpra que fan referència a la seva gran malicia.

Figura 22: Espasa goblin bàsic

Els goblin Berserker porten unes espases vermelles com les seves marques de guerra.

Figura 23: Espasa goblin berserker

Els Goblin tank porten unes espases ataronjades mostrant el seu bon equipament refinat a les forges.

Figura 24: Espasa goblin tank

5.3 Narrativa

5.3.1 Aclariment

El joc no conté una narrativa molt extensa, ja que es dedicarà el temps a altres aspectes com les mecàniques i l'apartat gràfic i sonor. Tot i això, sí que té una sinopsis per tal de situar-nos en què està passant i en què consisteix el joc.

5.3.2 Sinopsi

El nostre personatge és un habitant d'un poble que ha estat arrasat per les forces del mal i el seu gran exèrcit, en prou feines ha quedat res en peu. Per sort, uns quants han sobreviscut i han atrinxerat l'ajuntament. Ara només queda la força de voluntat i la valentia dels que queden, per intentar defensar el poc que queda del poble fins que tot hagi acabat. Sigui per bé, o per mal.

5.4 Mons de joc

5.4.1 Dimensió física

L'espai dins del joc és bastant ampli tot i que limitat. El poblat està rodejat per una muralla que delimita per on poden i no poden arribar entrar els enemics. Hi ha 4 entrades possibles on la muralla és destruible i els enemics poden arribar a entrar. Dins el poblat hi ha edificis que limiten per on es pot circular i per on no. Tot i això, el personatge té un espai suficient per moure's sense problemes i accedir a les 4 entrades de manera senzilla.
Fora el poblat hi ha una extensa prada verda, tot i que el personatge només pot avançar fins a cert límit.

5.4.2 Dimensió temporal

El temps és rellevant en el joc perquè és el que separa una onada de l'altre. Hi ha 30 segons entre onades perquè el jugador pugui triar quines millores adquirir entre onades per tal de preparar-se per la següent. Menys la primera onada que es deixarà 10 segons per tal que el jugador s'acostumi als controls, entre les següents onades hi haurà 30 segons.

5.4.3 Dimensió ambiental

Com que el joc només es porta a terme en un únic mapa, l'ambientació no canvia. Estem cituats en una gran prada verda on el mig hi ha un poble de sorre d'estil medieval. Els enemics són de fantasia molt fàcilment reconeixibles per les seves llargues orelles. Els soldats que ens ajuden porten unes armadures que recorden al medieval europeu.

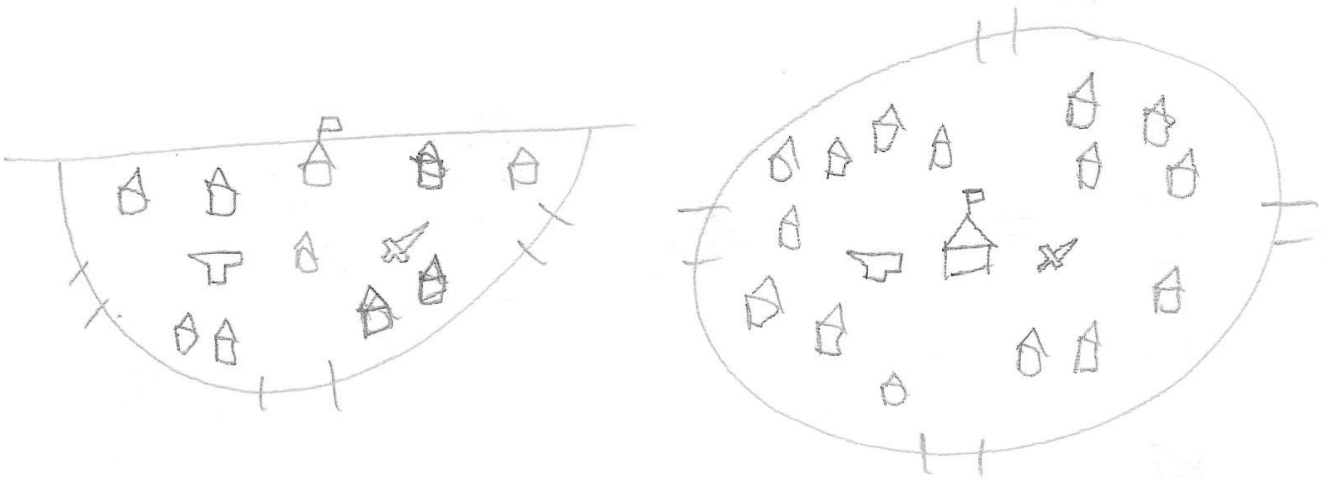


Figura 25: Esbossos poblat

5.5 Interficies

5.5.1 Menú principal

Veurem un menú molt senzill amb un botó de començar.

Figura 26: Esbós de menú principal

5.5.2 Joc

La interfície que el jugador veu mentre juga està composta de:

- **Cors de salut:** Indica la salut restant del jugador, podem pedre un cor sencer o una meitat quan ens fan mal.
- **Número d'onada actual:** Indica a quina onada estem.
- **Temporitzador de pròxima onada:** Quan una onada acaba, indica quan queda fins la següent. Al principi de cada onada desapareix.
- **Monedes del jugador:** Indica quantes monedes té el jugador.
- **Atac del jugador:** Indica l'atac del jugador.
- **Resistència del jugador:** Indica la resistència del jugador.
- **Barra de vida:** Els enemics, estructures i personatges aliats tenen una barra de vida que es mostra quan els hi fan mal. Aquesta canvia de color segons la vida que li quedi al personatge, anat de verd quan la té molt alta, a vermell quan li queda poca.
- **Millores de característiques:** Com a interfície espaial, tenim que quan ens acostem a un edifici en el qual podem portar a terme alguna acció, com la forja per millora alguna caracteristica o la caserna per reclutar ajudants, ens apareixen botons amb els símbols de l'acció a fer

També cal destacar els elements de control:

- **Joystick esquerra:** Permet moure el personatge principal en 8 direccions.
- **Botó d'atac:** Permet atacar en la direcció que s'està movent.

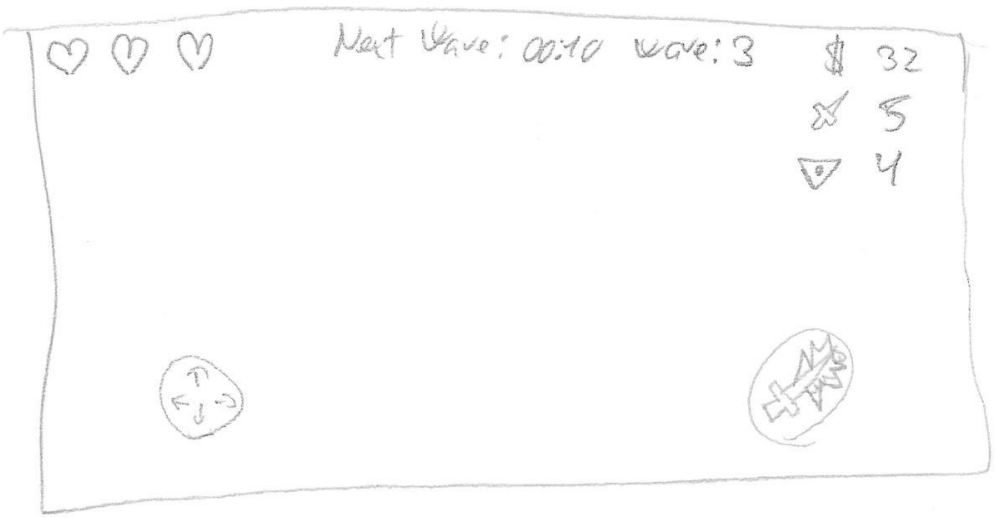


Figura 27: HUD del joc

Quan perdem la partida, la pantalla s'enfosqueix i en ensenya el motiu:

- Personatge principal ha mort.
- Ajuntament destruit.

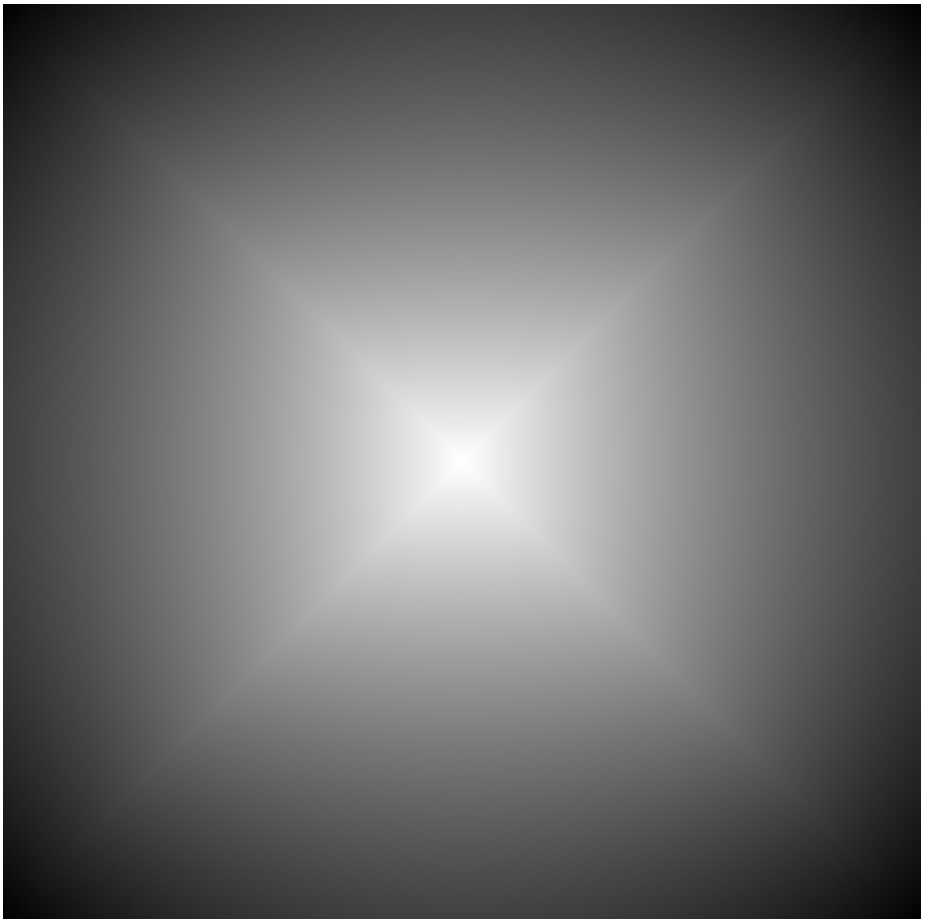


Figura 28: Efecte fade out

Figura 29: Efecte fade out quan es perd la partida

5.5.3 Game Over

A la pantalla de Game Over:

- Es veurà la frase Game Over a dalt en gran.
- La puntuació de l'onada que hem aconseguit i quina ha estat la millor puntuació que hem fet en aquell dispositiu.
- Un botó per tornar al menú principal.

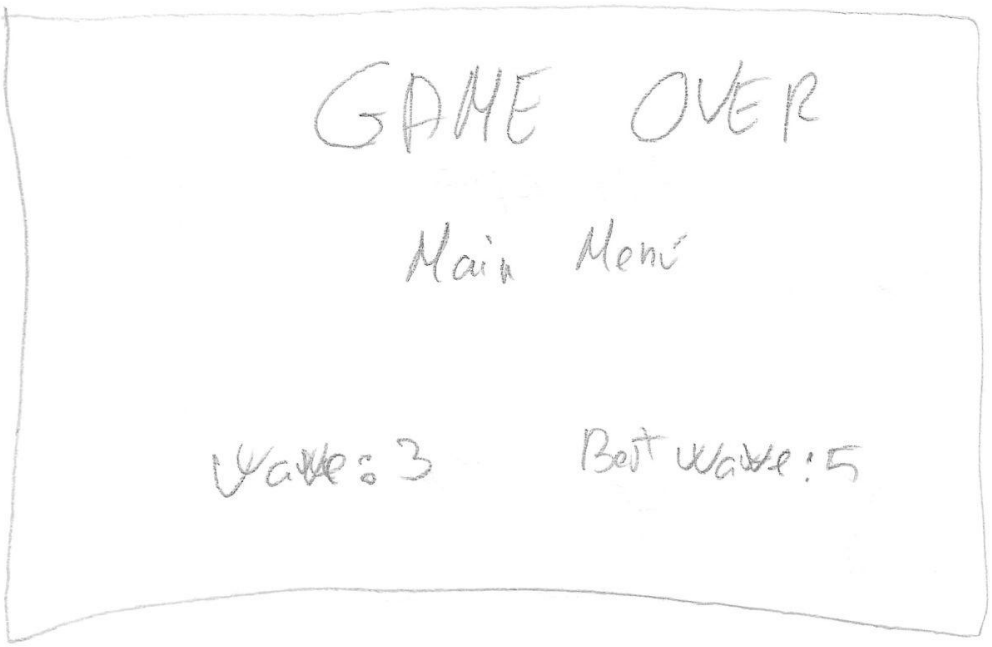


Figura 30: Esbós Game over

5.5.4 Tipografía i font

La tipografia i font que s'ha fet servir és de lliure ús i s'anomena VeniceClassic obtinguda de la pàgina web de *DaFont.com* ja que encaixava amb l'estil pixel art escollit pel joc.

Figura 31: Tipografia

5.6 Game Layout Charts

L'estructura del joc és la següent: comença el joc amb la primera onada, si el jugador la supera passa a la segona, si no, game over. Així es va repetint amb la peculiaritat que cada 2 onades puja el nivell de dificultat. Fent que arribin més enemics i més poderosos.

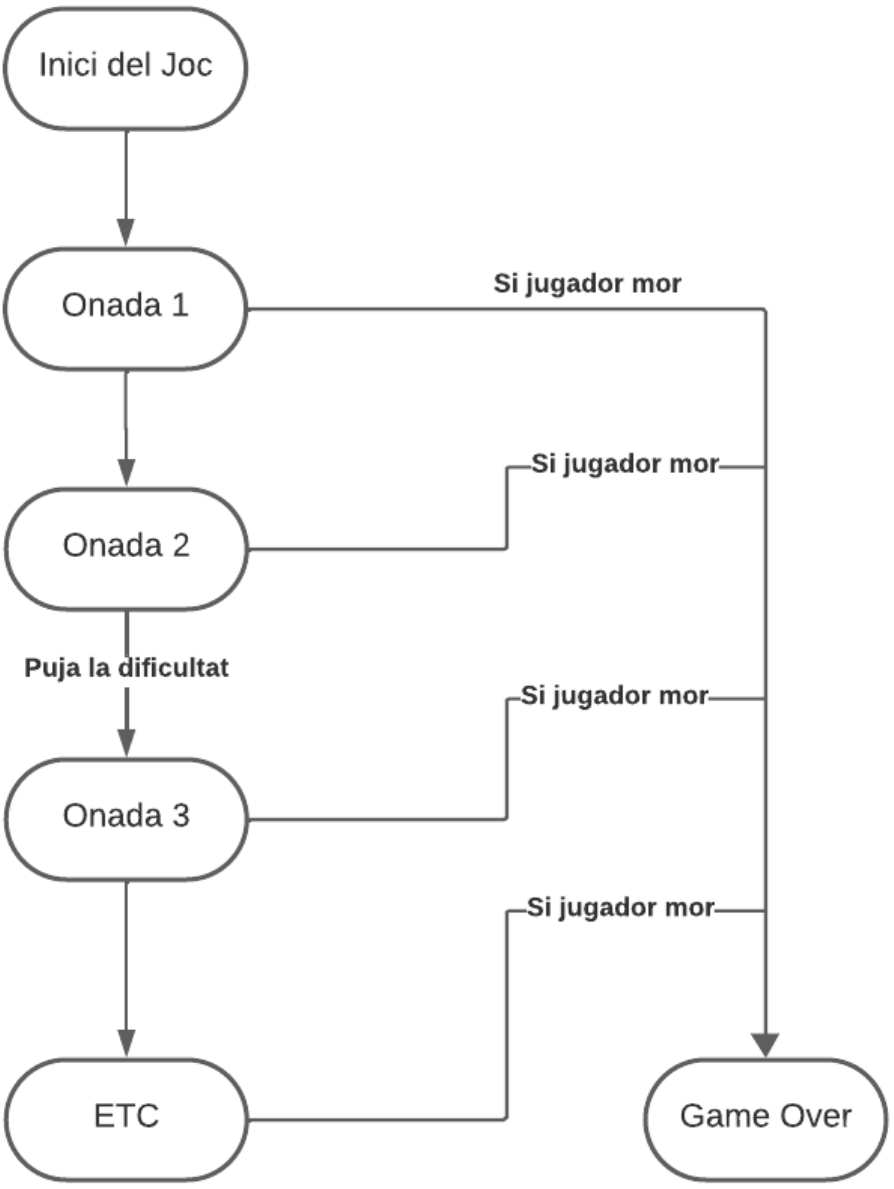


Figura 32: Diagrama de flux del joc

5.7 Model de negoci

Tot i que aquest projecte té un objectiu personal i professional d'aprofundiment d'habilitats i no comercial, en un projecte normal s'hauria de tenir en compte quin model de negoci es vol seguir. Els models actuals més utilitzats són:

- Gratuïts: Consisteix a oferir gratuïtament el producte. El rendiment econòmic es basa en diverses opcions: un cop estàs jugant pots pagar per certs articles, posant anuncis durant el joc, etc.
- Freemium: S'ofereix una part del joc de manera totalment gratuïta, però s'ha de pagar per la resta.
- De pagament: Aquest model és el més conegut i és el fet de haver de pagar (comprar) el producte per poder jugar. És el model més tradicional tot i que actualment es pot comprar una còpia en físic o en digital.
- De subscripció: Molt semblant al primer, però la diferència és que no hi ha un únic pagament per adquirir el producte sinó que s'ha d'anar pagant durant el temps que es vulgui jugar.

En el meu cas, tenint en compte que no volem la necessitat de connexió a internet, el model més adient seria el de pagament o Freemium amb post pagament únic. Distribuït a la plataforma de Play Store. Que es la botiga oficial dels dispositius Android.

5.8 Producció externa

Hi ha diversos recursos que no desenvoluparem nosaltres. Com tots els sons utilitzats per donar més vida al joc. Els quals els hem agafat de webs de sons lliures de drets.

6. Implementació i proves

6.1 Implementació del jugador

El jugador és un dels elements més importants del joc, ja que és el personatge que l'usuari ha de moure i fer servir a l'hora de jugar. És per això que és el personatge més cuidat a nivell de desenvolupament. I que més animacions, feedbacks i detalls diferencials té.

Els components finals del jugador són:

- Un primer objecte pare que conté:
    - L'script amb la lògica del jugador.
    - El Player Input.
    - Un rigidbody 2D i un BoxCollider2D per tractar les físiques del jugador.
    - AudioSource per reproduir certs sons del jugador.
- La càmera: com a fill del jugador, així és segur que ens seguirà en tot moment i el jugador està sempre al centre de la pantalla sense necessitat de codi extra. Aquesta té:
    - AudioListener que permet escoltar tots els sons del joc.
    - AudioSource per reproduir la música principal de fons.
- PlayerFX: Com a fill de l'objecte principal, és l'encarregat de pintar els sprites i animacions del jugador segons el que toqui. Conté:
    - SpriteRenderer és l'encarregat de pintar els sprites per pantalla.
    - Animator, encarregat de gestionar les animacions del player.
    - AudioSource per reproduir certs sons que volem que sonin sense interrompre els altres. Com els efectes de so de donar un cop amb l'espasa.
- Attack transform: Objecte que serveix per saber on està atacant el jugador.
- Player target: és l'objecte que ens dona la posició del player quan l'han de perseguir els enemics.

Figura 33: Composició objecte jugador

6.1.1 Implementació de l'art

S'ha començat per fer la part artística ja que com era un dels punts més complicats segons les meves habilitats, he intentat tenir sempre aquesta part feta per no tenir que preocupar-me més d'aquest apartat.

Després de varies proves, molts tutorials consultats i propostes perquè quedés com havia imaginat. He aconseguit un aspecte que m'ha agradat, i seguint de referencia els spritesheets que se solen fer pels jocs, he fet el meu. Hi ha animacions per estar quiet, animacions per moure's, per atacar i una per morir.

Figura 34: Spritesheet del jugador

6.1.2 Implementació de les mecàniques

Un cop acabat l'aspecte artístic m'he centrar en desenvolupar les mecàniques. He començat per fer un personatge que es mogués per la pantalla fent ús del seu rigidbody. Quant estava fent el moviment em va sorgir el dubte de com es podria aprofitar per desenvolupar des d'ordinador els controls i després aprofitar aquests per el dispositiu mòbil sense tenir que afegir codi extra més endavant. Així que he optat per fer ús del Input System de Unity, per tal de definir unes accions, que a través de diferents Inputs, sempre es fessin crides a aquestes accions. Fent així possible que es pogués controlar per ordinador, i més endavant fer els controls per mòbil.

També es va implementar la funció d'atac que en aquell moment només deixava anar un Debug per consola conforme s'havia rebut l'ordre d'atacar. Més endavant vaig acabar la funció d'atac.

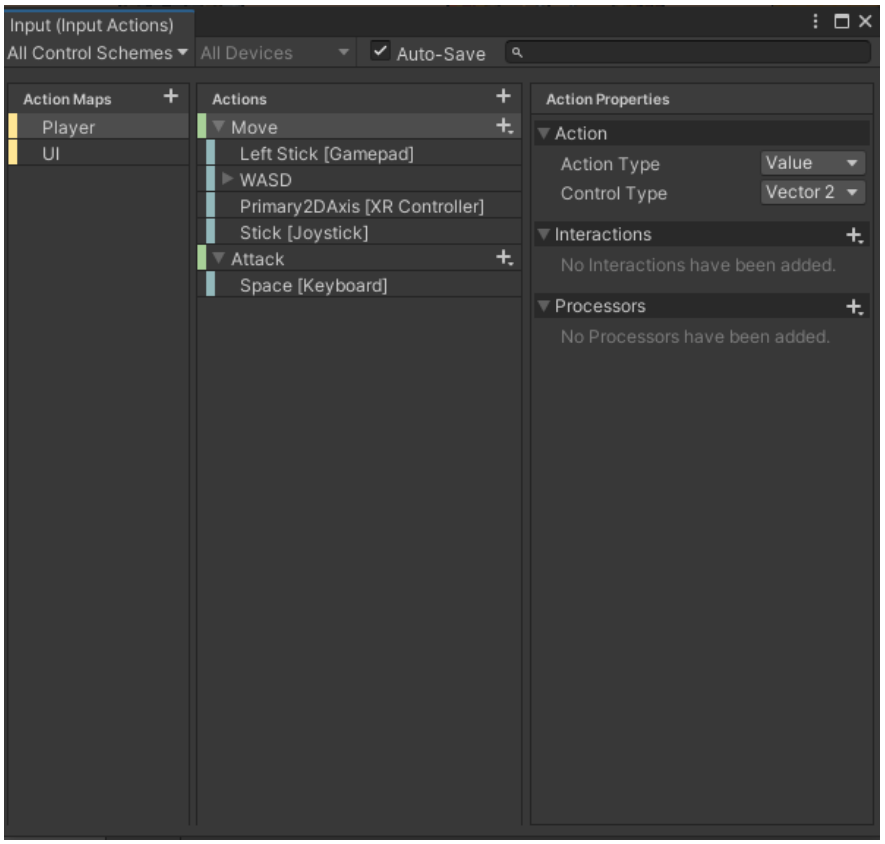


Figura 35: Input system de Unity pel jugador

Aquests Input criden a les funcion **OnMove()** i **OnAttack()** de l'script de la lògica del jugador.

```
0 referencias
void OnMove(InputValue movementValue)
{
    _movementInput = movementValue.Get<Vector2>();
}
```

Figura 36: Codi de la funció per moure's

```
0 referencias
public void OnAttack()
{
    if (!_died && _attackTimeCounter >= _timeBtwAttacks)
    {
        _animator.SetTrigger("Attack");
        _swordSource.clip = _swordClip;
        _swordSource.Play();
        //Debug.Log("Attack");
        _attackTimeCounter = 0f;

        Attack();
    }
}
```

Figura 37: Codi de la funció per atacar

Aquestes funcion només agafen l'ordre i dades dels Inputs i després criden a altres que realitzen les accions de moviment i atac.

```
Mensaje de Unity | 0 referencias
private void FixedUpdate()
{
    if(!_died && !GameManager.Instance._endOfGame)
    {
        if (_movementInput != Vector2.zero)
        {
            _animator.SetFloat("Horizontal", _movementInput.x);
            _animator.SetFloat("Vertical", _movementInput.y);
            updateAttackDirection();
        }

        _animator.SetFloat("Speed", _movementInput.sqrMagnitude);

        if (_movementInput.x < 0)
        {
            PlayerFX.transform.localScale = new Vector3(-1, 1, 1);
        }
        else if (_movementInput.x > 0)
        {
            PlayerFX.transform.localScale = new Vector3(1, 1, 1);
        }

        _rb.velocity = _movementInput * _moveSpeed;
    }

}
```

Figura 38: Codi de moviment del jugador

El moviment es realitza en el **FixedUpdate()** on segons els inputs recollits, ens movem, ens girem, etc.

```
void Attack()
{
    Collider2D[] hitEnemies = Physics2D.OverlapCircleAll(_attackTransform.position, _attackRange, _attackableLayers);

    foreach(Collider2D hit in hitEnemies)
    {
        Damageable damageable = hit.GetComponent<Damageable>();

        if (damageable != null)
        {
            //do dmg
            damageable.takeDamage(_attack);
        }
    }

}
```

Figura 39: Codi de com ataca el jugador

Per fer l'atac creem un Cercle que colisiona amb tots els layers especificats a **_attackableLayers** i ens proporciona un array on hi ha totes les col·lisions que s'han fet.

Després les recorrem per veure si són “Danageable”. Si és així, es crida la funció de fer mal amb la quantitat d’atac que tenim. A l’hora d’atacar, tenim les variables **_attackTimeCounter** i **_timeBtwAttacks** que regulen les vegades que pot atacar el player per segon.

S’ha fet una interfície pels enemics per poder mirar si els objectes col·lisionats se'ls hi pot fer mal o no. I els enemics l’hereten.

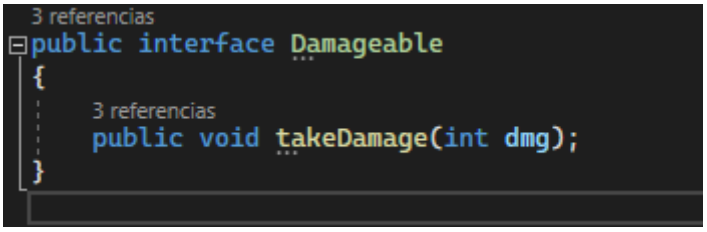


Figura 40: Interfície Damageable

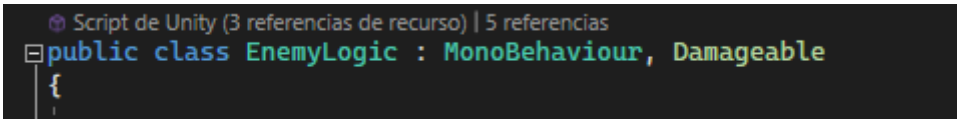


Figura 41: Classe enemic hereta de la interfície

Un cop he tingut els moviments i accions bàsiques vaig començar a donar-li característiques al jugador. Com la seva vida, el seu attack i la seva resistència.

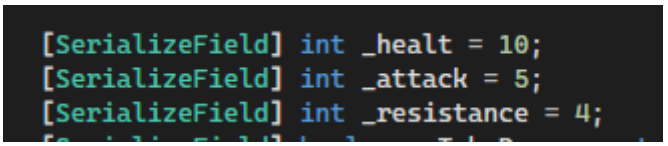


Figura 42: Característiques del jugador

Després de varies proves, vaig veure que tractar sempre la vida de l’1 al 10 i les resistències i atacs com a nombres enters i no amb decimals anava bé per anar controlant la progressió del joc sense tenir mases decimals com a factors de jugabilitat.

El jugador també havia de poder rebre mal, així que he fet la funció de **takedmg(int dmg)** la qual es crida quan algú fereix al jugador. Aquesta, funciona gairebé igual per tots els personatges, fent que quan es rep mal, es calcula la diferencia entre l'atac que s'està rebent i la resistència del personatge, i si es major a 0, es fa mal al personatge i se li resta vida. Si la vida cau a 0 o menor, es crida la funció **die()** que s'encarrega de la mort del personatge.
També hi ha un control per evitar que el jugador mori molt de pressa quan anem pujant de nivell d'onades, s'ha fet que un cop rep mal, es torna invulnerable durant un segon per tal de posicionar-se millor. Això es controla amb la variable **_canTakeDamage** que es torna a posar a *true* al cap d'un segon de patir un cop exitós.

```
1 referencia
public void takedmg(int dmg)
{
    if (!_died)
    {
        if (dmg - _resistance > 0 && _canTakeDamage)
        {
            _healt -= dmg - _resistance;
            _hurtSource.clip = _hurtClip;
            _hurtSource.Play();
            _animator.SetTrigger("Hurt");
            _canTakeDamage = false; // desactivem el fet que ens puguin fer mal per no rebre molt mal de cop
            StartCoroutine(reableTakeDamage());
            GameManager.Instance._playerNeedHeal = true;
            GameManager.Instance.UpdateHealthUI(_healt);

            if (_healt <= 0)
            {
                GameManager.Instance.UpdateHealthUI(0);
                die();
            }
        }
        else if(dmg - _resistance <= 0)
        {
            _hurtSource.clip = _noHurtDamageClip;
            _hurtSource.Play();
        }
    }
}
```

Figura 43: Funció takedmg del jugador

```
1 referencia
IEnumerator reableTakeDamage()
{
    yield return new WaitForSeconds(1f);
    _canTakeDamage = true;
}
```

Figura 44: Funció per tornar a activar que li puguin fer mal al jugador

```
1 referencia
void die()
{
    _died = true;
    _hurtSource.clip = _deathClip;
    _hurtSource.Play();
    _animator.SetTrigger("Die");
    _animator.SetBool("Died", true);

    GameManager.Instance.GameOver();
}
```

Figura 45: Funció de morir del jugador

En el cas del jugador, la funció **die()** crida també la funció de GameOver per fer un final de partida.

El player també té una funció de **updateAttackDirection()** la qual mou el cercle d'atac segon la direcció en la que miri el jugador.

També té unes funcions que van lligades a les millores que es pot comprar, com la de curar-se **Heal()**, i millorar atac i resistència **UpgradeAttack()** i **UpgradeResistance()** que es criden quan el player compra les millores.

```
1 referencia
public int UpgradeAttack()
{
    _attack += ((int)(GameManager.Instance._lvlOfForgeAttack / 10)) + 1;
    return _attack;
}

1 referencia
public int  UpgradeResistance()
{
    _resistance += ((int)(GameManager.Instance._lvlOfForgeResistance / 10)) + 1;
    return _resistance;
}
```

Figura 46: Funcions quan es compren millores

Cada funció del player crida els sons i animación que pertoquen a cada acció per tal de fer-ho més realista i satisfactori per l'usuari.

6.1.3 Implementació de les animacions del jugador

Les animacions del jugador s'han fet a través del editor de sprites i l'eina d'animació de unity. Un cop fetes ens ha quedat aquest diagrama d'animacions:

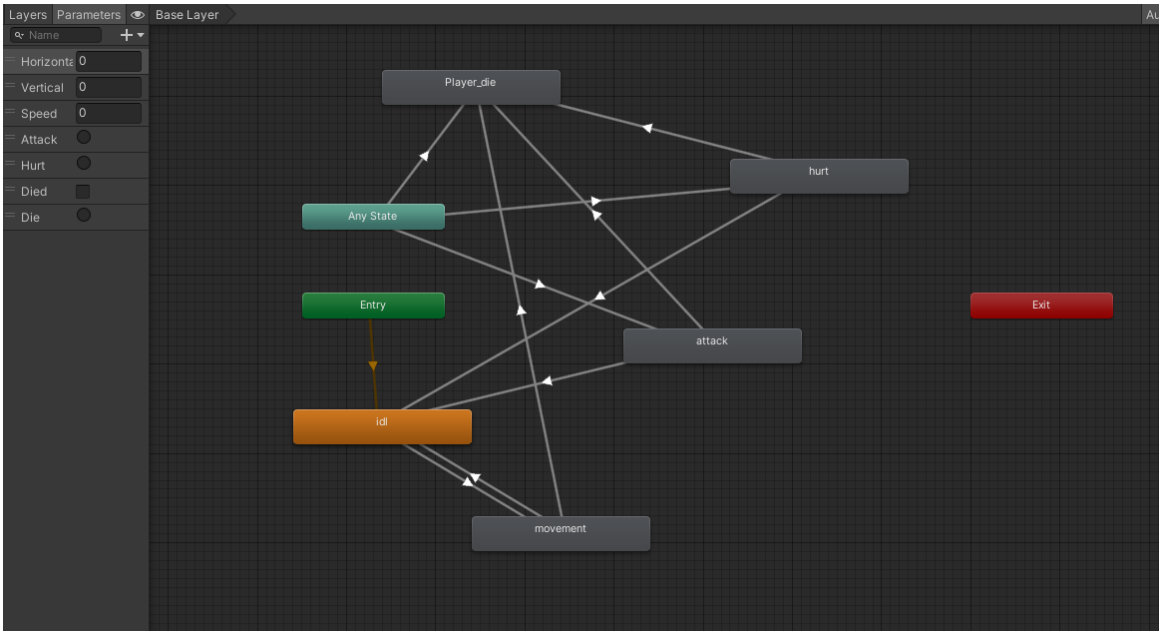


Figura 47: Diagrama d'animacions del jugador

Amb l'excepció de l'animació de morir, les altres estan encapsulades en Blend Tree's que permeten una millor gestió de les animacions sense tenir un diagrama principal molt gran. A més amb les variables Horitzontal i Vertical que es veuen a la imatge, es poden controlar molt fàcilment quines animacions reproduir segons cap on estigui mirant el personatge.

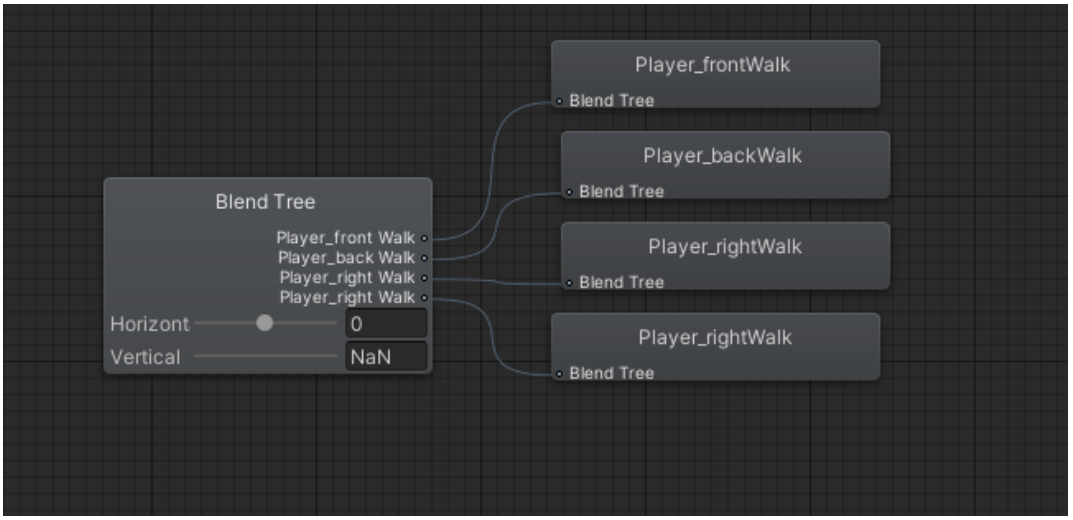


Figura 48: Blend tree del jugador

6.1.4 So

Els sons del jugador són:

- Quan el jugador pateix mal.
- Quan el jugador ataca.

Juntament a aquests sons que només són del jugador, comparteix altres sons amb altres personatges com el so de quan algú intenta fer mal a un altre personatge però el seu atac no es suficient, que sona una espasa que pica contra algo metallic simulant com si s'hagués picat contra l'armadura de l'objectiu i no s'hagués aconseguit donar el cop.
També hi ha el so de morir, el qual es comú en tots els personatges, tant enemics com aliats.

6.2 Implementació del mapa

6.2.1 Implementació de l'art

Per fer el mapa, he intentar crear un tileset per el terra i els murs de 16x16. Però no m'acabava d'agradar el resultat. Així que per el terra he decidir utilitzar un tileset ja fet de lliure ús del projecte *mystic-woods*, el qual també he fet servir per inspirar-me per fer les muralles amb les tanques que hi havien i les tanques les he utilitzat per fer la part destruible de la muralla, la qual parlarem amb més detall més endavant en el seu punt corresponent.

Figura 49: Sprites del terra pel mapa

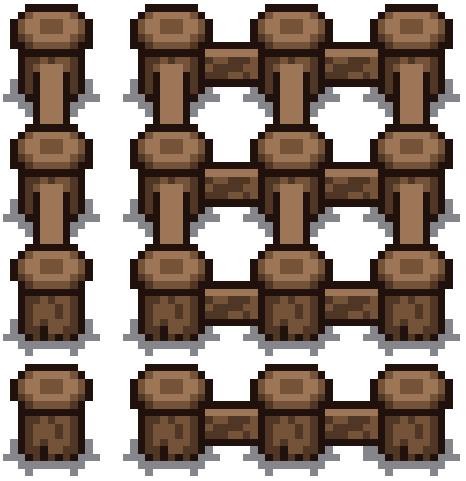


Figura 50: Sprite base les muralles

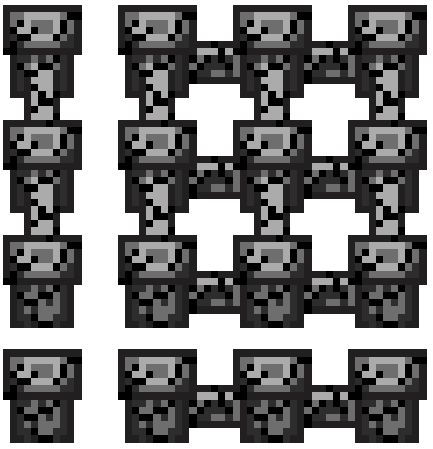


Figura 51: Sprite creat a partir de la base de les muralles

També vaig agafar un seguit d'edificis ja fets i els vaig modificar perquè semblessin destruits.

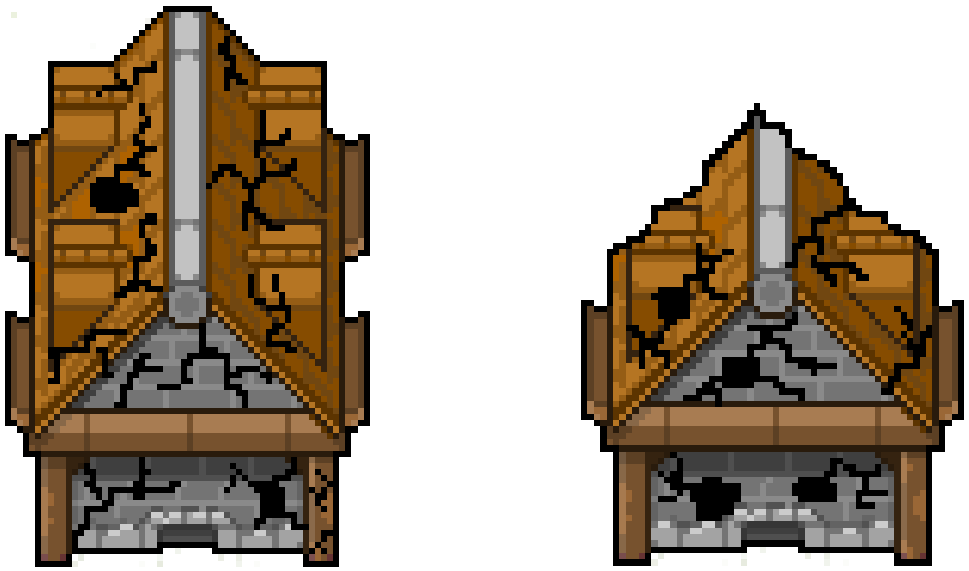


Figura 52: Edificis de decoració mapa

Aquests també els vaig adaptar perquè fossin amb mesures 16x16, de manera que d'un mateix edifici en puguin sortir diferents versions segons els tiles utilitzats.

6.2.2 Implementació del mapa

Per tal de construir el mapa he utilitzat l'eina de tilemap que porta Unity integrada, la qual et permet pintar un mapa de manera molt simple, afegir-hi diferents capes per donar profunditat, posar col·lisions en certes capes, etc.

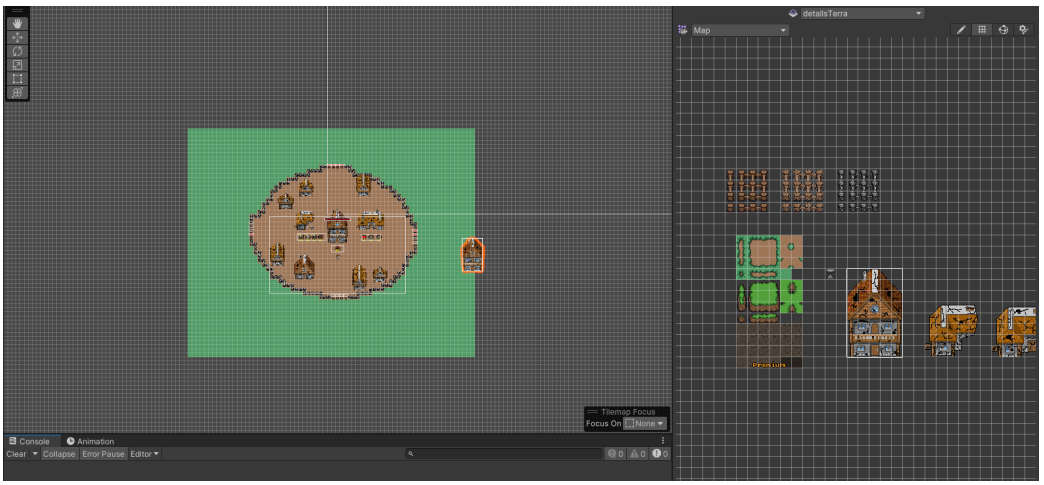


Figura 53: Eina de TileMap de Unity

6.2.3 So

Per el joc he triat una música que acompanyés durant la partida, però que no molestes ni li tregués importància als efectes de so que anirem escoltant de feedback quan passin coses. La cançó es titula: Battle Of The Dragons

6.3 Implementació Game Manager

Un dels components més important en un joc sol ser el GameManager o similar. Que sol ser l'encarregat de gestionar com va avançant la partida, i també serveix com a mètode de connexió d'estats d'aquesta.

Aquest objecte està compost per:

- Script de la lògica del GameManager
- Audiosource per reproduir certs sons com la compra de millores o el GameOver

El GameManager té un script amb el mateix nom que és una classe amb un patró singleton que ens assegura que només existeix una instància d'aquesta.
per això es declara una variable del mateix tipus de la clase "Instància" i aquesta s'inicia al començar i en cas que una altra s'inicií per un altre procés, es destrueix.

```
public static GameManager Instance;
```

Figura 54: Declaració instancia del GameManager

```
Mensaje de Unity | 0 referencias
private void Awake()
{
    if(Instance != null && Instance != this)
    {
        Destroy(this);
        return;
    }
    else
    {
        Instance = this;
    }
}
```

Figura 55: Codi per fer singleton el GameManager

Així doncs, quan es vol fer una crida, no es necessita una referència directa ja que és una clase publica i estàtica i se la pot cridar directament amb:

```
GameManager.Instance.
```

Figura 56: Crida de GameManager fora de la classe

Aquesta és una de les classes més extenses del projecte, ja que ha de posar en comú les diferents valors, és l'encarregada de actualizar el HUD, dir-li als altres objectes quan han de fer certes accions, la progressió del joc i el nivell de dificultat, etc.

Al iniciar es fa un Update de tot el HUD cridant a la funció **UpdateUI(TextMeshProUGUI toChange, string newText)** que ens permet actualizar tots els valors correctament.

També porta els controls de les millores que pot comprar el jugador: la millora d'atac, millora de resistència, millora de les resistències dels murs, restauració de murs, restauració d'edifici central, recuperació de vida del player, reclutar soldats i reclutar Capitans.
Podem veure un exemple:

```
0 referencias
public void UpgradeAttack()
{
    int cost = _lvlOfForgeAttack + _attackCostModifier;
    if(_totalCoins >= cost)
    {
        _audioSource.clip = _buyUpgradeClip;
        _audioSource.Play();

        _totalCoins -= cost;


        int newAttack = _playerTarget.GetComponentInParent<PlayerLogic>().UpgradeAttack();
        _lvlOfForgeAttack++;

        _attackCostModifier += _lvlOfForgeAttack + 1;

        //actualitzem UI
        int nextCost = _lvlOfForgeAttack + _attackCostModifier;
        int nextUpgrade = (int)(_lvlOfForgeAttack / 10) + 1;

        UpdateUI(_attackCostText, nextCost.ToString() + "$");
        UpdateUI(_attackNextUpgradeText, "+" + nextUpgrade.ToString());

        UpdateUI(_playerAttackUIText, ": " + newAttack.ToString());
        UpdateUI(_playerCoinsUIText, ": " + _totalCoins.ToString());

    }
    else
    {
        _audioSource.clip = _notBuyUpgradeClip;
        _audioSource.Play();
    }
}
```

Figura 57: Funció per pujar l'atac del jugador

Quan el jugador vol intentar comprar la millora d'atac, es fa un càlcul del cost actual d'aquesta millora (que es pinta per pantalla en tot moment perquè el jugador sàpiga quan es gastarà) i si es tenen prou monedes, es fa la resta de monedes, s'executa el so conforme s'ha comprat la millora, se li diu al player que executi la funció per millorar el seu atac, es millora el nivell de la forja perquè el cost sigui més elevat el pròxim cop, es calculen els propers costos i quant nivell ens donarà la propera millora, i s'actualitza el HUD per mostrar tots els canvis.

Si pel contrari no és tenen les monedes necessàries, sona un so conforme no s'ha pogut realitzar la compra.

Aquesta lògica es segueix en totes les millores disponibles per comprar dels diferents edificis.

Com el reclutament de soldats, que tot i que la lògica és molt similar, per comprovar si es pot comprar un soldat a part del cost també mira si ja s'han comprat tots els soldats possibles.

```
public void RecruitSoldier()
{
    int cost = recruitCost + (int)(_lvlDifficulty * 1.5f);
    bool canRecruit = false;
    int soldierNum = 0;

    while (soldierNum < _soldiers.Length && !canRecruit)
    {
        if (!_soldiers[soldierNum].activeSelf)
        {
            canRecruit = true;
        }
        else
        {
            soldierNum++;
        }
    }


    if(canRecruit && _totalCoins >= cost)
    {
        _totalCoins -= cost;

        _audioSource.clip = _buyUpgradeClip;
        _audioSource.Play();

        _soldiers[soldierNum].SetActive(true);
        _soldiers[soldierNum].GetComponent<SoldiersLogic>().resetValues();


        UpdateUI(_playerCoinsUIText, ": " + _totalCoins.ToString());
    }
    else
    {
        _audioSource.clip = _notBuyUpgradeClip;
        _audioSource.Play();
    }
}

}
```

Figura 58: Funció per reclutar soldats

També gestiona el control, de quan un mur destruible sigui destruït per l'enemic, ja que s'ha de tornar a escanejar el mapa perquè els enemics sàpiguen que ara ja poden passar a través del mur. Això es fa aplicant l'algorisme A* que explicarem en el següent apartat. En un inici, l'escaneig ho feia cada mur per separat, però feia que es creessin colls d'ampolla quan molts murs eren destruïts a la vegada. Així doncs, he optat perquè quan un mur sigui destruït, aquest avisi al GameManager, i aquest esperi una mica per tal de saber si més murs s'han destruït i fa falta fer més d'un escaneig, per tal d'esperar i fer un únic escaneig per tots els murs destruits.

```
1 referencia
public void destroyedWallsNeedScan()
{
    if(!_pendingToScan)
    {
        _pendingToScan = true;
        StartCoroutine(scanMap());
    }
}

1 referencia
IEnumerator scanMap()
{
    yield return new WaitForSeconds(.5f);
    worldGird.Scan();
    _pendingToScan = false;
}
```

Figura 59: Funcions per escaneig de mapa

Igualment, quan el jugador compra la restauració dels murs danyats o destruits, el GameManager és l'encarregat de dir a tots els murs que es restaurint i un cop acabin fa un únic escaneig del mapa per determinar el camí que poden seguir els enemics un cop restaurats.

```
0 referencias
public void RestoreWalls()
{
    int cost = _lvlOfWalls + _restoreWallsCost;
    if (_wallsNeedRepair && _totalCoins >= cost)
    {
        _totalCoins -= cost;
        _wallsNeedRepair = false;

        _audioSource.clip = _buyUpgradeClip;
        _audioSource.Play();

        for (int i = 0; i < _vallesNord.Length; i++)
        {
            _vallesNord[i].GetComponent<DestructibleObjectsLogic>().restoreWall();
            _vallesSud[i].GetComponent<DestructibleObjectsLogic>().restoreWall();
            _vallesEst[i].GetComponent<DestructibleObjectsLogic>().restoreWall();
            _vallesOest[i].GetComponent<DestructibleObjectsLogic>().restoreWall();
        }

        worldGird.Scan(); // tornem a fer scan per fer que els enemics no atravessin els murs

        _restoreWallsCost += _lvlOfWalls + 1;

        //actualitzem UI
        int nextCost = _lvlOfWalls + _restoreWallsCost;

        UpdateUI(_wallRepairCostText, nextCost.ToString() + "$");
        UpdateUI(_playerCoinsUIText, ": " + _totalCoins.ToString());
    }
    else
    {
        _audioSource.clip = _notBuyUpgradeClip;
        _audioSource.Play();
    }
}
```

Figura 60: Funció de restaurar murs

El GameManager com hem dit és l'encarregat de gestionar com avança la partida, per això es l'encarregat de mostrar el temps que falta per la pròxima onada, i de dir-li a l'encarregat de crear enemics, quan ho ha de fer. Això es fa amb un contador que es va pintant al HUD entre onada i onada.

```
2 referencias
public void startTimer(int temp)
{
    _timeBetweenWaves = temp;
    _timerWaveText.gameObject.transform.parent.gameObject.SetActive(true);
    UpdateUI(_timerWaveText, "00:" + _timeBetweenWaves.ToString("D2"));
    InvokeRepeating("countDown", 1f, 1f);
}

0 referencias
void countDown()
{
    _timeBetweenWaves--;
    UpdateUI(_timerWaveText, "00:" + _timeBetweenWaves.ToString("D2"));

    if (_timeBetweenWaves < 0) // temporitzador ha acabat, generem nova Wave
    {
        CancelInvoke("countDown");
        _timerWaveText.gameObject.transform.parent.gameObject.SetActive(false);
        _waveSpawner.GenerateWave();
        _waveCount++;
        UpdateUI(_countWaveText, _waveCount.ToString("D2"));
    }
}
```

Figura 61: Funcions de control de temps entre onades

La gestió de les monedes del jugador per saber quantes en té també es porta a través del GameManager per després poder saber si pot optar a una de les millores o no. Aquesta funció és cridada pels objectes moneda quan el jugador les agafa.

```
1 referencia
public void addCoins(int value)
{
    _totalCoins += value;
    UpdateUI(_playerCoinsUIText, ": " + _totalCoins.ToString());
}
```

Figura 62: Funció de sumar monedes

Quan el joc s'acaba i fem la transició cap la pantalla de Game Over també és una de les tasques del GameManager, fent que quan se li envia l'ordre de Game Over, apaga tots els sons de l'escena, reprodueix el so de Game Over fa un FadeOut en negre de la pantalla i mou la càmera segons el motiu de la partida finalitzada, si és el jugador que ha quedat sense vides, es queda igual, però si s'ha perdut la partida perquè han destruït l'edifici principal, mou la càmera per ensenyar que l'edifici principal ha estat destruït. I finalment, canvia d'escena.

```
2 referencias
public void GameOver()
{
    _endOfGame = true;
    AudioSource[] allAudios;
    allAudios = FindObjectsOfType<AudioSource>() as AudioSource[];
    foreach (AudioSource audioS in allAudios)
    {
        audioS.Stop();
    }

    _audioSource.clip = _gameOverClip;
    _audioSource.Play();

    if(PlayerPrefs.GetInt("BestWabe") < _waveCount) // actualitzem puntuació més alta si l'hem aconseguit
    {
        PlayerPrefs.SetInt("BestWabe", _waveCount);
    }
    PlayerPrefs.SetInt("WavesEnded", _waveCount); // i ens guardem la puntuació que hem fet aquesta partida

    StartCoroutine(changeScene());
}
```

Figura 63: Funció de Game Over per quan acaba la partida

```
1 referencia
public void MoveCamera()
{
    _mainCamera.transform.position = new Vector3(_objectiuPrincipalTarget.transform.position.x, _objectiuPrincipalTarget.transform.position.y, _mainCamera.transform.position.z);
}
```

Figura 64: Funció de moure la càmera al acabar partida

```
1 referencia
IEnumerator changeScene()
{
    while(_audioSource.isPlaying || _fadeOutImage.color.a < 1)
    {
        _fadeOutImage.color = new Color(_fadeOutImage.color.r, _fadeOutImage.color.g, _fadeOutImage.color.b, _fadeOutImage.color.a + 0.1f); // efecte fade out
        yield return new WaitForSeconds(.2f);
    }

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

Figura 65: Funció per canviar d'escena

6.4 Implementació d'algorisme A*

Durant el desenvolupament vaig intentar implementar el meu propi algorisme de A* per fer la cerca de camins que els enemics havien de buscar i fer. Després de varis intents, no estava obtenint els resultats que esperava ja que no acabava de funcionar correctament i era molt lent i poc optimitzat, a demés, l'única vegada que vaig aconseguir un sistema mínimament òptim perquè el joc no es pengés, va ser utilitzant el TileMap de Unity, aprofitant la graella que aquest fa. Però no em resolia certs problemes com per exemple la lògica de tanques destruibles on si hi havia la tanca no havia de deixar passar a l'enemic i un cop destruïda (que no era fàcil de destruir des de TileMap, ja que no es podien guardar dades i era una capa de "pintat"), tornar a generar nodes que ara si deixessin passar els enemics per allà. Vaig buscar més informació i fer l'algorisme correctament comportava molt més temps i habilitat del que en primer moment havia pensat ja que necessitava fer un sistema que creés una graella pròpia que identifiqués correctament els obstacles i anès fent un escaneig del mapa quan fos necessari.

Buscant més informació de com poder fer això, em vaig topar amb el projecte **A* Pathfinding Project**, un projecte lliure de drets que a través d'un algorisme de A* feia que els enemics perseguissin el jugador. Després de llegir la documentació, tutorials i exemples del projecte vaig provar-ho fent modificacions als exemples que donaven a la seva documentació per tal de que els enemics es comportessin com jo necessitava. L'explicació de com es va implementar la part de l'enemic s'explicarà més endavant en el seu apartat, ara ens centrarem en explicar què faig servir del projecte **A*** en el meu cas.

Així doncs, aquest package em deixa crear una graella virtual on definir-li quins Layers són obstacles i fer un escaneig amb les dimensions del mapa que li marquis.

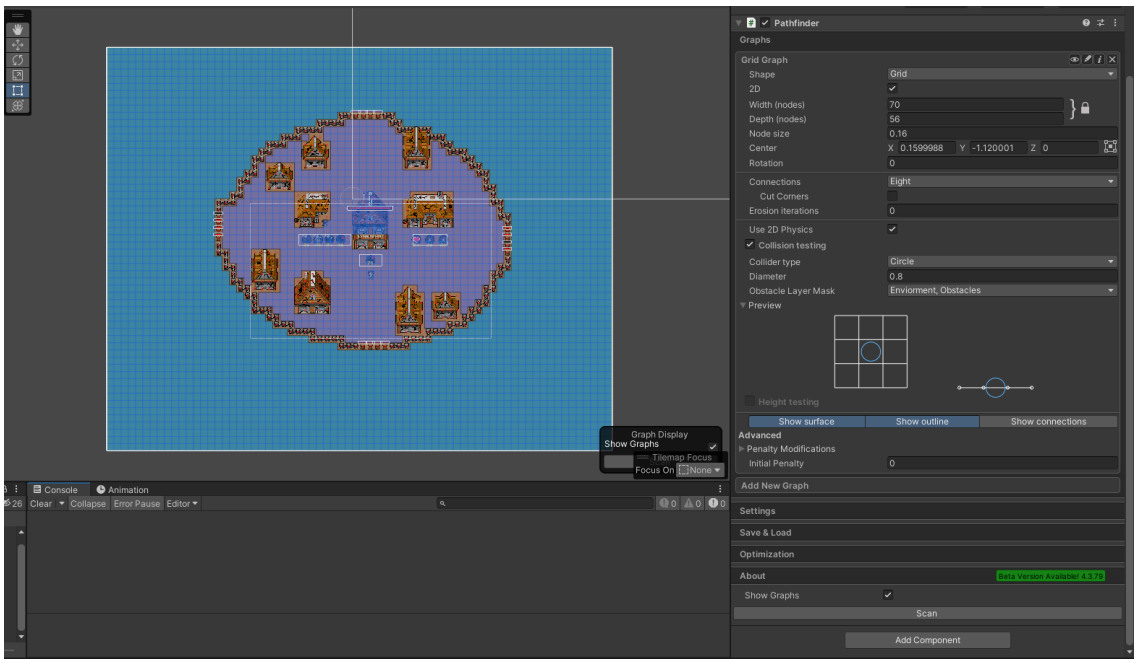


Figura 66: Graella A*

Quan es fa un escaneig inicial el mapa queda tancat, i quan una tanca és destruïda, fent un altre escaneig es pot determinar quin serà el nou camí que poden seguir els enemics.

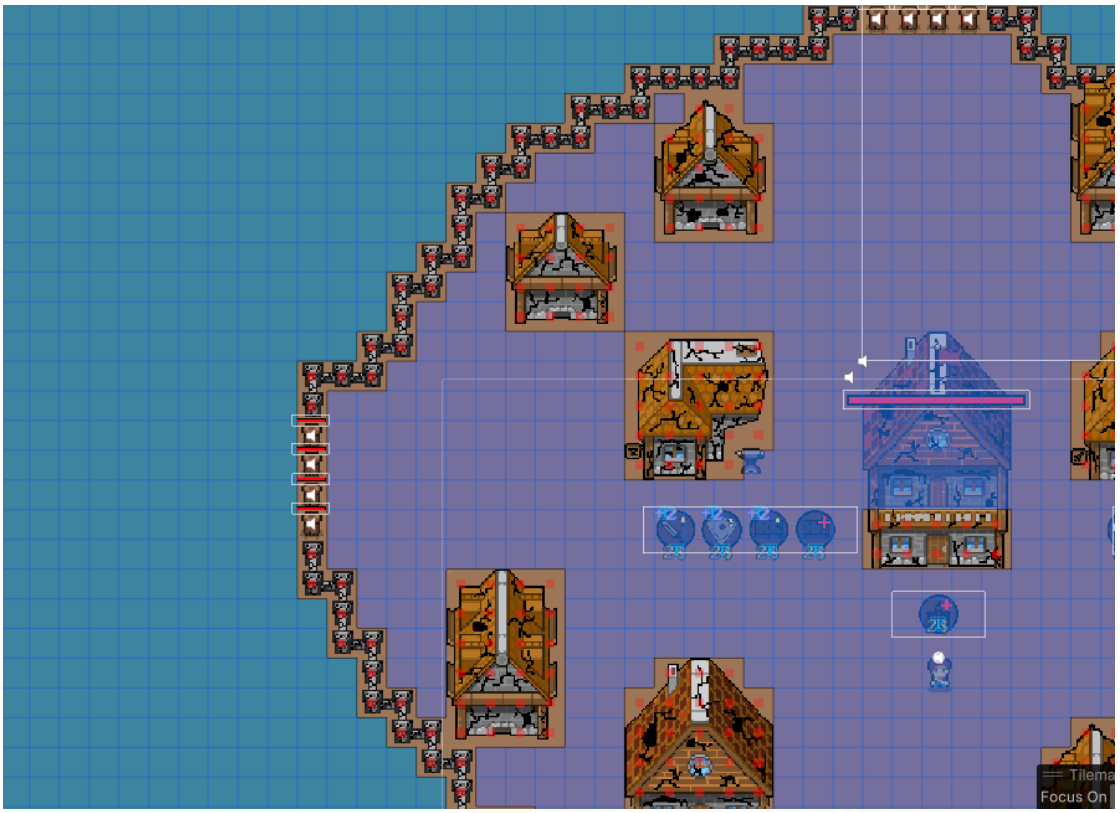


Figura 67: Escaneig de graella amb la muralla

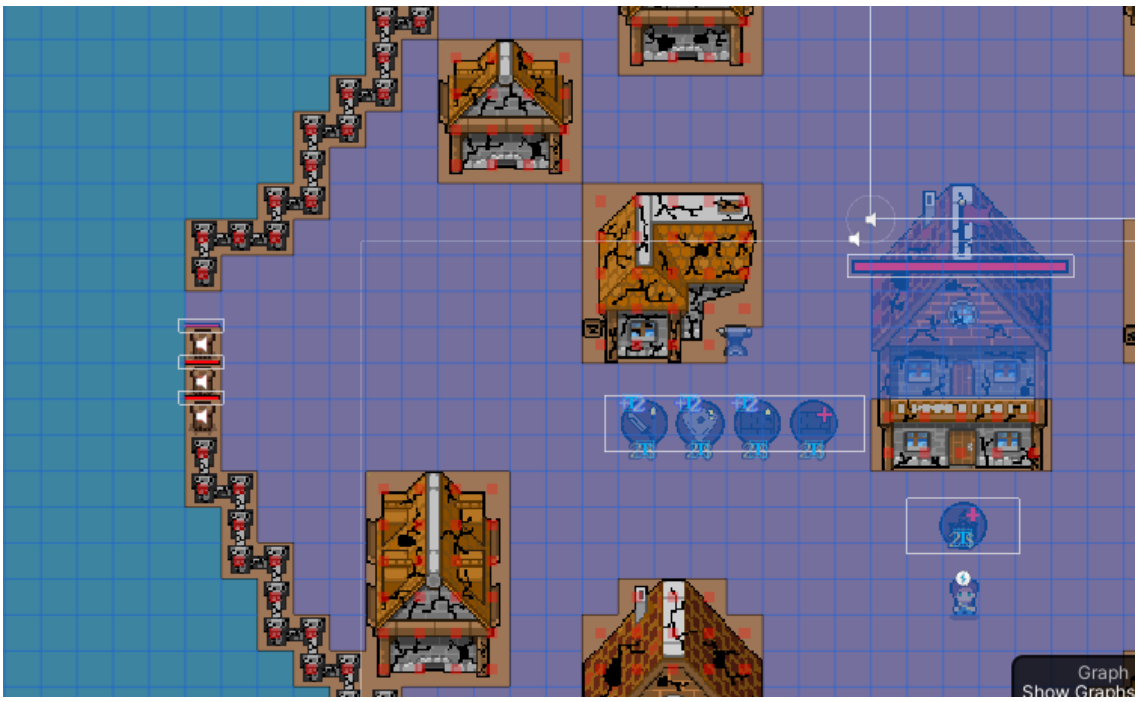


Figura 68: Escaneig de graella amb una muralla destruida

6.5 Implementació dels enemics

Els enemics estan compostos de diferents parts igual que el Jugador:

- Objecte principal on hi ha:
    - Script de la lògica per trobar els camins cap els objectius
    - Script de la lògica de l'enemic amb les seves característiques, vida, etc.
    - Un rigidbody i un CercleCollider2D per les físiques i col·lisions.
    - Un Audiosource per reproduir certs sons com quan l'enemic pateix mal.

- Un prefab de la vida de l'enemic explicat en l'apartat de les interfícies en profunditat:
    - El canvas que es pinta espaialment sempre sobre l'enemic
    - La barra de vida que va disminuint i canvia de color quan queda poca vida
    - L'script de la lògica de la barra de vida

- Enemic FX:
    - SpriteRenderer per pintar l'enemic
    - Animator per la gestió de les animacions
    - AudioSource per reproduir certs sons com quan l'enemic fa un atac.

- Detector:
    - La lògica del detector, encarregat de mirar quan un player o soldat està a l'abast de l'enemic per canviar el target de prioritat cap aquest.

Figura 69: Prefab enemic Goblin bàsic

Igual que amb el jugador, he fet servir els Layers de Unity per detectar les col·lisions que m'interessen per cada element.

6.5.1 Implementació de l'art

Per els enemics es va començar fent l'enemic bàsic i a partir d'aquest es van crear els altres 2.

Seguint com a referència el personatge per les mides i les animacions a fer, es va crear un sprite sheet per el primer enemic:

Figura 70: Spritesheet goblin bàsic

D’aquest primer es va crear el del Goblin Berserker:

Figura 71: Spritesheet goblin berserker

I el del goblin Tank:

Figura 72: Spritesheet goblin tank

6.5.2 Implementació de mecàniques

Per la implementació de les mecàniques explicarem els 2 scripts principals de l'enemic i l'script del detector.

EnemyAI:

Aquest script s'encarrega de trobar els camins cap els objectius de l'enemic i el moviment. Donant sempre prioritat al seu TargetPrincipal, que si el jugador o els soldats no estan a prop, sempre serà l'edifici central. I si no pot arribar a aquest, intentarà anar a per les muralles més pròximes al seu punt d'aparició.

EnemyLogic:

Encarregat de tota la resta de la logica de l'enemic que no tingui a veure amb la cerca de camins i moviment. Com la gestió de la vida, atac, resistència, quan mor, etc.

El flux que segueix un enemic és que quan el generador d'onades el crea es crida una funció d'EnemyLogic la qual inicialitza l'enemic i li dóna els objectius a cercar. Un cop assignats, li diu a EnemyAI que comenci a buscar camins possibles. La Corutina de buscar camins, que cada cert temps intenta buscar camins nous si surt l'oportunitat com per exemple si es trenca la muralla o si apareix el jugador a prop de l'enemic i és possible anar a atacar-lo.

```
1 referencia
public void AssignObjectives(int origin)
{
    _ai = GetComponent<EnemyAI>();
    _ai.firstPrincipalTarget = _ai.principalTarget = GameManager.Instance._objectiuPrincipalTarget.transform; //asignem objectiu principal

    switch (origin) // assignem objectius alternatius segons on hem spawnejat
    {
        case 0: // nord

            for(int i = 0; i < GameManager.Instance._vallesNord.Length; i++)
            {
                _ai.alternativeTargets.Add(GameManager.Instance._vallesNord[i].transform);
            }

            break;

        case 1: // sud

            for (int i = 0; i < GameManager.Instance._vallesSud.Length; i++)
            {
                _ai.alternativeTargets.Add(GameManager.Instance._vallesSud[i].transform);
            }

            break;

        case 2: // est

            for (int i = 0; i < GameManager.Instance._vallesEst.Length; i++)
            {
                _ai.alternativeTargets.Add(GameManager.Instance._vallesEst[i].transform);
            }

            break;

        case 3: // oest

            for (int i = 0; i < GameManager.Instance._vallesOest.Length; i++)
            {
                _ai.alternativeTargets.Add(GameManager.Instance._vallesOest[i].transform);
            }

            break;
    }
    StartCoroutine(searchPaths());
}
```

Figura 73: Funció d'assignar objectius als enemics

```
1 referencia
IEnumerator searchPaths()
{
    while(!_died)
    {
        yield return new WaitForSeconds(.5f);
        _ai.SearchClosest();
    }
}
```

Figura 74: Funció de cercar camins que es va repetint cada mig segon

EnemyAI começa a buscar els camins possibles cap els objectius marcats, primer sempre intenta trobar un camí cap a l'objectiu principal. I després mira els altres objectius. Un cop ha fet els camins a tots els objectius, mira quin és l'objectiu que té més aprop. Normalment intenta prioritzar l'objectiu principal, però fem servir la variable **marginEffortToPriority** per determinar quina distància un enemic ha de deixar estar el seu target principal i anar per un secundari. Això ens permet per exemple que els Goblins Berserker tingui aquest marge molt alt, fent que molt sovint prioritzin l'edifici central. Però per exemple els Goblin Tank, si han de recórrer molta distància primer es dediquin a destruir més la muralla.

Un cop determinat l'objectiu i el camí en el **FixedUpdate()** fem que el goblin es mogui cap allà.

```
1 referencia
public void SearchClosest()
{
    //Si algun path s'estava calculant, el cancelem
    if (principalTargetPath != null)
        principalTargetPath.Error();

    if (lastPaths != null)
    {
        for (int i = 0; i < lastPaths.Length; i++)
            lastPaths[i].Error();
    }

    finalTargets.Clear();

    if (!_dead)
    {
        for (int i = 0; i < alternativeTargets.Count; i++)
        {
            if (alternativeTargets[i].gameObject.GetComponent<SpriteRenderer>().enabled) // si la muralla esta activa
            {
                finalTargets.Add(alternativeTargets[i]);
            }
        }

        //Creem un nou array de lastPaths
        if (lastPaths == null || lastPaths.Length != finalTargets.Count) lastPaths = new Path[finalTargets.Count];

        //Reset variables
        bestPath = null;
        numCompleted = 0;
        currentWaypoint = 0;

        principalTargetLenght = float.PositiveInfinity;
        principalTargetPath = null;

        if (!_dead)
        {
            //Creem un nou path cap al target principal
            ABPath p = ABPath.Construct(transform.position, principalTarget.position, OnPriorityPathCompete);
            p.nnConstraint = NNConstraint.None;
            AstarPath.StartPath(p);
        }
    }
}
```

Figura 75: Funció per inicialitzar cerca de camins

```
1 referencia
private void OnPriorityPathCompete(Path p)
{
    if (p.error)
    {
        //Debug.LogWarning("No es pot arribar al target prioritari!\n" + p.errorLog);
        principalTargetLenght = float.PositiveInfinity;
    }
    else
    {
        principalTargetLenght = p.GetTotalLength();
        principalTargetPath = p;
    }


    if (!_dead)
    {
        //Mirem la resta de paths per tots els targets
        if(finalTargets.Count > 0)
        {
            for (int i = 0; i < finalTargets.Count; i++)
            {
                //Creem un nou path cap al target
                ABPath newPath = ABPath.Construct(transform.position, finalTargets[i].position, OnTestPathComplete);


                lastPaths[i] = newPath;

                AstarPath.StartPath(newPath);
            }
        }
        else
        {
            CompleteSearchClosest();
        }

    }
}
```

Figura 76: Funció que comprova si s'ha pogut arribar a l'objectiu principal

```
// Quan s'acaba de calcular un path
1 referencia
public void OnTestPathComplete(Path p)
{
    if (p.error)
    {
        //Debug.LogWarning("One target could not be reached!\n" + p.errorLog);
    }


    //Make sure this path is not an old one
    for (int i = 0; i < lastPaths.Length; i++)
    {
        if (lastPaths[i] == p)
        {
            numCompleted++;

            if (numCompleted >= lastPaths.Length)
            {
                CompleteSearchClosest();
            }
            return;
        }
    }
}
```

Figura 77: Funció que comprova si s'ha pogut arribar als objectius alternatius

```
// Quant tots els paths han acabat
1 referencia
public void CompleteSearchClosest()
{
    //Find the shortest path
    Path shortest = null;
    float shortestLength = float.PositiveInfinity;

    //Loop through the paths
    for (int i = 0; i < lastPaths.Length; i++)
    {
        //Get the total length of the path, will return infinity if the path had an error
        float length = lastPaths[i].GetTotalLength();

        if (shortest == null || length < shortestLength)
        {
            shortest = lastPaths[i];
            shortestLength = length;
        }
    }

    //Debug.Log(principalTargetLenght - shortestLength);
    if (principalTargetPath != null && principalTargetLenght - shortestLength < marginEffortToPriority)
    {
        bestPath = principalTargetPath;
        shortestLength = principalTargetLenght;
    }
    else
    {
        bestPath = shortest;
    }

    //Debug.Log("Path trobat amb una logintud de " + shortestLength);


}
```

Figura 78: Funció que comprova quin és l'objectiu més proper segons prioritat

```
    Mensaje de Unity | 0 referencias
    void FixedUpdate()
    {
        if (bestPath == null)
            return;

#if (UNITY_EDITOR)

        if (bestPath.vectorPath != null)
        {
            for (int i = 0; i < bestPath.vectorPath.Count - 1; i++)
            {
                Debug.DrawLine(bestPath.vectorPath[i], bestPath.vectorPath[i + 1], Color.green);
            }
        }
#endif
        _destination = bestPath.vectorPath[bestPath.vectorPath.Count-1];

        if (currentWaypoint >= bestPath.vectorPath.Count)
        {
            reachedEndOfPath = true;
            return;
        }
        else
        {
            reachedEndOfPath = false;
        }


        Vector2 direction = ((Vector2)bestPath.vectorPath[currentWaypoint] - rb.position).normalized;
        float distance = Vector2.Distance(rb.position, bestPath.vectorPath[currentWaypoint]);

        if(distance > 0.02f)
        {
            rb.MovePosition(rb.position + direction * speed * Time.fixedDeltaTime);
            logic.updateAnimations(direction);
        }
        else
        {
            logic.updateAnimations(Vector2.zero);
        }


        if (distance < nextWayPointDistance)
        {
            currentWaypoint++;
        }
    }
```

Figura 79: FixedUpdate on fem el moviment de l'enemic

Per la implementació d'aquest script s'ha consultat la documentació i s'ha participat en la comunitat de Discord del projecte ***"A* Pathfinding Project"*** El qual ens proporciona tota l'estructura de la graella per la qual es creen nodes que els enemics intentaran seguir per fer el camí o "*path*" fins al seu objectiu.

Per la resta de lògica de l'enemic tenim l'script de EnemyLogic el qual és similar al del jugador.
Es declaren les característiques de l'enemic, les monedes que deixarà anar, la seva barra de vida i el seu Detector, així com els diferents components per els sons, les animacions i les característiques per atacar:

```
[SerializeField] EnemyAI _ai;

[SerializeField] GameObject _coin;

//dades
[SerializeField]
int _health = 10;
[SerializeField]
int _damage = 1;
[SerializeField]
int _resistance = 1;
[SerializeField]
int _enemyType = 0;

public GameObject _healthCanvas;
public HealthBar _healthBar;

[SerializeField] DetectorLogic _detector;


//sounds
[SerializeField] AudioSource _hurtSource;
[SerializeField] AudioClip _hurtClip;
[SerializeField] AudioClip _noHurtDamageClip;
[SerializeField] AudioClip _deathClip;

[SerializeField] AudioSource _swordSource;
[SerializeField] AudioClip _swordClip;


//atack
[SerializeField] float _timeBtwAttacks = 0.5f;
private float _attackTimeCounter;
[SerializeField] float _attackRange = 1.5f;
[SerializeField] LayerMask _attackableLayers;

//animacions
[SerializeField]
GameObject EnemyFX;
[SerializeField]
Animator _animator;
[SerializeField]
public bool _died = false;
```

Figura 80: Variables de característiques enemic

En l'Start inicialitza l'enemic perquè tingui un nivell segons el nivell de dificultat que haguem arribat:

```
Mensaje de Unity | 0 referencias
private void Start()
{
    _damage += (int)(GameManager.Instance._lvlDifficulty * 2f);
    _resistance += (int)(GameManager.Instance._lvlDifficulty * 1.5f);
    _coin = Resources.Load<GameObject>("Objects/Coins/Coins");

    _healthCanvas.SetActive(false);
    _healthBar.SetMaxHealth(_health);

    _hurtSource.clip = _hurtClip;
    _swordSource.clip = _swordClip;
}
```

Figura 81: Funció Star de la lògica de l'enemic

Hi ha una funció pensada per anar actualitzant les animacion de l'enemic segons si s'està movent, ha d'atacar, etc.

```
2 referencias
public void updateAnimations(Vector2 direction)
{
    if(!_died)
    {
        if (direction != Vector2.zero)
        {
            _animator.SetFloat("Horizontal", direction.x);
            _animator.SetFloat("Vertical", direction.y);
        }

        _animator.SetFloat("Speed", direction.sqrMagnitude);

        if (direction.x < 0)
        {
            EnemyFX.transform.localScale = new Vector3(-1, 1, 1);
        }
        else if (direction.x > 0)
        {
            EnemyFX.transform.localScale = new Vector3(1, 1, 1);
        }
    }
}
```

Figura 82: Funció que actualiza les animacions de l'enemic

L’atac de l’enemic es mira primer en el FixedUpdate per saber si estem tocant alguna cosa quan ataquem.

```
Mensaje de Unity | 0 referencias
private void FixedUpdate()
{
    if(!_died && !GameManager.Instance._endOfGame)
    {
        Collider2D[] hitEnemies = Physics2D.OverlapCircleAll(transform.position, _attackRange, _attackableLayers);
        if(hitEnemies.Length > 0)
        {
            if (_attackTimeCounter >= _timeBtwAttacks)
            {
                _animator.SetTrigger("Attack");
                _swordSource.Play();

                foreach (Collider2D hit in hitEnemies)
                {
                    attack(hit.gameObject);
                }

                _attackTimeCounter = 0f;
            }
        }
    }
}
```

Figura 83: FixedUpdate enemic

I després s’executa la funció d’atacar segons el tipus d’objectiu que estiguem atacant.

```
1 referencia
void attack(GameObject obj)
{
    _animator.SetTrigger("Attack");
    _swordSource.Play();
    //Debug.Log("Enemy Attack");
    if(obj.layer == LayerMask.NameToLayer("Player"))
    {
        obj.GetComponent<PlayerLogic>().takedmg(_damage);
    }
    else if(obj.layer == LayerMask.NameToLayer("Soldiers"))
    {
        obj.GetComponent<SoldiersLogic>().takedmg(_damage);
    }
    else if(obj.layer == LayerMask.NameToLayer("Obstacles"))
    {
        obj.GetComponent<DestructibleObjectsLogic>().takedmg(_damage);
    }
    else if( obj.layer == LayerMask.NameToLayer("Ajuntament"))
    {
        obj.GetComponentInParent<HallTownLogic>().takedmg();
    }

}
```

Figura 84: Funció d’atac de l’enemic

La funció de takeDamage(int dmg) que s'ha de definir obligatoriament al heretar de la classe Damageable:

```
3 referencias
public void takeDamage(int dmg)
{
    if(!_died)
    {
        if (dmg - _resistance > 0)
        {
            if (!_healthCanvas.activeSelf)
            {
                _healthCanvas.SetActive(true);
            }

            _health -= dmg - _resistance;
            _healthBar.SetHealth(_health);
            _hurtSource.clip = _hurtClip;
            _hurtSource.Play();

            if (_health <= 0)
            {
                die();
            }
        }
        else
        {
            _hurtSource.clip = _noHurtDamageClip;
            _hurtSource.Play();
        }
    }
}
```

Figura 85: Funció takeDamage enemic

I hi ha la funció per quan l'enemic es mor que crida una Corrutina que permet anar deshabilitant les coses poc a poc per tal de donar un bon efecte de mort i que no hi hagi cap problema.

```
void die()
{
    //animació de morir i destruir objecte;
    GetComponent<CircleCollider2D>().enabled = false;
    _animator.SetTrigger("Die");
    _hurtSource.clip = _deathClip;
    _hurtSource.Play();
    _died = true;
    GameManager.Instance._waveSpawner._spawnedEnemies.Remove(this.gameObject); // el borrem de l'spawner
    _ai.clearAll();
    _ai._dead = true;
    _ai.enabled = false;
    _detector.enabled = false;
    StartCoroutine(destroy());

}

1 referencia
IEnumerator destroy()
{
    yield return new WaitForSeconds(2f);
    _healthCanvas.SetActive(false);
    var newCoin = Instantiate(_coin,transform.position,Quaternion.identity);
    newCoin.GetComponent<coinsLogic>().CreateCoin(_enemyType);
    GetComponentInChildren<SpriteRenderer>().enabled = false;
    yield return new WaitForSeconds(3f);
    Destroy(this.gameObject);
}
```

Figura 86: Funcions de morir i destruir-se d'enemic

Per acabar, els enemics tenen un component que s'anomena Detector, el qual detecta en certa zona si el jugador o un soldat està al seu abast per anar-los a atacar. Aquest component porta un script que es dedica a mirar si algú s'apropa.

```
private void FixedUpdate()
{
    if (!_enemyLogic._died && !GameManager.Instance._endOfGame)
    {
        Collider2D[] hitPlayer = Physics2D.OverlapCircleAll(transform.position, _detectorkRange, _playerLayer);
        Collider2D[] hitSoldiers = Physics2D.OverlapCircleAll(transform.position, _detectorkRange, _detectableLayers);
        if (hitPlayer.Length > 0)
        {
            _enemyAI.principalTarget = GameManager.Instance._playerTarget.transform;
        }
        else if (hitSoldiers.Length > 0)
        {
            if (hitSoldiers.Length > 0)
            {
                float distance = float.PositiveInfinity;
                Transform soldierPos = null;
                foreach (Collider2D hit in hitSoldiers)
                {
                    Vector2 newVec = hit.gameObject.transform.position - _enemyLogic.transform.position;
                    float newDist = ((Vector2)(hit.gameObject.transform.position - _enemyLogic.gameObject.transform.position)).magnitude;

                    if (newDist < distance)
                    {
                        distance = newDist;
                        soldierPos = hit.gameObject.transform;
                    }
                }

                _enemyAI.principalTarget = soldierPos;
            }
        }
        else
        {
            if(_enemyAI.principalTarget != GameManager.Instance._objectiuPrincipalTarget.transform)
            {
                _enemyAI.principalTarget = GameManager.Instance._objectiuPrincipalTarget.transform;
            }
        }
    }
}
```

Figura 87: Detector enemic

Sempre es dóna prioritat al jugador, per tal de poder donar més marge a fer una pilota d'enemics que et vagin perseguint i anar-los derrotant segons l'estratègia triada, pot ser anar-te acostant i allunyant, fer que es trobin amb els soldats, etc.

6.5.3 Implementació de les animacions

Igual que al jugador, fent ús de l'eina de Unity s'han creat les animacions dels enemics. Tot i que tots tenen un sprite diferent per les animacions, l'arbre d'animacions és pràcticament el mateix:

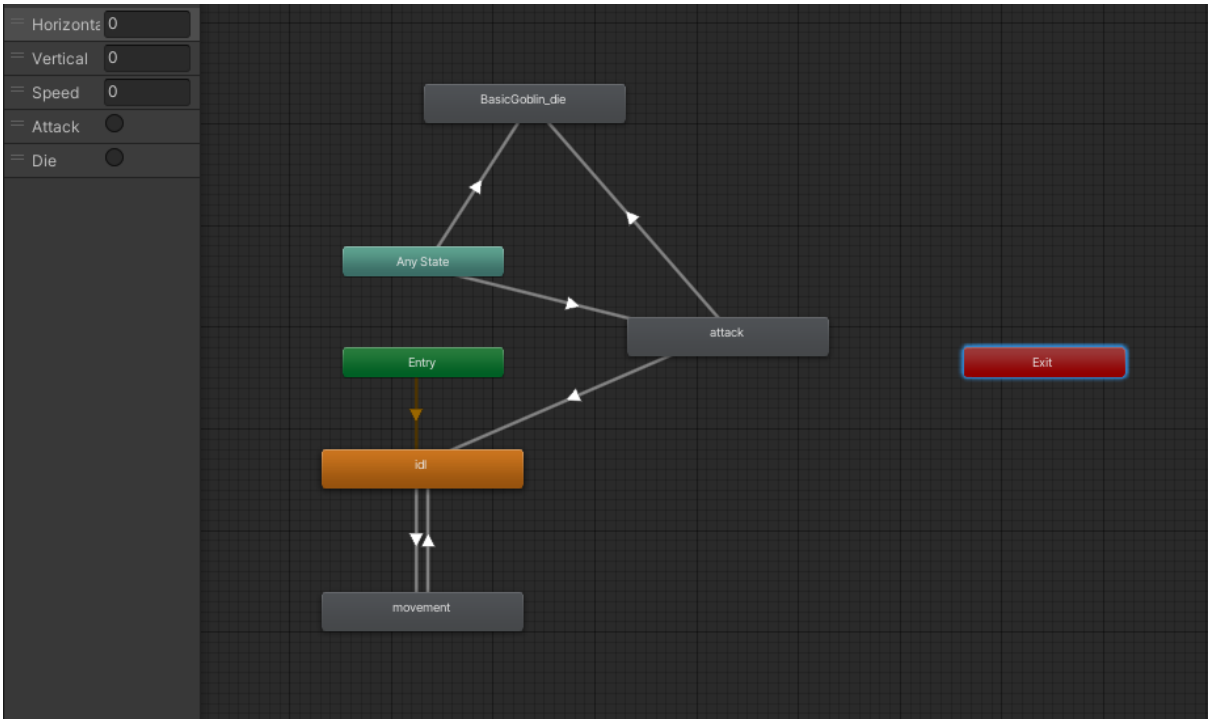


Figura 88: Animacions enemic

També s'han utilitzat els Blend Tree's com en el jugador.

6.5.4 So

Els sons propis dels enemics són:

- Quan un enemic pren mal
- Quan un enemic ataca

Igual que el jugador, comparteix els sons abans mencionats com el de mort i quan l'ataquen però no li fan mal.

6.6 Implementació de les onades d'enemics

Un dels aspectes més importants del nostre joc és que tingués la capacitat de crear diferents onades d'enemics. Per això es s'ha creat la classe i objecte WaveSpawner. L'objecte que crea els enemics es composa de l'script amb la lògica de la creació d'onades i fills que són les posicions on podrà crear enemics.

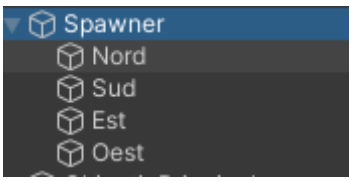


Figura 89: Composició WaveSpawner

Aquesta classe recull els tipus d'enemics que hi ha, el pressupost que s'assigna per l'onada actual i quan portem gastat. S'assigna un tipus i cost per enemics amb la classe següent:

```
[System.Serializable]
2 referencias
public class EnemyType
{
    public GameObject enemyPrefab;
    public int cost;
}
```

Figura 90: Classe tipus d'enemic

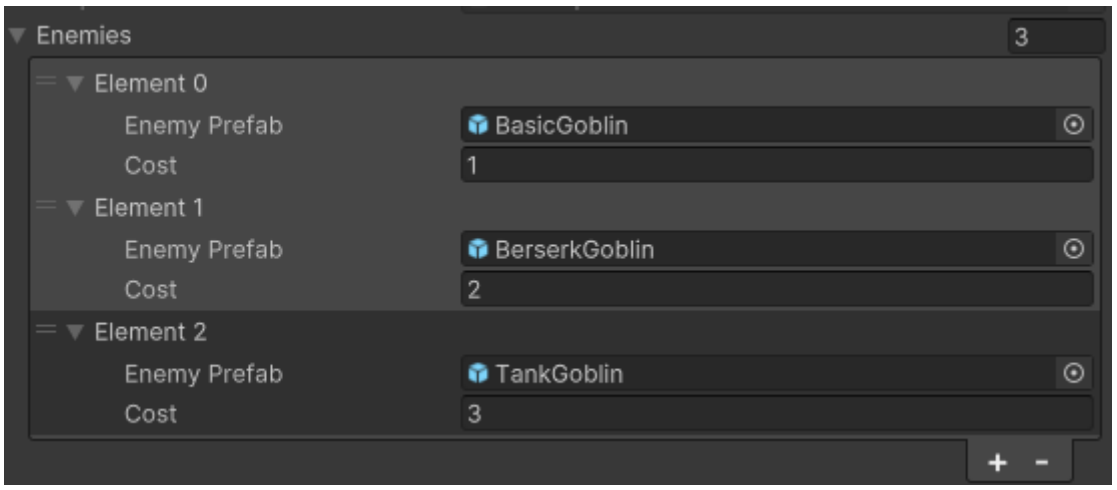


Figura 91: Diferents enemics assignats

Així doncs, la lògica d'aquesta classe es basa en assignar un pressupost a cada onada, i quan se li diu que l'onada ha de començar es mira quins enemics es generaran. Se li ha assignat un cost a cada tipus d'enemic. Així que el WaveSpawner intenta crear una onada d'enemics segon el pressupost que té. Per fer-ho, tria aleatoriament un tipus d'enemic i mira si amb el pressupost que té el pot crear. Si es així, el crea i es resta el cost, si no, mira si en pot crear un altre fins que tingui un pressupost de 0.

```
1 referencia
public void GenerateWave()
{
    _waveValue = _currWave; // donem pressupost al spawner per crear enemics
    GenerateEnemies();

    _waveEnded = false;
    _endOfWaveAnounced = false;
}
```

Figura 92: Funció generació d'onades

```
1 referencia
public void GenerateEnemies()
{
    List<GameObject> generatedEnemies = new List<GameObject>();
    while(_waveValue > 0)
    {
        int randEnemyId = Random.Range(0, _enemies.Count);
        int randEnemyCost = _enemies[randEnemyId].cost;

        if(_waveValue-randEnemyCost >= 0)
        {
            generatedEnemies.Add(_enemies[randEnemyId].enemyPrefab);
            _waveValue -= randEnemyCost;
        }
        else if(_waveValue <= 0)
        {
            break;
        }
    }

    _enemiesToSpawn.Clear();
    _enemiesToSpawn = generatedEnemies;
}
```

Figura 93: Funció de generació d'enemics segons pressupost

Un cop ha creat l'onada d'enemics, els comença a crear aleatoriament en els punts del mapa on els pot fer aparèixer. I els anem traient de la llista d'enemics a generar i la posem a la llista d'enemics creats. Quan aquesta llista d'enemics creats arriba a 0, es dona la onada per acabada i es comunica al GameManager perquè posi en marxa un comptador fins la següent. I, si han passat 2 onades, li diem també al GameManager que pugi la dificultat.

```
Mensaje de Unity | 0 referencias
void FixedUpdate()
{
    if(_spawnTimer <= 0)
    {
        //spawn enemy
        if(_enemiesToSpawn.Count > 0)
        {
            int i = Random.Range(0, _spawnLocations.Length);
            GameObject enemy = Instantiate(_enemiesToSpawn[0], _spawnLocations[i].position, Quaternion.identity);
            enemy.GetComponent<EnemyLogic>().AssignObjectives(i);
            _enemiesToSpawn.RemoveAt(0);
            _spawnedEnemies.Add(enemy);
            _spawnTimer = _spawnInterval;
        }
        else
        {
            if (_spawnedEnemies.Count <= 0)
            {
                //end of wave
                if (!_endOfWaveAnounced)
                {
                    endOfWave();
                    _endOfWaveAnounced = true;
                }
            }


        }
    }
    else
    {
        _spawnTimer -= Time.fixedDeltaTime;
    }
}
```

Figura 94: Instancia dels enemics per l'onada

```
1 referencia
void endOfWave()
{

    _waveEnded = true;

    GameManager.Instance.startTimer(30); // temporitzador fins propera onada


    _waveCount++;
    if (_waveCount > 1) //sumar dificultat cada 2 onades
    {
        GameManager.Instance.AddLvlDifficulty();
        _currWave += (int)(GameManager.Instance._lvlDifficulty * 1.5f);
        _waveCount = 0;
    }
}
```

Figura 95: Funció per quan acaba una onada

6.7 Implementació dels Menús

6.7.1 Main Menu

El Menú principal és molt senzill, ja que al estar pensat per dispositius mòbils, no el volia sobrecarregar. Per això està compost d'un únic botó de començar el joc, el millor puntuatge que s'ha fet en aquest dispositiu, el player al costat fent la seva animació base, i un goblin que va passant de tant en tant perquè no sigui un menú estàtic i poc canviant.

6.7.1.1 Implementació de l'art

Per l'art del menú s'han creat diferents recursos com els fons on col·locar els botons, s'han fet servir el TextMeshPro de Unity per els textos i s'han utilitzat colors que es faran servir en el joc més endavant perquè tot tingui continuïtat.

Figura 96: Menú principal

Figura 97: Menú principal animació goblin 1

Figura 98: Menú principal animació goblin 2

6.7.1.2 Implementació lògica

Per aconseguir aquest efecte del jugador i el goblin caminant hem fet unes petites animacion i un script que a part de controlar el botó que comença el joc, també fa que el goblin passi per la pantalla cada cert temps.

```
void Update()
{

    if (_move)
    {
        _goblin.rectTransform.anchoredPosition = new Vector2(_goblin.rectTransform.anchoredPosition.x + 200f * Time.deltaTime, _goblin.rectTransform.anchoredPosition.y);

        if (_goblin.rectTransform.localPosition.x >= 669.2f)
        {
            _move = false;
            float temp = Random.Range(5f, 10f);
            Invoke("StartMove", temp);
        }
    }

}

0 referencias
void StartMove()
{
    _goblin.rectTransform.localPosition = _from;

    _move = true;
}
```

Figura 99: Lògica del goblin passant per la pantalla

```
0 referencias
public void StartGame()
{
    SceneManager.LoadScene("Game");
}
```

Figura 100: Funció canviar d'escena a joc

6.7.1.3 So

Per el menú principal hem triat una música que ens donés la sensació de que estem a punt de començar un gran repte.
La cançó es titula: Kings of camelot

6.7.2 Game Over

El menú de Game Over també és molt sencill, però canviem totalment com es veu posant un negre en comptes d'un verd per reflectir que hem perdut i ens han derrotat.

6.7.2.1 Implementació de l'art:

Apareix les lletres Game Over, la teva puntuació, la millor puntuació, un botó per anar al menú principal i 3 goblin que tot i que és el seu moviment IDL, amb la música que acompanya el menú hem aconseguit que sembli que ballin amb sarcasme.

Figura 101: Menú Game Over

6.7.2.2 Implementació de la lògica

Com en el menú principal, tenim una lògica bàsica per canviar d'escena quan s'apreta el botó.

```
0 referencias
public void mainMenu()
{
    SceneManager.LoadScene("MainMenu");
}
```

Figura 102: Funció canviar escena a menú principal

6.7.2.3 So

Per el so hem apostat per una música més tenebrosa que es va repetint mentre estiguis a la pantalla de Game Over, que sumat al moviment dels goblin del mig de la pantalla, dóna totalment l'efecte de que hem perdut i el poble ha estat devastat.
La cançó s'anomena: KL Music Box Game Over II.

6.8 Implementació interficies

Per la creació del HUD i altres parts de la interfície s'han crear diferents botons, assets de fonts del TextMeshPro de Unity i diferents sprites per representar cada una.

6.8.1 Implementació de l'art

Tenim el HUD principal, on apareixen els controls per mòbil, la vida del jugador, el contador fins la següent onada, el número d'onada actual, les monedes que té el jugador i el seu atacar i defensa.

Figura 103: HUD del joc

A part del HUD principal també s'han crear els botons que apareixen quan t'acostes a una estructura on pots comprar alguna millora o efecte.

Figura 104: Botons de millores de la forja

Figura 105: Botó de recuperació edifici central

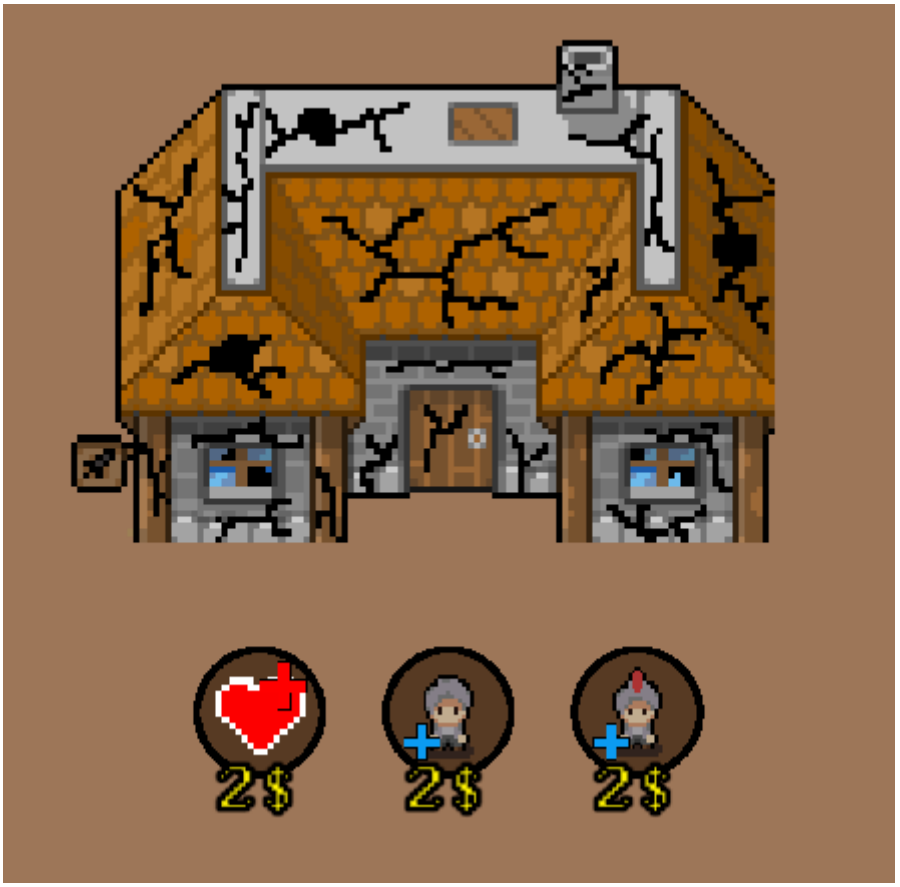


Figura 106: Botons de curar-se i reclutar ajudants

6.8.2 Implementació de la lògica

Per fer les interfícies s'ha creat funcions específiques dins el GameManager que ens ajuden a actualitzar les dades que es mostren per pantalla. Aquest, també té una referència de tots els texts que s'han d'anar actualitzant per donar la informació a l'usuari i els cors que es fan servir per pintar la vida del jugador.

```
//---------UI----------------//

[SerializeField] Image[] _playerHearths;
[SerializeField] Sprite _fullHearth;
[SerializeField] Sprite _halfHearth;
[SerializeField] Sprite _emptyHearth;

[SerializeField] TextMeshProUGUI _timerWaveText;
[SerializeField] int _timeBetweenWaves = 30;

[SerializeField] TextMeshProUGUI _countWaveText;

[SerializeField] TextMeshProUGUI _playerCoinsUIText;
[SerializeField] TextMeshProUGUI _playerAttackUIText;
[SerializeField] TextMeshProUGUI _playerResistanceUIText;

[SerializeField] TextMeshProUGUI _attackCostText;
[SerializeField] TextMeshProUGUI _attackNextUpgradeText;

[SerializeField] TextMeshProUGUI _resistanceCostText;
[SerializeField] TextMeshProUGUI _resistanceNextUpgradeText;

[SerializeField] TextMeshProUGUI _wallupgradeCostText;
[SerializeField] TextMeshProUGUI _wallNextUpgradeText;

[SerializeField] TextMeshProUGUI _wallRepairCostText;


[SerializeField] TextMeshProUGUI _hallTownRepairCostText;


[SerializeField] TextMeshProUGUI _playerHealCostText;
[SerializeField] TextMeshProUGUI _capitainCostText;
[SerializeField] TextMeshProUGUI _soldierCostText;

[SerializeField] Image _fadeOutImage;
```

Figura 107: Referències dels textos i components de la interfície

Així doncs, quan es vol actualitzar alguna dada a mostrar, cridem a la funció **UpdateUI().** Aquesta demana un TextMeshPro de Unity i un string el qual serà el nou text a introduir.

```
40 referencias
void UpdateUI(TextMeshProUGUI toChange, string newText)
{
    toChange.text = newText;
}
```

Figura 108: Funció per actualitzar la UI

```
UpdateUI(_playerResistanceUIText, ": " + newResistance.ToString());
```

Figura 109: Exemple de com es fa servir la funció d'actualitzar la UI

Figura 110: Exemple de text a la UI

Per actualizar la vida del jugador, al no ser un text i esta composta per diferents cors, vaig fer una funció que segons la vida que el jugador digués que li queda, es pintin cors sencers, mitjos o cors buits, quan aquell cor ja no compta com a vida.

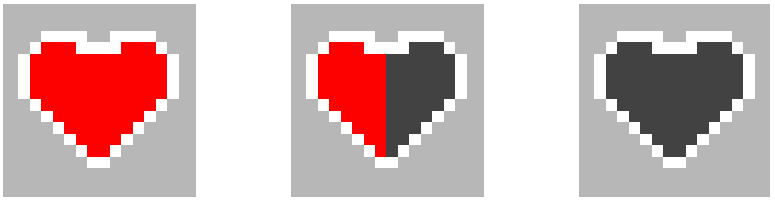


Figura 111: Diferents estats de la vida del jugador

```
public void UpdateHealthUI(int health)
{
    if(health % 2 == 0) // la vida és un numero parell, per tant pintem cors sencers
    {
        for(int i = 0; i < (int)health/2; i++)
        {
            _playerHearths[i].sprite = _fullHearth;
        }

        if(health < 10)
        {
            for(int i = (health / 2); i < _playerHearths.Length; i++)
            {
                _playerHearths[i].sprite = _emptyHearth;
            }
        }

    }
    else // la vida és numero senar
    {
        for( int i = 0; i < (int)health/2; i++)
        {
            _playerHearths[i].sprite = _fullHearth;
        }
        _playerHearths[(health/2)].sprite = _halfHearth;

        if (health < 9)
        {
            for (int i = (health/2) + 1; i < _playerHearths.Length; i++)
            {
                _playerHearths[i].sprite = _emptyHearth;
            }
        }
    }
}
```

Figura 112: Funció que actualitza els cors que es mostren per pantalla segons la vida del jugador

6.8.3 So

Els botons per comprar millores o efectes tenen un so per quan el podem comprar i un quan no alhora de prémer el botó. Aquests sons es reprodueixen al AudioSource del GameManager.

6.9 Implementació recursos

El recurs principal del joc són les monedes, que el jugador aconsegueix al derrotar enemics i que pots fer servir per obtenir diferents millores.

6.9.1 Implementació de l'art

Per la creació de les monedes hem agafat un recurs d'internet ja creat, ja que no aconseguia el resultat que buscava.

Figura 113: Sprites de monedes

6.9.2 Implementació mecàniques

Per la implementació de les monedes vaig voler que cada enemic deixes anar certa quantitat de monedes. Després de investigar com fer això vaig descobrir el ScriptableObject. Que em permeten guardar les dades de manera molt òptima i endreçada per tal de després llegir aquestes dades a l'hora de crear l'objecte dins l'escena.

Així doncs vaig crear un script de ScriptableObject de CoinType:

```
using UnityEngine;

[CreateAssetMenu(fileName = "ScriptableCoin", menuName = "ScriptableObjects/CoinType")]
Script de Unity | 4 referencias
public class CoinType : ScriptableObject
{
    [SerializeField] private int coinValue;
    [SerializeField] private Sprite coinSprite;
    [SerializeField] private AudioClip coinSound;

    3 referencias
    public Sprite CoinSprite { get { return coinSprite; } }
    3 referencias
    public int CoinValue { get { return coinValue; } }
    3 referencias
    public AudioClip CoinSound { get { return coinSound; } }
}
```

Figura 114: Codi ScriptableObject per les monedes

Un cop creat l'script podia crear tants coinTypes com necessites per les monedes amb el mateix menu de creació de Unity:

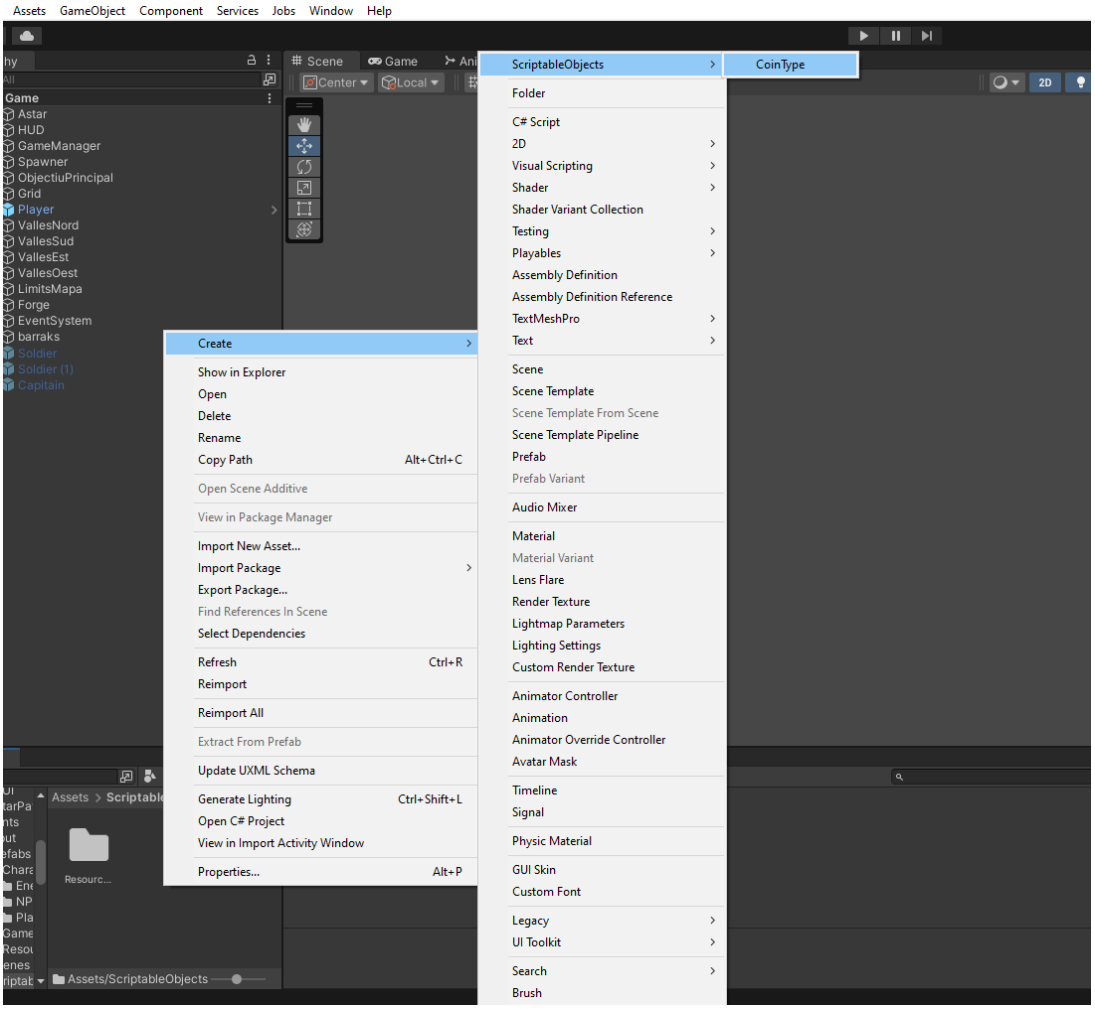


Figura 115: Creació d'un ScriptableObject al menú de Unity

I vaig crear 3 tipus de monedes diferents: SmallCoins, mediumCoins i BigCoins. Cada un d'aquests tenia associat un valor, una imatge i un soroll diferent quan les reculls.

Figura 116: ScriptableObjects de monedes

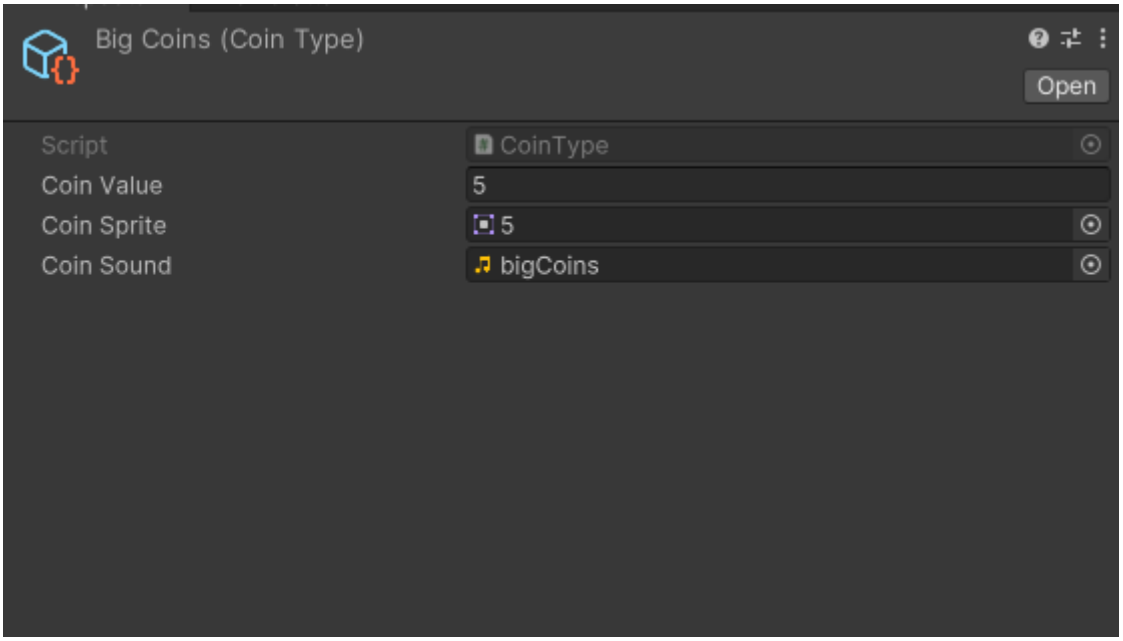


Figura 117: Valors d'un ScriptableObject de tipus de moneda

Gràcies a això amb un únic prefab de monedes es pot modificar el valor d'aquest segons el tipus de ScriptableObject que se li assigni.
Aquest prefab conté:

- Script de la lògica de les monedes
- SpriteRenderer per pintar l'sprite que se li digui.
- boxCollider2D per detectar quan el jugador les toca i que les pugui recollir.
- AudioSource per reproduir el soroll de recollir monedes

Script de lògica té una funció que es crida quan es crea la moneda en el joc. Aquest script li assigna un ScriptableObject segons el tipus d'enemic que s'hagi derrota per saber si ha de ser un objecte moneda amb més o menys valor. Quin sprite pintar i quin so reproduir.

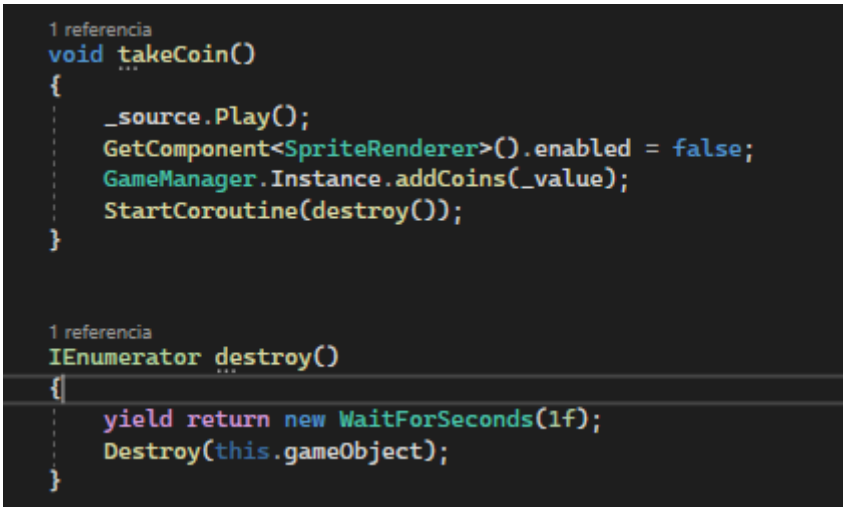
```
1 referencia
public void CreateCoin(int type)
{
    _source = GetComponent<AudioSource>();
    switch (type)
    {
        case 0:

            coinType = Resources.Load<CoinType>("Coins/smallCoins");
            _value = coinType.CoinValue + (int)(GameManager.Instance._lvlDifficulty * 1.5f);
            _sprite = coinType.CoinSprite;
            GetComponent<SpriteRenderer>().sprite = _sprite;
            _source.clip = coinType.CoinSound;

            break;

        case 1:

            coinType = Resources.Load<CoinType>("Coins/mediumCoins");
            _value = coinType.CoinValue + (int)(GameManager.Instance._lvlDifficulty * 1.5f);
            _sprite = coinType.CoinSprite;
            GetComponent<SpriteRenderer>().sprite = _sprite;
            _source.clip = coinType.CoinSound;

            break;

        case 2:

            coinType = Resources.Load<CoinType>("Coins/bigCoins");
            _value = coinType.CoinValue + (int)(GameManager.Instance._lvlDifficulty * 1.5f);
            _sprite = coinType.CoinSprite;
            GetComponent<SpriteRenderer>().sprite = _sprite;
            _source.clip = coinType.CoinSound;

            break;

        default:
            break;
    }
}
```

Figura 118: Funció que es crida al crear una moneda

També detecta quan el jugador passa per sobre les monedes per agafar-les, i es controla que no es puguin agafar més d'un cop. Per detectar quan el jugador passa per sobre fem servir un Tag de Unity. Un cop agafades, es reprodueix el so, enviem al GameManager que ens sumi les monedes agafades i destruïm l'objecte.

```
Mensaje de Unity | 0 referencias
private void OnTriggerEnter2D(Collider2D collision)
{
    if (collision.gameObject.tag == "Player" && !_taked)
    {
        takeCoin();
        _taked = true;
    }
}
```

Figura 119: Funció que controla quan el jugador toca les monedes



```
1 referencia
void takeCoin()
{
    _source.Play();
    GetComponent<SpriteRenderer>().enabled = false;
    GameManager.Instance.addCoins(_value);
    StartCoroutine(destroy());
}


1 referencia
IEnumerator destroy()
{
    yield return new WaitForSeconds(1f);
    Destroy(this.gameObject);
}
```

Figura 120: Funció d'agafar les monedes i destruir l'objecte per pantalla

6.9.3 So

Per els sons de les monedes he volgut destacar si són unes poques, unes quantes, o moltes monedes. De manera que he buscat un so d'agafar monedes que m'agradi i l'he modificat perquè sonés diferent segons si s'agafen unes monedes o unes altres.

6.10 Implementació de les tanques

Les tanques són un prefab com els enemics que es componen de:

- Un objecte pare que conté:
    - SpriteRenderer per pintar l'sprite de la valla.
    - un BoxCollider2D perquè l'enemic pugui detectar la valla i atacar-la.
    - Script de lògica DestructibleObjectsLogic
    - AudioSource per reproduir sons quan li fan mal a la valla.
- Objecte de barra de vida presentat a interficies
    - Canvas espaial
    - Barra de vida

6.10.1 Implementació de l'art

Per fer les tanques he utilitzat les que ja venien fetes en el tileset de *mystic-woods* com el terra.

Però també he fet una versió millorada de les tanques per una posible implementació de millora de muralles que tingués impacte visual i no només lògic, tot i que al final no les he fet servir (figura 119).

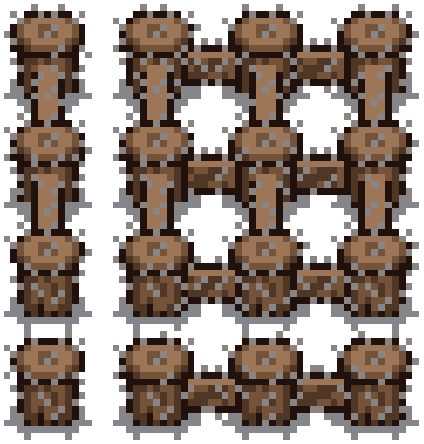


Figura 121: Tanques millorades

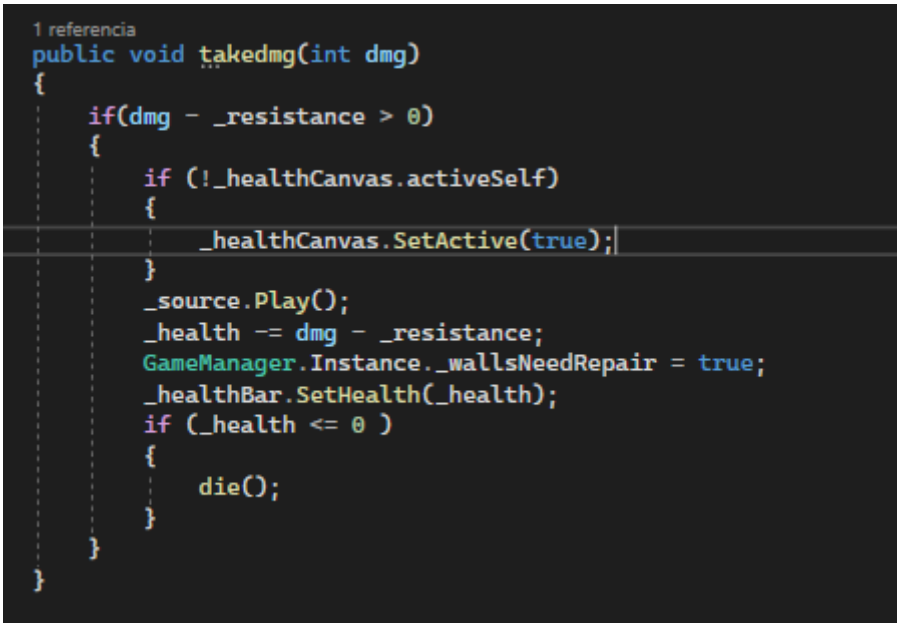
6.10.2 Implementació mecàniques

La lògica de l'script és similar als enemics i player però amb menys característiques i funcions, ja que l'únic que tenen les tanques és: vida, resistència, una funció per quan els hi fan mal, una per quan són destruïdes, una per restaurar-se i una per millorar la resistència quan el jugador compra la millora.

Per tal que el jugador pugui passar a través de les tanques, les he col·locat en un Layer diferent i he fet que el Layer del jugador i el de les tanques no col·lisionin entre ells.

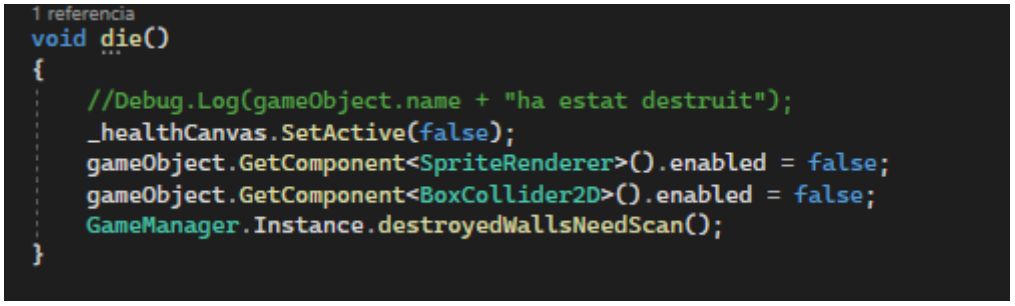
```
[SerializeField]
int _health = 10;
[SerializeField]
int _resistance = 0;

public GameObject _healthCanvas;
public HealthBar _healthBar;

[SerializeField]
AudioSource _source;
```

Figura 122: Característiques de les tanques



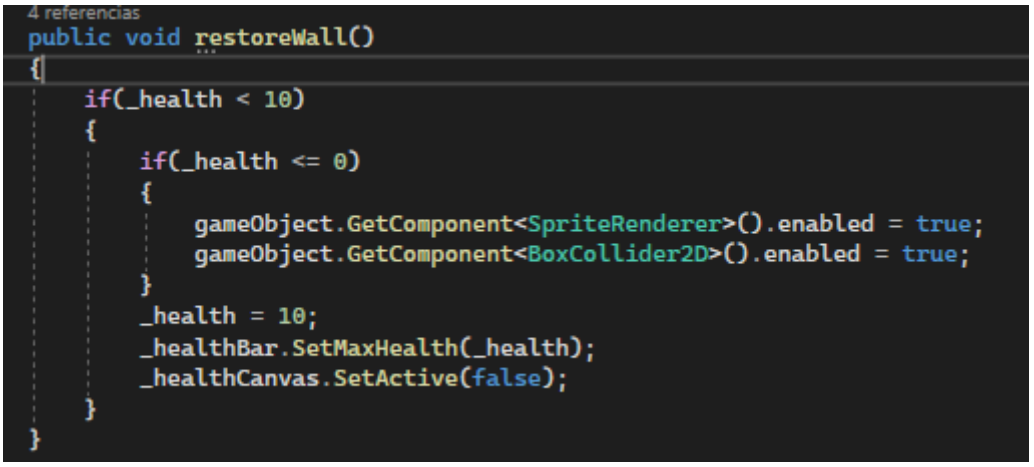
```
1 referencia
public void takedmg(int dmg)
{
    if(dmg - _resistance > 0)
    {
        if (!_healthCanvas.activeSelf)
        {
            _healthCanvas.SetActive(true);
        }
        _source.Play();
        _health -= dmg - _resistance;
        GameManager.Instance._wallsNeedRepair = true;
        _healthBar.SetHealth(_health);
        if (_health <= 0 )
        {
            die();
        }
    }
}
```

Figura 123: Funció per quan una tanca és atacada



```
1 referencia
void die()
{
    //Debug.Log(gameObject.name + "ha estat destruit");
    _healthCanvas.SetActive(false);
    gameObject.GetComponent<SpriteRenderer>().enabled = false;
    gameObject.GetComponent<BoxCollider2D>().enabled = false;
    GameManager.Instance.destroyedWallsNeedScan();
}
```

Figura 124: Funció de destrucció de tanques



```
4 referencias
public void restoreWall()
{
    if(_health < 10)
    {
        if(_health <= 0)
        {
            gameObject.GetComponent<SpriteRenderer>().enabled = true;
            gameObject.GetComponent<BoxCollider2D>().enabled = true;
        }
        _health = 10;
        _healthBar.SetMaxHealth(_health);
        _healthCanvas.SetActive(false);
    }
}
```

Figura 125: Funció per restaurar les tanques

```
4 referencias
public void upgradeWalls()
{
    _resistance += ((int)(GameManager.Instance._lvlOfWalls / 10)) + 2;
}
```

Figura 126: Funció per millorar les tanques

6.10.3 So

Les tanques tenen un so característic quan els hi treuen vida

6.11 Implementació dels edificis interactuables

6.11.1 Edifici Principal

És l'edifici més important i l'objectiu a protegir pel jugador ja que si el destrueixen s'acaba el joc. Està situat al centre del poblat perquè sigui fàcilment localitzable i no s'hagin de fer voltes per arribar-hi.

6.11.1.1 Implementació de l'art

Per la creació d'aquest edifici he agafat un edifici ja fet igual que amb els altres del poblat i l'he modificat perquè sembles mig destruït com els altres. També he agafat una bandera amb animació d'sprites d'internet i l'he modificat perquè encaixés en el projecte. Per tal de marcar que és un edifici important a diferència de la resta. També és l'únic el qual disposa de barra de vida per saber quan serà destruït.

Figura 127: Edifici central

Figura 128: Edifici central amb barra de vida

6.11.1.2 Implementació de les mecàniques

Per la implementació de les mecàniques de l'edifici principal s'ha agafat de referència l'script de les tanques destructibles. Però l'edifici principal no té resistència, ja que es vol donar la possibilitat al jugador de tenir més temps si un enemic comença a destruir aquest edifici. Així que en comptes de la lògica que s'aplica a la resta, a l'edifici central només se li pot treure 1 de vida per segon. Fent que els enemics tardin més a destruir l'edifici i el jugador tingui temps a reaccionar.

Així doncs, la funció de **takedmg()** queda de la següent manera:

```
1 referencia
public void takedmg() // els enemics han de donar 10 cops per derruir l'edifici
{
    if (_canTakeDamage)
    {
        if(!_healthCanvas.activeSelf)
        {
            _healthCanvas.SetActive(true);
        }
        _canTakeDamage = false;
        _source.Play();
        _health -= 1;
        GameManager.Instance._hallTownNeedRepair = true;
        _healthBar.SetHealth(_health);
        StartCoroutine(reableTakeDamage());
        if (_health <= 0)
        {
            die();
        }
    }
}

1 referencia
IEnumerator reableTakeDamage()
{
    yield return new WaitForSeconds(1f);
    _canTakeDamage = true;
}
```

Figura 129: Funció de rebre mal de l'edifici central i Funció de tornar a rebre mal

I quan l'edifici principal és destruït, la seva funció de mort crida el GameOver i el moure càmera del GameManager que abans s'ha comentat.

```
1 referencia
void die()
{
    Debug.Log(" L'ajuntament ha estat destruit!!!!");
    GameManager.Instance.MoveCamera();
    GameManager.Instance.GameOver();
}
```

Figura 130: Funció quan edifici central és destruït.

Tenim la funció que es crida quan el jugador compra la restauració de l'edifici.

```
1 referencia
public void restoreHallTown()
{
    if (_health < 10)
    {
        _health = 10;
        _healthBar.SetMaxHealth(_health);
        _healthCanvas.SetActive(false);
    }
}
```

Figura 131: Funció per restaurar edifici central

També té la lògica de quan mostrar la compra de la restauració de l'edifici central, però l'explicarem en el següent apartat.

6.11.1.3 Implementació de les animacions

L'edifici central disposa d'una animació bàsica de la bandera movent-se per tal que no es vegi tot tant estàtic i donar-li més vida i importància que els altres edificis.

6.11.1.4 So

L'edifici central té el seu propi so per quan un enemic l'està atacant.

6.11.2 Forja

La Forja és l'edifici on el jugador pot comprar les millores per ell, les tanques i la restauració d'aquestes. És un edifici important, ja que permet millorar les característiques del jugador per poder anar avançant d'onades, però oferint cert punt d'estratègia a les decisions del jugador en triar quina millora comprar, perquè no sempre tindrà monedes suficients per comprar-les totes. Està situada a l'esquerra de l'edifici central perquè sigui fàcil de veure i arribar-hi.

6.11.2.1 Implementació de l'art

Per començar he fet una enclusa per tal de deixar clar que allò és una forja. Després he agafat un edifici ja fet i l'he modificat perquè sembli destruït perquè estigui en sintonia amb la resta d'edificis i molt més destruït que molts d'ells perquè així l'enclusa estigui fora l'edifici. També he fet un cartell que recordi a una forja fent un símbol d'una enclusa en ell.

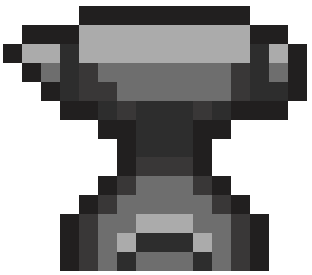


Figura 132: Sprite de l’enclusa

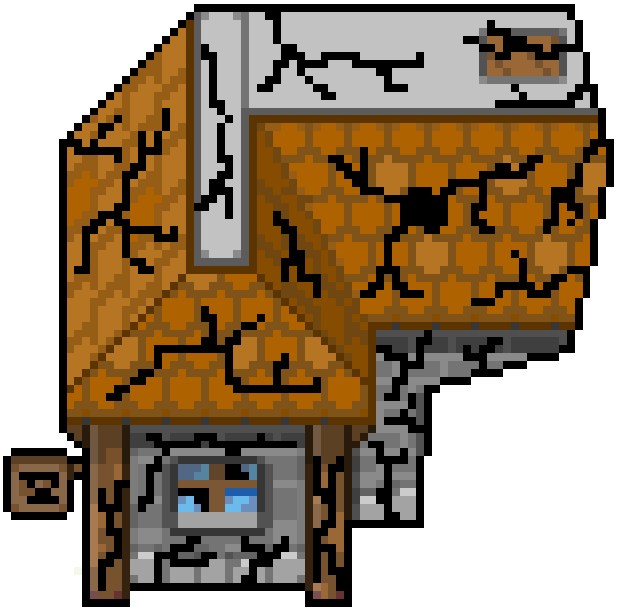


Figura 133: Imatge de la forja

6.11.2.2 Implementació de les mecàniques

La lògica de l’edifici perquè quan el jugador s’acosta a ell, ens ensenyi les millores que hi ha disponibles i els preus. Es fa fent que si el jugador entra en una àrea delimitada per un colider, es detecti el seu Tag i s’ensenyin els botons, i quan el jugador sorti d’aquesta àrea, s’amaguin.

```
Mensaje de Unity | 0 referencias
private void OnTriggerEnter2D(Collider2D collision)
{
    if (collision.gameObject.tag == "Player")
    {
        actiavateUI();
    }
}

Mensaje de Unity | 0 referencias
private void OnTriggerExit2D(Collider2D collision)
{
    if (collision.gameObject.tag == "Player")
    {
        desactiveUI();
    }
}

1 referencia
void actiavateUI()
{
    _canvas.SetActive(true);
}

2 referencias
void desactiveUI()
{
    _canvas.SetActive(false);
}
```

Figura 134: Funcions per ensenyar i amagar millores

6.11.3 Caserna

La caserna permet al jugador reclutar soldats com ajudants per defensar l'edifici central. També li permet curar-se si ha perdut vida. Els soldats són unitats entrenades i poderoses, per tant són cars de contractar des del principi i puja el preu segons anem avançant en la partida. Està situada a la dreta de l'edifici central perquè sigui fàcil de veure i arribar-hi.

6.11.3.1 Implementació de l'art

Per fer la caserna he agafat el mateix edifici de la forja, però l'he destrossat menys per mostrar que com era el lloc on el soldats resideixen, ha pogut aguantar una mica més la embestida inicial. La caserna porta una espasa en el cartell de l'edifici per identificar-la correctament.

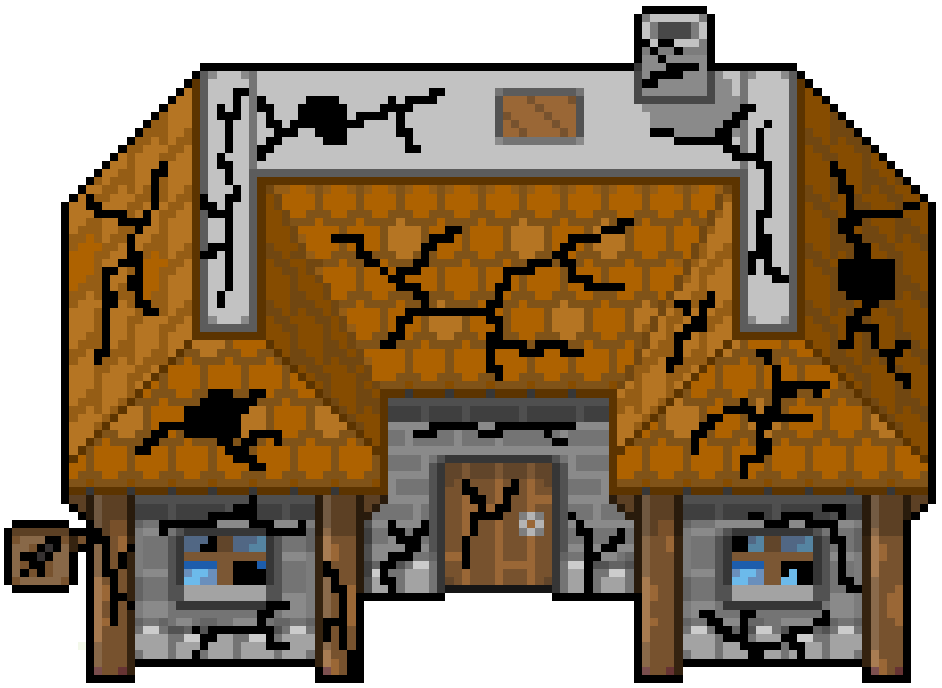


Figura 135: Imatge de la caserna

6.11.3.2 Implementació de les mecàniques

La lògica de la caserna és la mateixa que la de la forja per quan s'acosta el jugador a ella.

6.12 Implementació de NPCs aliats

Per ajudar al jugador a defensar l'edifici principal, es poden reclutar soldats que faran guardia davant d'aquest. Es poden diferenciar 2 tipus de soldats: el soldat bàsic i el capità.

Són molt similars als enemics ja que tenen:

- Un objecte pare que conté:
    - L'script de lògica dels soldats
    - AudioSource per reproduir alguns sons com quan el soldat pren mal.
    - CircleCollider2D per tal de controlar quan es troba amb un enemic.
- La barra de vida explicada a l'apartat anterior 6.blabla:
    - Canvas espaial
    - barra de vida
- SoldierFX:
    - Conte el SpriteRenderer que ens permet pintar el soldat per pantalla
    - AudioSource per reproduir alguns sons com quan el soldat fa un atac.

    - Animator per controlar les animacions del soldat.
- Detector soldiers:
    - Script de lògica del detector dels soldats.

Aquests elements els tenen tant els soldats com els capìtans.

6.12.1 Implementació de l'art

Per fer el soldat bàsic he agafat de referència al jugador i li he afegit una armadura i casc. Per al capità he aprofitat el soldat i li he afegit una pluma al casc per mostrar un rang superior.

Figura 136: Spritesheet dels soldats

Figura 137: SpriteSheet del capità

6.12.2 Implementació de mecàniques

Per implementar les mecàniques he seguit l'estructura que he fet pels enemics i el jugador, però sense implementar el moviment. Els soldats els volem fent guàrdia a l'edifici central i que siguin l'última línia de defensa en cas que els enemics entrin.

Quan un soldat o capità és reclutat es crida la funció **resetValues()** per assegurar que tots els seus valors s'han reiniciat. Aquesta reseteja les animacions, posa la vida del soldat en 10, actualitza la seva barra de vida, posa l'atac i la resistència segons pertoqui per nivell, etc.

```
2 referencias
public void resetValues()
{
    _animator.SetFloat("Horizontal", 0);
    _animator.SetFloat("Vertical", 0);
    _animator.SetTrigger("Reset");

    _health = 10;
    _damage = _initialDmg + (int)(GameManager.Instance._lvlDifficulty * 2f);
    _resistance = _initialRes + (int)(GameManager.Instance._lvlDifficulty * 1.5f);
    _died = false;

    _detector.enabled = true;

    _healthCanvas.SetActive(false);
    _healthBar.SetMaxHealth(_health);

    _hurtSource.clip = _hurtClip;
    _swordSource.clip = _swordClip;

}
```

Figura 138: Funció de reiniciar els soldats

Igual que els enemics, hi ha la funció que permet actualitzar els valors de les animacions. Mentre que en l'enemic aquesta funció es crida des del seu script de moviment, els soldats com que no es mouen la crida el seu detector per actualitzar cap on han de mirar.

```
1 referencia
public void updateAnimations(Vector2 direction)
{
    if (!_died)
    {
        _animator.SetFloat("Horizontal", direction.x);
        _animator.SetFloat("Vertical", direction.y);

        if (direction.x < 0)
        {
            _soldierFX.transform.localScale = new Vector3(-1, 1, 1);
        }
        else if (direction.x > 0)
        {
            _soldierFX.transform.localScale = new Vector3(1, 1, 1);
        }
    }
}
```

Figura 139: Funció d'actualitzar animacions soldats

En el FixedUpdate es mira si pot atacar a algún enemic i, si està al seu abast, l'ataca. Ho fa igual que l'enemic i el jugador amb el Physics2D.OverlapCircleAll() explicat a la secció del jugador.

```
Mensaje de Unity | 0 referencias
private void FixedUpdate()
{
    if (!_died && !GameManager.Instance._endOfGame)
    {
        Collider2D[] hitEnemies = Physics2D.OverlapCircleAll(transform.position, _attackRange, _attackableLayers);
        if(hitEnemies.Length > 0)
        {
            if (_attackTimeCounter >= _timeBtwAttacks)
            {
                _animator.SetTrigger("Attack");
                _swordSource.Play();

                foreach (Collider2D hit in hitEnemies)
                {
                    hit.gameObject.GetComponent<EnemyLogic>().takeDamage(_damage);
                }

                _attackTimeCounter = 0f;
            }
        }
    }
}
```

Figura 140: Lògica d'atac dels soldats

Igual que tots els personatges, té una funció per rebre mal. Però en els soldats i capitans s'ha modificat ja que al ser personatges amb molt bones característiques, només un tipus d'enemic podria arribar a fer-lis mal, i no era gaire realista. Per això hem modificat com reben el mal.

Els soldats miren quant de mal total han rebut en l'últim mig segon, i si aquest és superior a la seva resistència, reben mal. Per posat un exemple: si el soldat té 8 de resistència, i ve 1 Goblin berserker amb un mal de 6, guanyara el soldat i el Goblin no li farà mal. En canvi, si venen 2 Goblin Berserker que entre els 2 fan 12 de mal, com que la resistència del soldat és de 8, li faran 4 de mal. Això fa que els soldats siguin molt forts 1 vs 1 però puguin ser derrotats si es troben contra varis enemics a la vegada.

```
1 referencia
public void takedmg(int dmg)
{
    if (!_died)
    {
        _damageRecived+=dmg;
        if(!_damaged)
        {
            StartCoroutine(damageTaked());
            _damaged = true;
        }
    }
}
```

Figura 141: Funció de quan s'ataca als soldats

```
1 referencia
IEnumerator damageTaked()
{
    yield return new WaitForSeconds(0.5f);

    if (_damageRecived - _resistance > 0)
    {
        if (!_healthCanvas.activeSelf)
        {
            _healthCanvas.SetActive(true);
        }

        _health -= _damageRecived - _resistance;
        _healthBar.SetHealth(_health);
        _hurtSource.clip = _hurtClip;
        _hurtSource.Play();

        if (_health <= 0)
        {
            die();
        }
    }
    else
    {
        _hurtSource.clip = _noHurtDamageClip;
        _hurtSource.Play();
    }
    _damageRecived = 0;
    _damaged = false;
}
```

Figura 142: Funció de patir mal dels soldats

Igual que la resta de personatges, si la seva vida arriba a 0 es mor. Però amb els soldats, per no anar creant i destruint aquests soldats, els desactivem i abans de tornar-los a activar quan el jugador els torni a comprar, es fa servir la funció **resetValues()** abans mencionada.

```
1 referencia
void die()
{
    //animació de morir i dehabilitar objecte
    GetComponent<CircleCollider2D>().enabled = false;
    _animator.SetTrigger("Die");
    _hurtSource.clip = _deathClip;
    _hurtSource.Play();
    _died = true;
    _detector.enabled = false;
    StartCoroutine(destroy());

}

1 referencia
IEnumerator destroy() // disable en el cas dels soldats
{
    yield return new WaitForSeconds(2f);
    _healthCanvas.SetActive(false);
    this.gameObject.SetActive(false);
}
```

Figura 143: Funció de morir i desapareixer de pantalla dels soldats

També té una funció per actualitzar les característiques en cas que el soldat estigui viu i augmenti el nivell de dificultat. Fent que els soldats sempre siguin poderosos encara que avanci la partida.

```
2 referencias
public void UpgradeSoldiers()
{
    _damage = _initialDmg + (int)(GameManager.Instance._lvlDifficulty * 2f);
    _resistance = _initialRes + (int)(GameManager.Instance._lvlDifficulty * 1.5f);
}
```

Figura 144: Funció per millorar soldats

6.12.3 Implementació de les animacions

Molt similars a les dels enemics i el jugador, però no es faran servir les del moviment, ja que no es necessitaran i farem servir l'element "*Exit*" per tal que torni a l'inici, l'element "*Entry*", i es torni al flux normal abans d'entrar a l'animació de mort.

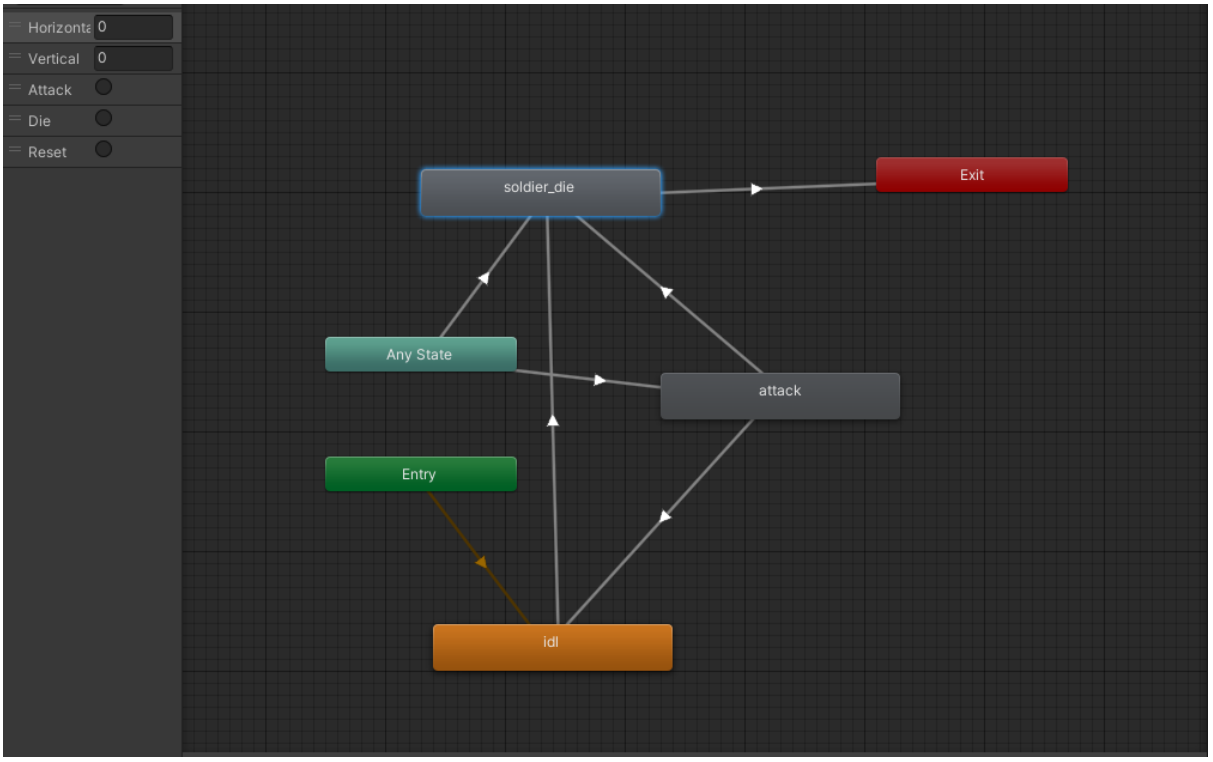


Figura 145: Diagrama d'animacions de soldats

Seguirem fent ús dels blend Tree's per simplificar animacions i tenir-les ben encapsulades.

6.12.4 So

Els sons dels soldats són una mescla entre alguns dels enemics i alguns del jugador. Però tenen la mateixa quantitat de sons que els altres 2.

6.13 Implementació de controls per mòbil

Com que es va fer servir el Input System de Unity i el paquet de Joysticks del Asset Store, ens vam poder centrar en fer totes les mecàniques del joc i vam poder deixar l'implementació dels controls per mòbils com un dels últims passos.

6.13.1 Implementació de l'art

Per la creació de l'art dels controls del mòbil he fet un botó d'atacar i un joystick per moure's seguint la mateixa estètica que la resta del joc.

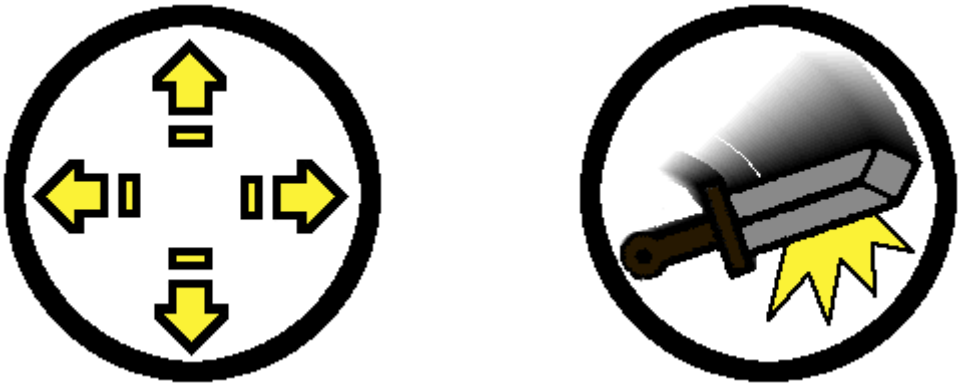


Figura 146: Sprites de controls per mòbil

6.13.2 Implementació de lògica

Per la implementació del controls he creat un botó de Unity normal per l'atac, que té de referencia a la funció **OnAttack()** anomenada anteriorment quan parlavem del Input System en l'apartat del jugador.

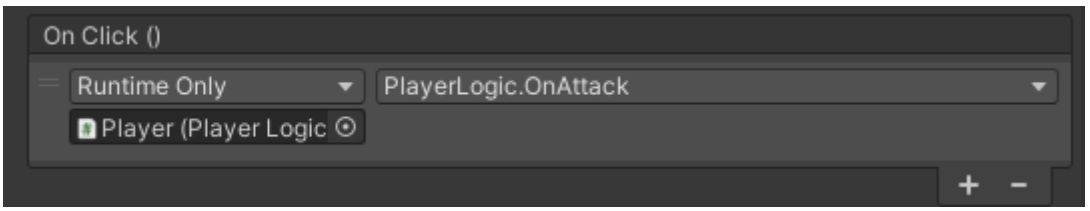


Figura 147: Referència del botó a la funció d'atac

Per el joystick he creat un script que agafa els inputs del joystick i els passa al player a la funció **OnMobileMove()** el qual rep un Vector2 amb els inputs i el moviment es fa amb la lògica explicada anteriorment a l'apartat del jugador.

```
public class MobileControlsAdapter : MonoBehaviour
{
    [SerializeField] FixedJoystick joystick;
    [SerializeField] PlayerLogic _player;

    Mensaje de Unity | 0 referencias
    private void FixedUpdate()
    {
        Vector2 input = new Vector2(joystick.Horizontal, joystick.Vertical);

        _player.OnMobileMove(input);

    }
}
```

Figura 148: Controls de mòbil per moviment

```
1 referencia
public void OnMobileMove(Vector2 input)
{
    _movementInput = input;
}
```

Figura 149: Funció per passar l'input del mòbil al moviment

6.14 Implementació de re-jugabilitat

Per garantir una re-jugabilitat he implementat un contador d'onades per tal de saber quin és el teu record i intentar superar-te.

Per fer això he fet ús del PlayerPrefs. Un mecanisme propi de Unity per guardar certs paràmetres. Tot i que normalment el PlayerPrefs es fa servir per guardar certs paràmetres de configuració, com el volum del joc, resolució de la pantalla triada per l'usuari, etc. Com

que el nostre projecte només tenia que guardar una variable de enters de quina és la major onada a la que has arribat, he descartat fer un sistema de guardat en local a través de JSON o similar. Ja que per una simple variable no era necessari.
També s'ha descartat el guardat en línia ja que el nostre producte no fa servir la connexió a internet.

Així doncs, per fer ús del sistema de PlayerPrefs l'únic que s'ha de fer és cridar-ho en el moment en què necessitem consultar o escriure sobre aquesta variable.
Com en el menú principal on es veu la millor onada, durant el joc quan acabem la partida per veure si s'ha superat, i al Game Over per veure el resultat tant de l'onada actual com la millor.

```
Mensaje de Unity | 0 referencias
private void Awake()
{
    if (!PlayerPrefs.HasKey("BestWabe"))
    {
        PlayerPrefs.SetInt("BestWabe", 0);
    }
}
```

Figura 150: Menú principal mira si existeix millor onada

```
if(PlayerPrefs.GetInt("BestWabe") < _waveCount) // actualitzem puntuació més alta si l'hem aconseguit
{
    PlayerPrefs.SetInt("BestWabe", _waveCount);
}
PlayerPrefs.SetInt("WavesEnded", _waveCount); // i ens guardem la puntuació que hem fet aquesta partida
```

Figura 151: Quan acaba el joc, s'actualitza la millor puntuació i es guarda actual

```
void Start()
{
    _wavesText.text = PlayerPrefs.GetInt("WavesEnded").ToString("D2");
    _bestWave.text = PlayerPrefs.GetInt("BestWabe").ToString("D2");
}
```

Figura 152: Game Over ensenya les onades per pantalla

6.15 Traducció del joc

Per traduir el joc he utilitzat l'eina dedicada de Unity que et permet construir Locale's fàcilment i sense necessitat de codi. Gràcies a aquest paquet, en els textos només els he d'assignar l'identificador corresponent perquè canviïn entre els idiomes seleccionats.

Per aquest projecte ja es va fer un plantejament inicial on no hi hagués gaire text escrit per facilitar-ne la traducció. D'aquesta manera, només he hagut de configurar uns 5-6 textos utilitzant el Localization de Unity perquè el joc fos traduït.

De moment he fet que el joc estigui disponible en Anglès i Català. Però es podrien afegir més idiomes de manera molt simple.

L'idioma és selecciona automàticament segons l'idioma del sistema on s'executa el joc. Tot i que en el menú principal hi ha 2 botons amb les banderes de l'idioma perquè es pugui canviar en qualsevol moment.

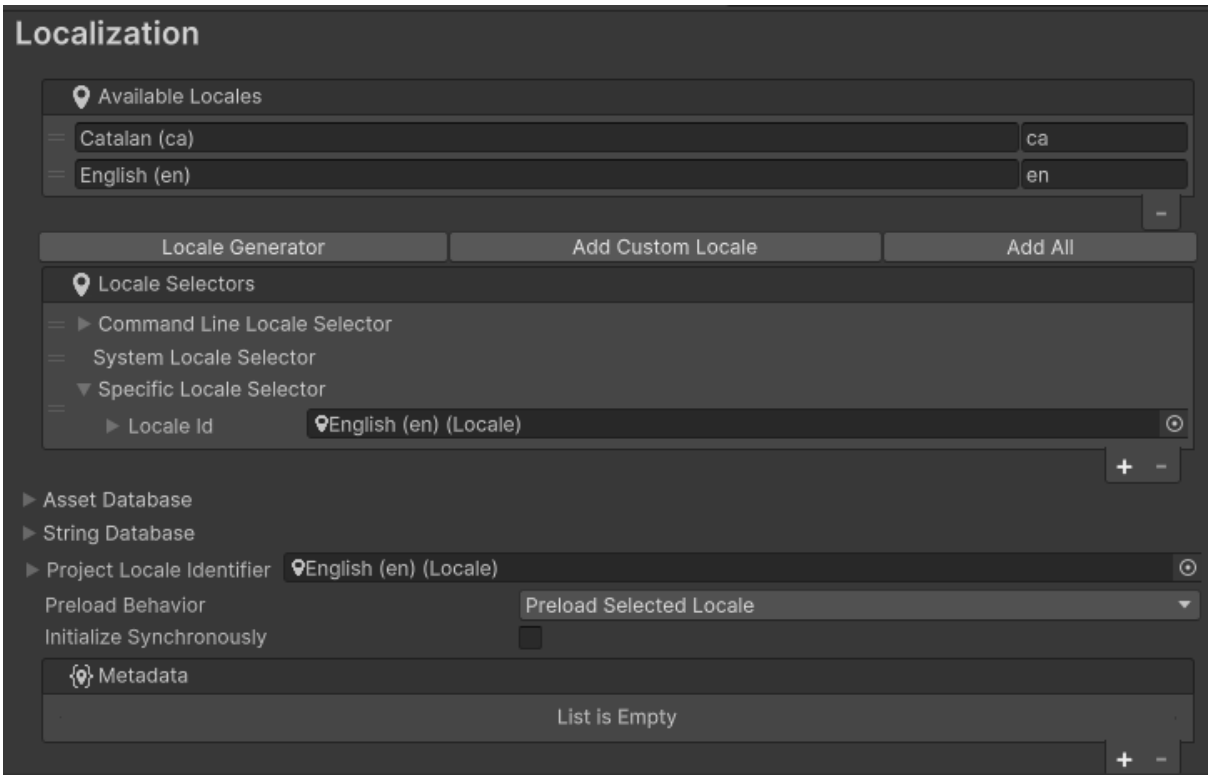


Figura 153: Localization de Unity

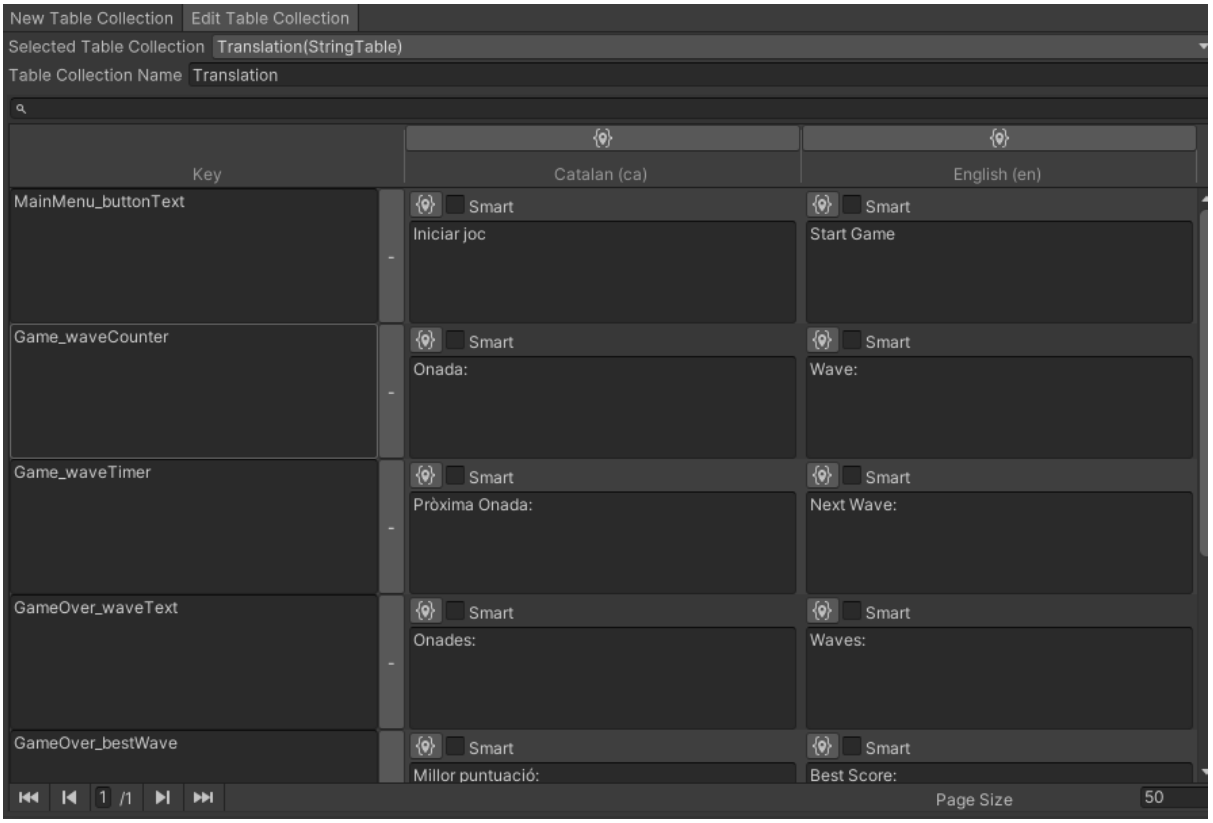


Figura 154: Taula d'strings per les traduccions

Figura 155: Menú principal amb les banderes a baix a la dreta per canviar d'idioma

Figura 156: Game Over en Anglès

Figura 157: Game Over en Català

6.16 Proves realitzades

Durant el desenvolupament del joc, i com s'ha explicat a la planificació. Per cada apartat que s'ha desenvolupat s'han fet diverses proves de funcionament. Algunes més unitàries com comprovar que certes funcions tinguessin el resultat esperat, mentre que altres més general fent alguna partida quan el projecte estava prou avançat com per poder provar certes mecàniques concretes o una partida sencera. Durant les proves, moltes vegades me n'he adonat que el comportament que havia decidit no era del tot correcte amb la jugabilitat esperada, i a vegades he fet alguns petits canvis per tal d'ajustar el funcionament. Com per exemple: quan i com els enemics pujaven les seves característiques segons avançaven les onades, la quantitat de monedes que deixaven anar, els preus de les millores al començament de la partida i com anaven evolucionant amb tota l'economia, etc.

He aprofitat que era per dispositius mòbils per deixar-ho provar a diversos companys i amics perquè em donessin els seus feedbacks de certs aspectes, o si creien que es podia millorar algun en concret.

També es van fer proves a diferents dispositius mòbils per mirar que a diferents dispositius Android funcionés el joc.

7. Resultats

7.1. Legislació i normativa vigent

El joc desenvolupat no presenta problemes a nivell legislatius. No es desa informació personal de l'usuari final, per tant no aplica en cap situació la LOPD (Llei Orgànica de Protecció de Dades).

Pel que fa els aspectes de Copyright o drets d'Autor, tots els recursos utilitzats en el projecte són de creació propia o de llicències de lliure ús i distribució. Totes les eines de creació de contingut utilitzades també estan lliure de drets d'ús, menys Unity. En el cas de Unity, si el producte creat genera un benedific brut de més de 100.00$, s'hauria d'adquirir la llicència de pagament tal com s'indica en els seus termes i condicions d'ús.

7.2 PEGI

PEGI, o Pan European Game Information, és un sistema de classificació de contingut per a videojocs. Utilitza icones per indicar l'edat recomanada per a un joc i per identificar contingut específic, com ara violència, drogues, sexe o paraulotes.

PEGI va ser creat per la Interactive Software Federation of Europe (ISFE) l'abril de 2003. Actualment, es fa servir a 16 països europeus, inclosa la Unió Europea.

El sistema PEGI es basa en dos conjunts d'icones: un per a l'edat recomanada i un altre per al contingut específic.

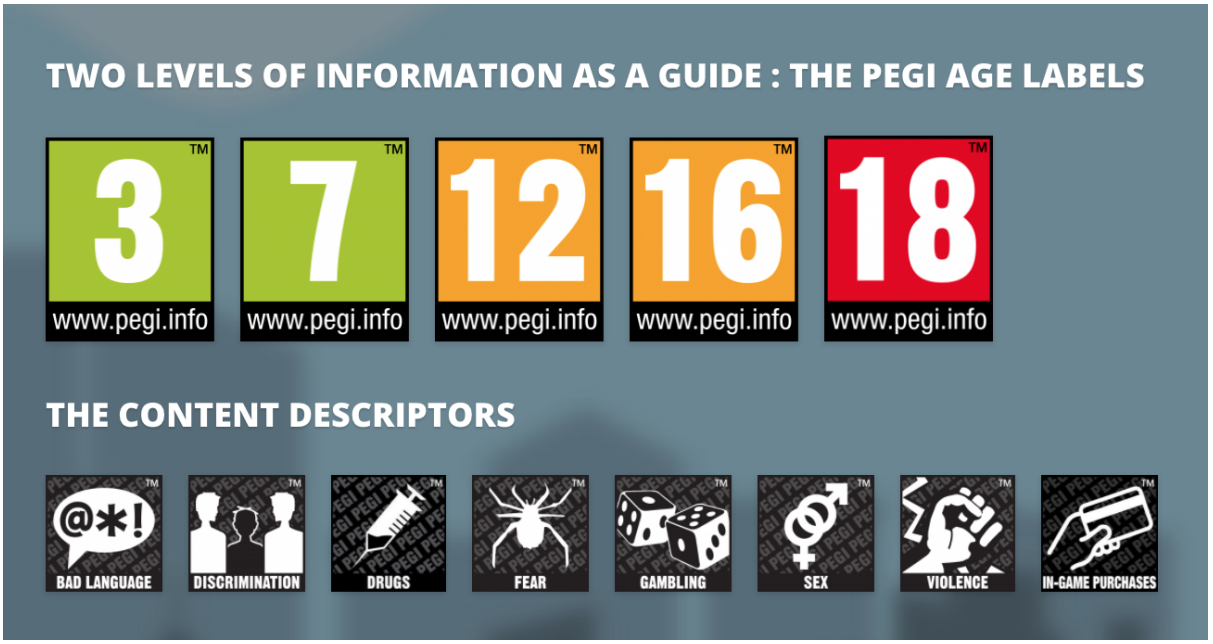


Figura 158: Normativa PEGI

En el cas d'aquest joc, com que sí que hi ha violència i sorolls estridents no es podria considerar PEGI 3, però al ser d'uns personatges fantàstics, no ser d'un estil gràfic realista i la violència no és detallada es podria qualificar de PEGI 7, tot i que sí que hauria de portar l'identificador de violència.

Figura 159: Normativa PEGI al nostre producte

7.3 Resultat final

En aquesta secció podreu veure imatges de les diferents escenes del joc com a resultat final del projecte. En ella hi ha: El menú principal, l'escena del joc en diferents moments i l'escena de Game Over.

A part de les imatges, també podeu visualitzar un vídeo on s'ensenya una partida des del menú principal fins arribar al Game Over. El podreu trobar a: https://youtu.be/WTB36axjADY

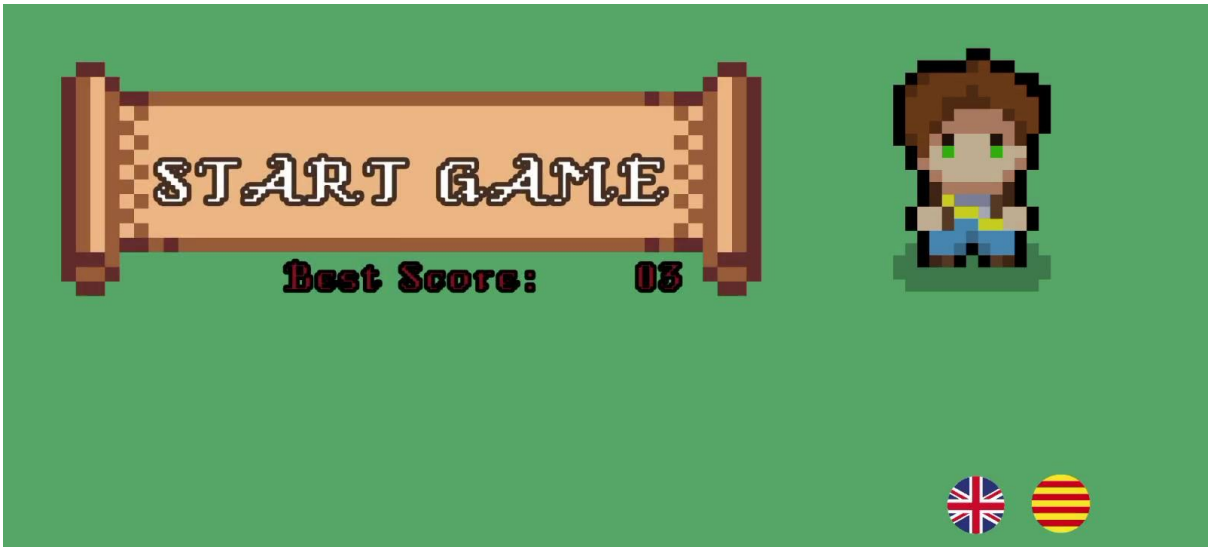


Figura 160: Resultats menú principal en anglès

Figura 161: Resultats menú principal en català

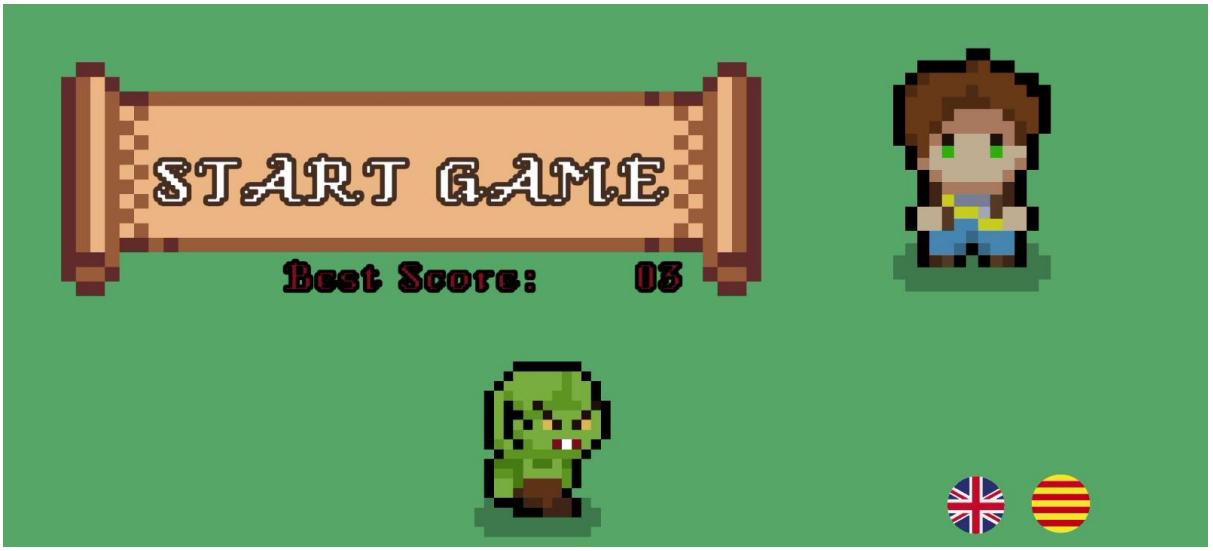


Figura 162: Resultats menú principal en anglès 2

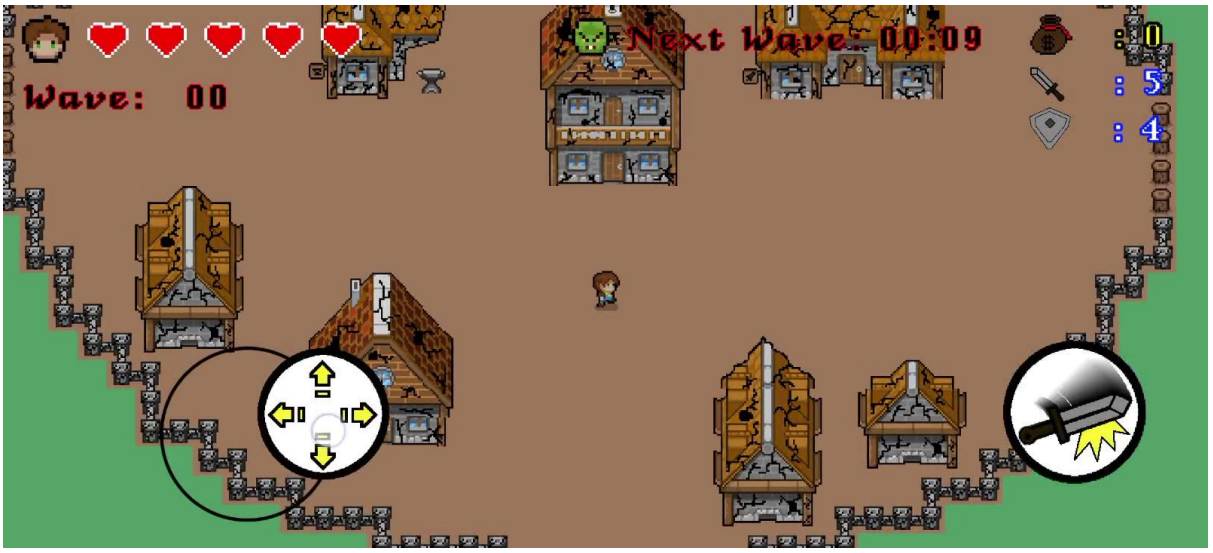


Figura 163: Resultats durant el joc

Figura 164: Resultats durant el joc 2

Figura 165: Resultats durant el joc 3

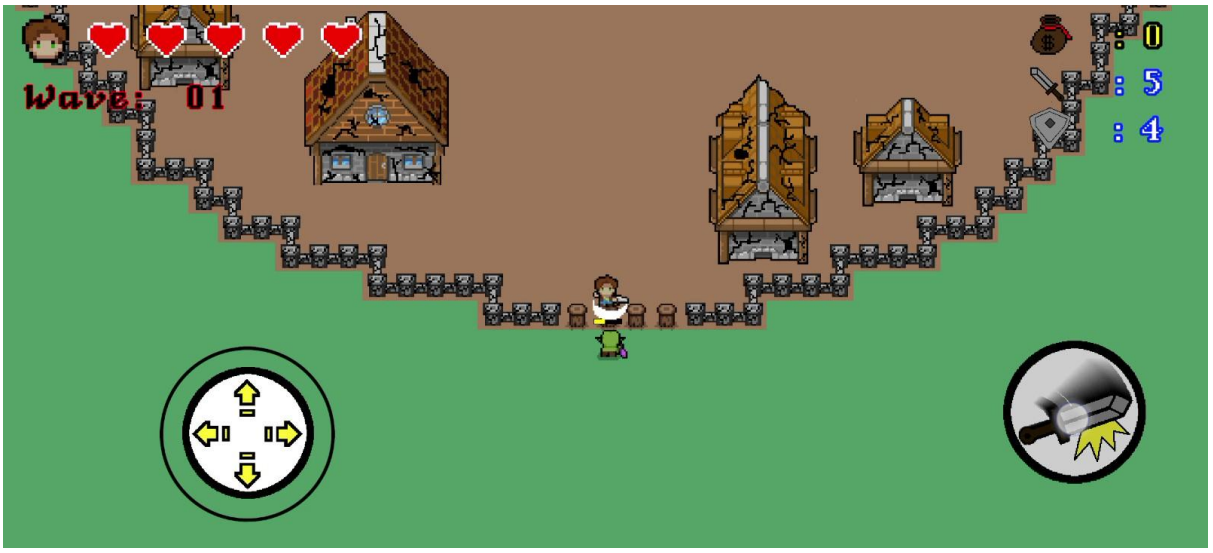


Figura 166: Resultats durant el joc 4

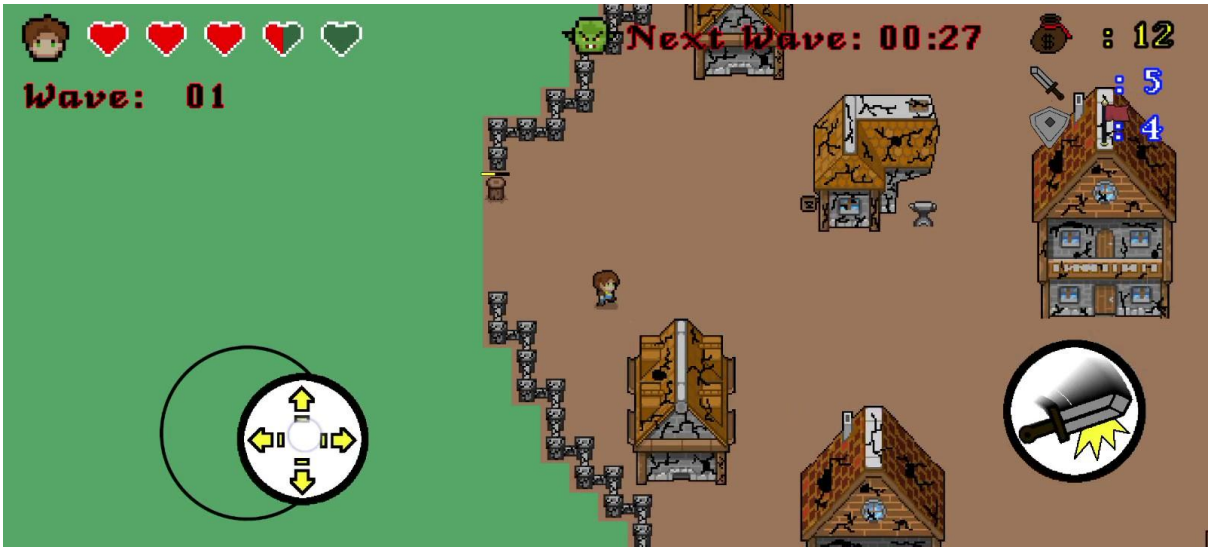


Figura 167: Resultats durant el joc 5

Figura 168: Resultats durant el joc 6

Figura 169: Resultats durant el joc 7

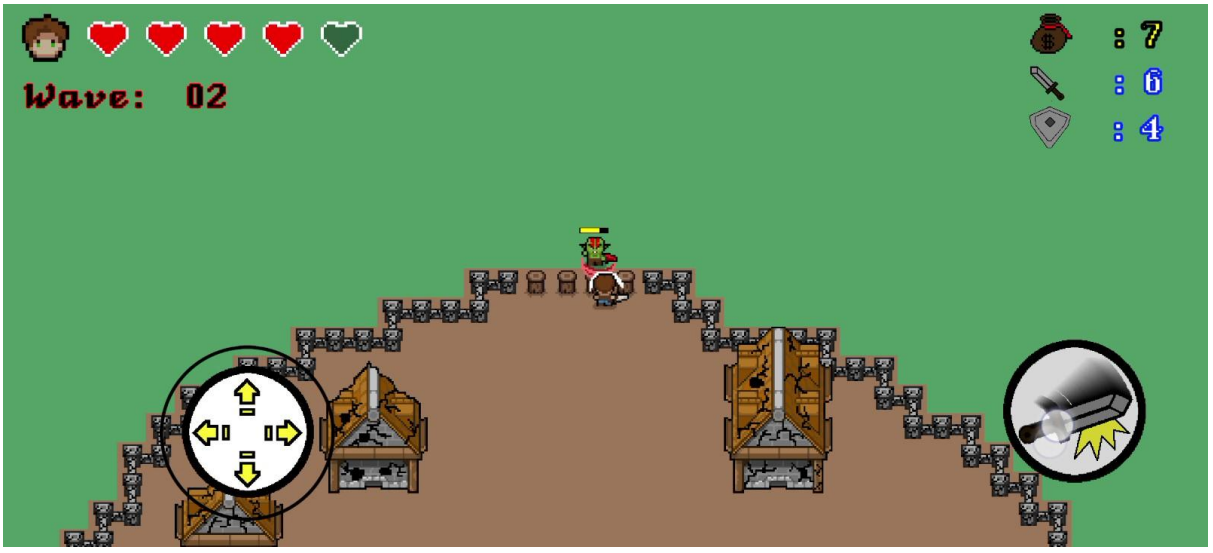


Figura 170: Resultats durant el joc 8

Figura 171: Resultats durant el joc 9

Figura 172: Resultats durant el joc 10

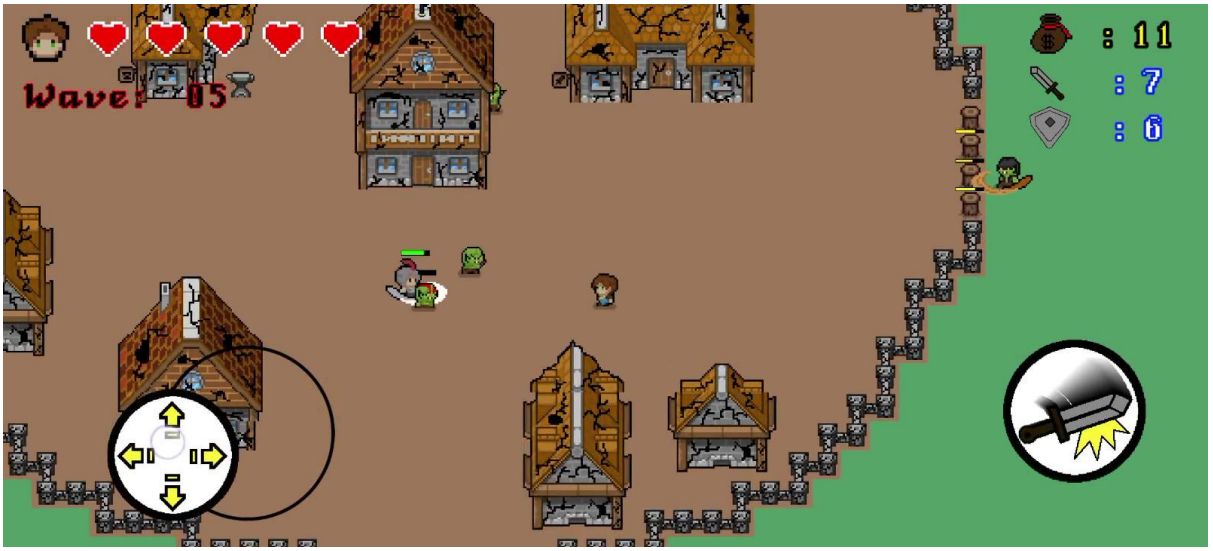


Figura 173: Resultats durant el joc 11

Figura 174: Resultats durant el joc 12

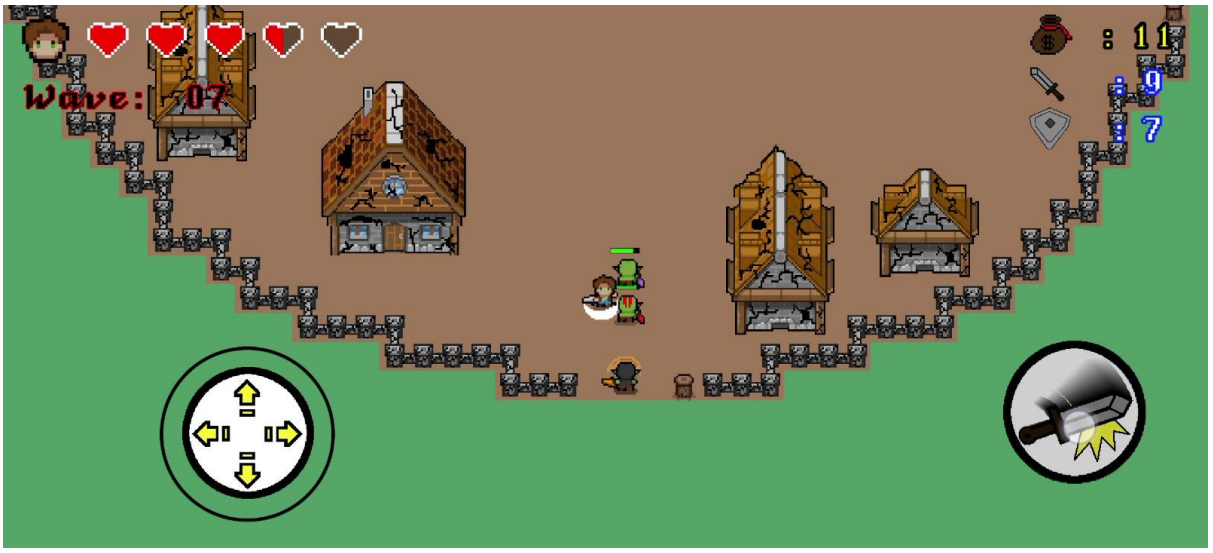


Figura 175: Resultats durant el joc 13

Figura 176: Resultats durant el joc 14

Figura 177: Resultats durant el joc 15

Figura 178: Resultats Game Over

8. Conclusions

8.1 Valoració del treball

Amb l'elaboració d'aquest projecte he pogut aprofundir en alguns camps on tenia curiositat per saber com es feien certes coses de la millor manera. Un exemple clar són les animacions. Anteriorment, havia treballat amb les animacions de Unity, però sempre a un nivell molt alt, i sense utilitzar els Blend Trees. Amb aquest projecte vaig poder entendre el seu funcionament i quan era bo usar-los i quan no. Com fer-los funcionar i que les animacions quedessin ben quadrades i amb ordre i lògica segons el que estava passant.

Un altre aspecte és el de generació d'enemics per onades. Tot i que altres vegades havia fet onades d'enemics mai havia fet un generador que tingués un pressupost per poder generar enemics. I tot i que no m'he endinsat gaire més en aquest món i no he aprofundit en les "*enemy pools*" o piscines d'enemics, les quals fan molt més òptima la generació d'aquests. Ara que ja tinc una base sòlida sobre la generació d'onades seria un dels següents punts a millorar i experimentar.

Tot i que el resultat del projecte és l'esperat, sí que hi ha certes parts que no s'han acabat de fer com en un principi estaven pensades. Una d'elles és la creació d'un algorisme propi de A* per la lògica de buscar camins. Tot i que fer aquest algorisme portaria molta feina, sí que un dels canvis que m'agradarien seria estudiar més en detall el projecte que he fet servir finalment com a substitució per tal d'aconseguir un resultat més òptim per a quan s'han de trobar diversos camins possibles i com es fan els diferents escanejos de mapa quan aquest canvia, per tal que fos tot molt menys costos a nivell de rendiment. Ja que amb l'aprofundiment d'aquest funcionament i un sistema més òptim es podria millorar la intel·ligència artificial dels enemics perquè tinguessin moltes opcions possibles. Com tenir en compte què estan fent els altres enemics per muntar estratègies més complexes d'atac.

Tot i això, el comportament dels enemics és correcte i tot i que no tenen molta estratègia d'atac, sí que s'ha aconseguit una jugabilitat interessant on els goblins bàsics no suposen un problema, però quan veus aparèixer un berserker o un tank poses nerviós al jugador en haver de pensar alguna estratègia segons a quin dels 2 s'està enfrontant.

Per tant, puc dir que estic satisfet amb el resultat final del projecte i els seus components.

8.2 Desviacions de la planificació original

Tot i que al princi els temps es van seguir correctament, i aquest projecte estava pensat per entregar-se a la convocatòria de gener-febrer de 2023. Tot i això, per temes laborals, vaig haver de parar el desenvolupament durant un temps, fent que finalment s'acabés el projecte per la convocatòria de setembre.

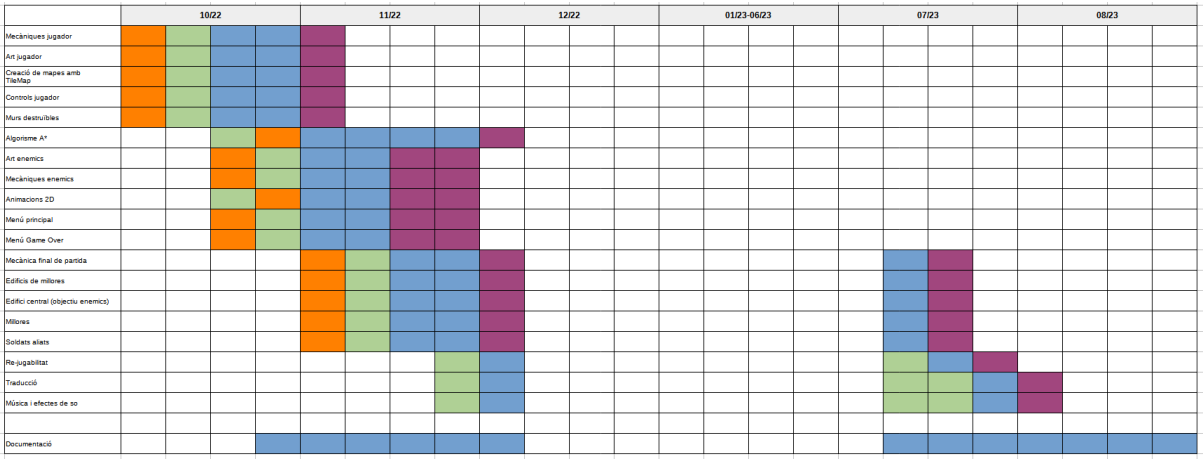


Figura 179: Diagrama de Gantt resultant

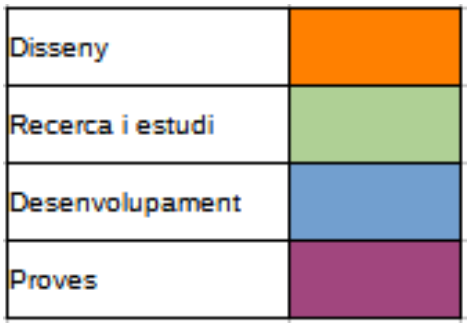


Figura 180: Llegenda diagrama de Gantt

A part d'aquesta gran desviació en la planificació, hi va haver una desviació en el desenvolupament de l'algorisme A* i la lògica del enemics amb aquest com ja s'ha explicat en l'apartat d'implementació. També hi van haver certes modificacions de certes idees inicials que es van canviar perquè no encaixaven amb el resultat i es volia buscar un bon acabat i un millor producte.

8.3 Recull de tasques finalitzades

Llista de tasques finalitzades:

🟥 Prioritat Alta 🟪Prioritat Mitja 🟨Prioritat Baixa

✔ Fet ✔ Adaptació

- Disseny:
    1. Disseny del jugador 🟥 ✔
    2. Disseny de mecàniques de jugador 🟥 ✔
    3. Disseny d'enemics 🟥 ✔
    4. Disseny del mapa 🟥 ✔
    5. Disseny del HUD i menús 🟥 ✔
    6. Disseny d'aliats 🟨 ✔
    7. Disseny de millores 🟥 ✔

- Programació:
    8. Desenvolupament mecàniques jugador 🟥 ✔
    9. Implementació animacions jugador 🟪 ✔
    10. Implementació sons de jugador 🟨 ✔
    11. Creació del mapa amb TileMap de Unity 🟥 ✔
    12. Controls del jugador 🟥 ✔
    13. Implementació de murs destruibles 🟪 ✔
    14. Implementació dels sons de murs destruibles 🟨 ✔
    15. Creació Algorisme A* 🟪 ✔
    16. Desenvolupament mecàniques enemics 🟥 ✔
    17. Implementació d'animacions enemics 🟪 ✔
    18. Implementació sons enemics 🟨 ✔
    19. Implementació menu principal 🟥 ✔
    20. Implementació menú Game Over 🟥 ✔
    21. Implementació final de partida 🟥 ✔
    22. Implementació d'edificis interaccionales 🟪 ✔
    23. Implementació de recursos 🟥 ✔
    24. Implementació de millores de característiques 🟪 ✔
    25. Implementació soldats 🟨 ✔
    26. Implementació re-jugabilitat 🟨 ✔
    27. Implementació traducció 🟨 ✔
    28. Implementació músiques i altres efectes de so 🟨 ✔

- Art:
    29. Creació de sprites jugador 🟥 ✔
    30. Creació de Tilesets pel mapa 🟪 ✔
    31. Creació art de murs destruibles 🟪 ✔
    32. Creació sprites enemics 🟥 ✔
    33. Creació botons per menús 🟪 ✔
    34. Creació botons per millores 🟪 ✔
    35. Creació elements HUD 🟪 ✔
    36. Creació art edificis interaccionales 🟪 ✔
    37. Creació art de recursos (monedes) 🟪 ✔
    38. Creació sprites soldats 🟨 ✔

9. Treball futur

Actualment el joc està compost per un nivell infinit d'onades, així que hi hauria la possibilitat de seguir desenvolupament aquest projecte de varies maneres.

Una seria la continuïtat d'aquest mode infinit afegint:

- **Enemics més variats**, amb habilitats especials o comportaments especials. Com per exemple la capacitat de volar i passar per sobre les barreres, la capacitat de nedar, enemics que explotessin al arribar al seu objectiu fent un gran mal, etc. Aquests enemics nous podrien ser varietats dels enemics que hi ha ara o totalment nous amb diferents races i aspectes.

- **Més ajudants** soldats o ajudants diferents amb noves mecàniques i que aquests tinguessin més possibilitats de jugabilitat, fent que es poguessin moure, perseguir enemics, mantenir certes posicions, patrullar, etc.

- **Més millores** tant pel jugador com per l'entorn. Per exemple que les tanques poguessin evolucionar a tanques recobertes de filferro punxant o alguna defensa extra que fes que els enemics al xocar contra elles es fessin mal. O afegir algun tipus de torre que ataqui als enemics a distància o voladors.

- **Més mapes** per tenir varietat on jugar aquestes onades infinites. Amb diferents ambientacions climes i potser efectes especials com per exemple en un mapa on hi hagués gel, els personatges petinessin als moure's.

Sumat a tot a l'anterior, una altra possibilitat seria desenvolupar un **mode campanya o història**, en el qual ens endinséssim en desenvolupar la història que passa en aquest joc, perquè un gran exèrcit ha assolat el poble on vivies, què passa després, etc.

També m'agradaria poder millorar certs aspectes ja fets, com l'algorisme A* implementat, que fos més òptim i eficient.

10. Bibliografia

- Ankousse26. (2022). flag animation [PNG]. 300x60 píxels. [OpenGameArt.org]. Consultat el 15 de novembre de 2022. https://ankousse26.itch.io/free-flag-with-animation
- Aron Granberg (2012). A* Pathfinding Project. [arongranberg.com]. Consultat el 20 novembre de 2022. https://arongranberg.com/astar/
- Battle Of The Dragons [Cançó]. [Pixabay]. Consultat el 20 juliol de 2023. https://pixabay.com/es/music/titulo-principal-battle-of-the-dragons-8037/
- Bonsaiheldin. (2016). 3 [PNG]. 16x16 píxels. [itch.io]. Consultat el 25 novembre de 2022. https://opengameart.org/content/gold-treasure-icons-16x16
- Bonsaiheldin. (2016). 4 [PNG]. 16x16 píxels. [itch.io]. Consultat el 25 novembre de 2022. https://opengameart.org/content/gold-treasure-icons-16x16
- Bonsaiheldin. (2016). 5 [PNG]. 16x16 píxels. [itch.io]. Consultat el 25 novembre de 2022. https://opengameart.org/content/gold-treasure-icons-16x16
- Coins 2 [Clip]. [Samplefocus]. Consultat el 20 juliol de 2023. https://samplefocus.com/samples/coins-2
- Dagger woosh [Clip]. [mixkit]. Consultat el 20 novembre de 2023. https://mixkit.co/free-sound-effects/sword/
- Drummyfish. (2018). house [PNG]. 144x83 píxels. [OpenGameArt.org]. Consultat el 15 de novembre de 2022. https://opengameart.org/content/top-down-rpg-mockup-scene

- Drummyfish. (2018). house2 [PNG]. 89x132 píxels. [OpenGameArt.org]. Consultat el 15 de novembre de 2022. https://opengameart.org/content/top-down-rpg-mockup-scene
- Drummyfish. (2018). house3 [PNG]. 60x92 píxels. [OpenGameArt.org]. Consultat el 15 de novembre de 2022. https://opengameart.org/content/top-down-rpg-mockup-scene
- Game Endeavor. (2021). plains [PNG]. 96x192 píxels. [itch.io]. Consultat el 20 d'octubre de 2022. https://game-endeavor.itch.io/mystic-woods
- Game Endeavor. (2021). fences [PNG]. 64x64 píxels. [itch.io]. Consultat el 20 d'octubre de 2022. https://game-endeavor.itch.io/mystic-woods
- Kings of camelot [Cançó]. [Pixabay]. Consultat el 20 juliol de 2023. https://pixabay.com/es/music/titulo-principal-knights-of-camelot-8038/
- KL Music Box Game Over II [Cançó]. [Pixabay]. Consultat el 20 juliol de 2023. https://pixabay.com/es/sound-effects/kl-music-box-game-over-ii-152200/
- Llei orgànica 3/2018, de 5 de desembre, de protecció de dades personals i garantia dels drets digitals. (2018, 5 de desembre). Boletín Oficial del Estado, núm. 294, pp. 10907-10990.
- Ough! [Clip]. [Pixabay]. Consultat el 20 novembre de 2023. https://pixabay.com/es/sound-effects/ough-47202/
- Unity Technologies. Unity Manual. Consultat el 10 d'octubre de 2022. https://docs.unity3d.com/Manual/index.html
- Unity Technologies. Unity Forum. Consultat el 10 d'octubre de 2022. https://forum.unity.com/
- Unity Technologies. Unity Answers. Consultat el 10 d'octubre de 2022. https://answers.unity.com
- Unity Technologies. Input System. Consultat el 15 d'octubre de 2022. https://docs.unity3d.com/Packages/com.unity.inputsystem@1.0/manual/index.html
- Unity Technologies. About Localization. Consultat el 10 juliol de 2023. https://docs.unity3d.com/Packages/com.unity.localization@1.0/manual/index.html

11.Manual d’usuari i d’instal·lació

11.1 Instal·lació

Per tal d'instal·lar l’aplicació haurem de seguir els passos següents:

- Posar l’arxiu APK situat a la carpeta de builds dins un dispositiu mòbil Android.
- Haurem d’activar la configuració de poder instalar de fonts no segures, ja que no ens hem donat d’alta per poder penjar l’aplicació a la Play Store.
- Un cop instalada la podem obrir com qualsevol altra aplicació de dispositiu mòbil.

11.2 Manual d’usuari

11.2.1 Controls

Els controls del joc són:

- Amb el joystick virtual esquerra es pot moure el personatge.
- Amb el botó virtual dret es pot atacar.
- Quan ens acostem a una estructura interaccionable (forja, ajuntament o caserna) apareixeran uns botons virtuals per realitzar diferents opcions. Prémer el botó corresponent a l'opció que es vol dur a terme. Com per exemple, pujar l'atac del jugador, curar-se, recuperar defenses, etc.

11.2.2 Objectiu del joc

L’objectiu del joc és aguantar tantes onades d’enemics com sigui possible mentre vas millorant el teu personatge i les defenses de la base.

Per aconseguir-ho, haurà de pensar com fer ús de les millores disponibles a la forja i el reclutament d’ajudants de la caserna.